# Application Note



# Arrowhead Client on ZynqBerry Device Installation for Ubuntu 16.04 LTS

Lukáš Kohout
*kohoutl@utia.cas.cz*

## Revision history

| Rev. | Date | Author | Description |
|---|---|---|---|
| 0 | 25.03.2019 | L. Kohout | Initial version |
| 1 | | | |
| 2 | | | |
| | | | |

## Contents




## Acknowledgement

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

# 1 Introduction

This application note describes an installation procedure of Arrowhead client on Zynq 7000 device. The concrete board is ZynqBerry, which is Zynq-7010 in Raspberry Pi form factor board from Trenz Electronic [1]. The Zynq part consists of dual core 32-bits ARM Cortex-A9 processor and programmable logic in a single chip. The device runs Xilinx PetaLinux 2018.2 kernel with Debian 9.8 Stretch distribution (03.25.2019). The client acts as a *Producer* of a service or as a *Consumer* requesting the service. The base hardware platform for the Zynq device is compiled with Xilinx Vivado 2018.2 tool. The entire installation procedure has been tested on Ubuntu 16.04 LTS host. To run and test Arrowhead client, it is required to have running Arrowhead services.

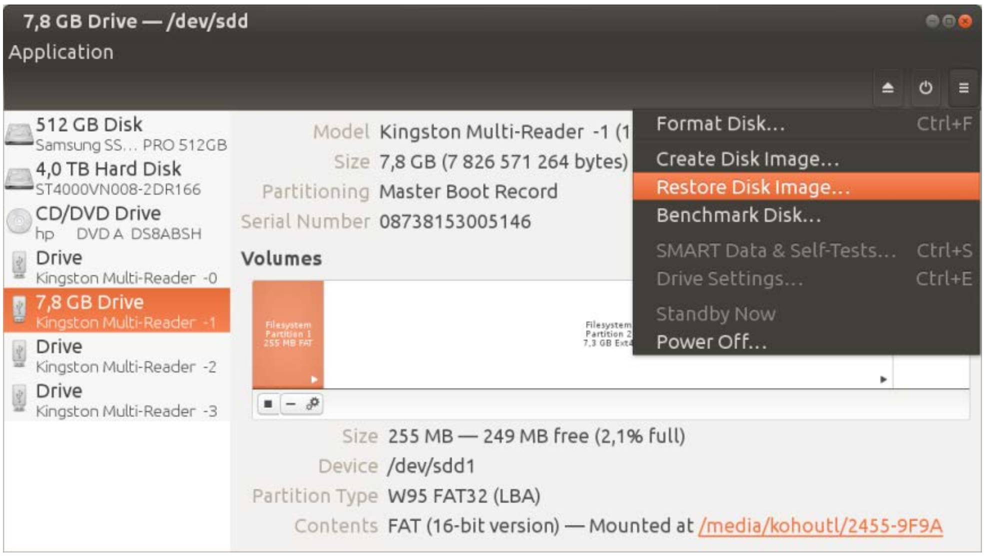
# 2 Arrowhead Services

Testing and running the Arrowhead client require running Arrowhead services [2]. It is recommended to use prepared image for Raspberry Pi 3 (RPi3). It includes Raspberian linux distribution with already installed and configured Arrowhead framework G4.0 lightweight implementation. The image is available as result 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 *ownClowd* repository in section WP1, task 1.4. Please contact coordinator of the consortium for further information about access to the image. The image is zipped to 3 files *Arrowhead-40-raspi.z01*, *Arrowhead-40-raspi.z02* and *Arrowhead-40-raspi.zip*. To write image to the SD card follow these steps:

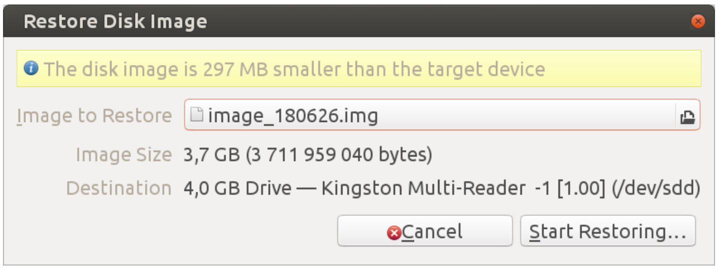
1. Unzip three downloaded zip files into one image file. The name of the image file is *image_180626.img*.
2. Insert a micro SD card to the reader. The minimal size of the card should be 4 GB and the card should be as fast as possible (class 10 for example). The speed of the card strongly affects the speed of the running system as well as the time needed to write the image to the card.
3. Write the image to the card. For the purpose, there is a tool within the Ubuntu distribution called *gnome-disks*. From the PC terminal execute:

```
gnome-disks
```

4. In the left column of the tool, select the drive corresponding to the inserted card. From the menu of the tool, select *Restore Disk Image…*



5. Select the image file to write (*image_180626.img*), and click on *Start Restoring…* button. Confirm the restoration with your superuser password.




6. After writing is finished, plug the card to the RPi3 board.
7. Power the board on, the power supply is provided via micro USB cable. Connect it to the PC or use a power adapter.
8. RPi3 provides its screen using HDMI connector, the resolution is 1920x1080p60. It can be controlled with USB keyboard and mouse. The user is *pi* and password is *raspberry*.
9. Connect the ethernet cable providing an internet connectivity and DHCP server.
10. To get the RPi3 board IP address use its terminal via HDMI screen and keyboard, execute command:

```
ifconfig
```

It returns the listing similar to this:

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.42.0.141  netmask 255.255.255.0  broadcast 10.42.0.255
        inet6 fe80::ba27:ebff:fe6d:80eb  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:6d:80:eb  txqueuelen 1000  (Ethernet)
        RX packets 77  bytes 9143 (8.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 92  bytes 16200 (15.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Alternatively, on user host machine execute:

```
sudo arp-scan --interface=eth0 –localnet
```

where *eth0* is the ethernet interface of PC connecting subnet with ZynqBerry device. This command returns all IP addresses within the subnet.

11. To shutdown the RPi3 properly use command:

```
sudo halt
```

# 3 Hardware

The hardware is compiled with Xilinx Vivado 2018.2 tool, the design is based on a board support package provided by Trenz Electronic for ZynqBerry board.

1. Download the board support package for Xilinx tools in version 2018.2 from the Trenz Electronic web page, choose package called *zynqberrydemo1*: http://www.trenz-electronic.de/fileadmin/docs/Trenz_Electronic/Modules_and_Module_Carriers/special/TE0726/Reference_Design/2018.2/zynqberrydemo1/te0726-zynqberrydemo1-vivado_2018.2-build_03_20181120163939.zip.
2. Unpack the package, it will create *zynqberrydemo1* folder.

NOTE: Inside the package there are bash scripts without permissions for execution. Before using them, change their permissions by command:

```
chmod ugo+x sript_name
```

3. On PC, open linux terminal window, go to the *zynqberrydemo1* folder and create an initial setup:

```
cd zynqberrydemo1
chmod ugo+x _create_linux_setup.sh
./_create_linux_setup.sh
```

It will create 3 scripts: *design_basic_settings.sh*, *vivado_create_project_guimode.sh* and *vivado_open_existing_project_guimode.sh*.

4. Select the board version you own, in our case it is *te0726-03m* (see the content of the *zynqberrydemo1/board_files/TE0726_board_files.csv* file).

In *design_basic_settings.sh* script locate the line containing

```
PARTNUMBER= LAST_ID
```

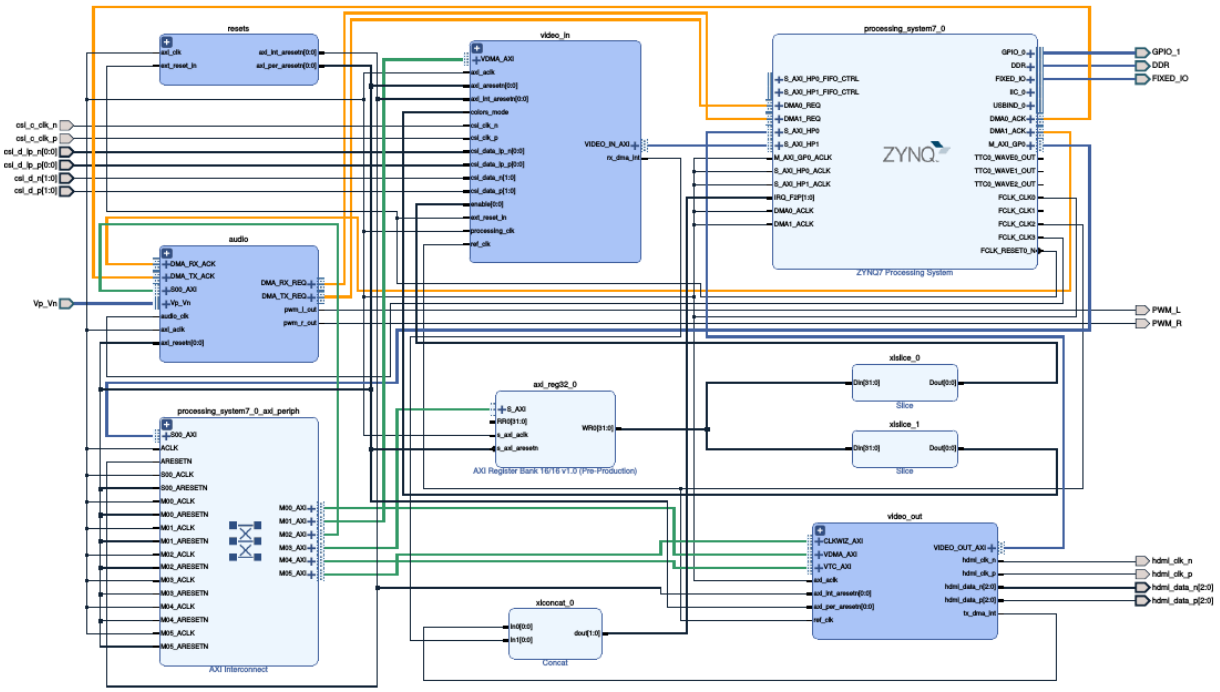
and change it to

```
PARTNUMBER=3
```

5. Start the Xilinx Vivado 2018.2 and create the design, use the script:

```
chmod ugo+x design_basic_settings.sh
chmod ugo+x vivado_create_project_guimode.sh
./vivado_create_project_guimode.sh
```

The figure shows block design of the created system.



6. Build the design, use TCL script provided within the board support package. From the Vivado TCL console execute command:

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

After the compilation, a hardware description file (HDF) will be located in folder *zynqberrydemo1/prebuilt/hardware/m*.

NOTE: Keep the Vivado tool running for the future steps.

# 4 PetaLinux 2018.2

Modify the PetaLinux 2018.2 distribution to have kernel image and its file system on separate partitions of the SD card.

1. On PC 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
```

2. Go to the folder with PetaLinux, it already contains a prepared configuration according to ZynqBerry board requirements.

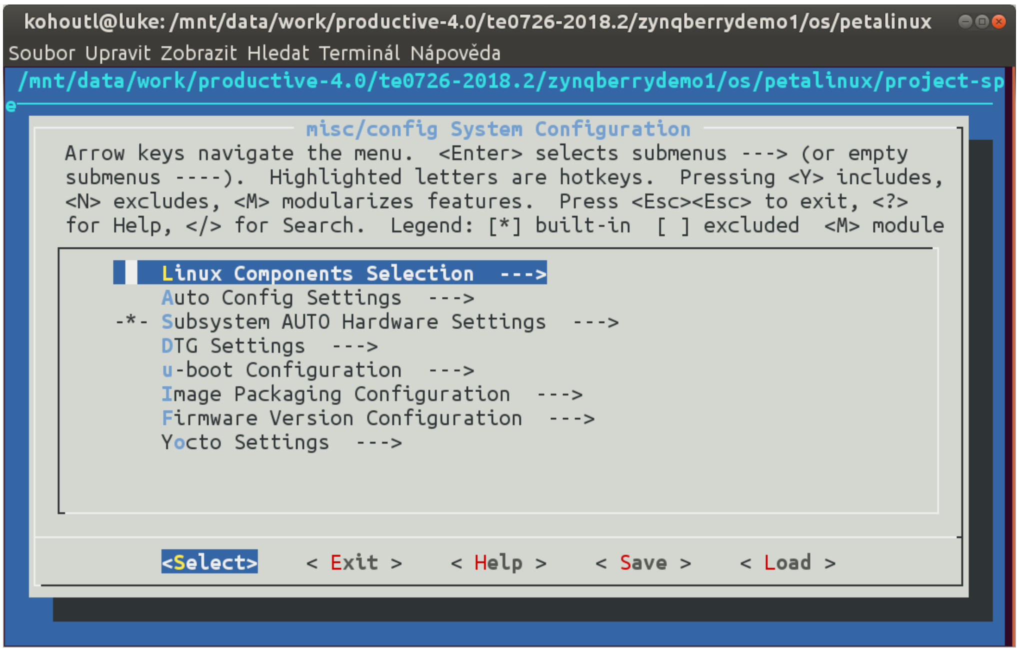
```
cd zynqberrydemo1/os/petalinux
```

3. Copy precompiled HDF to the *zynqberrydemo1/os/petalinux* folder.

```
cp -f ../../prebuilt/hardware/m/zynqberrydemo1.hdf .
```

4. Load the HDF to current PetaLinux configuration.

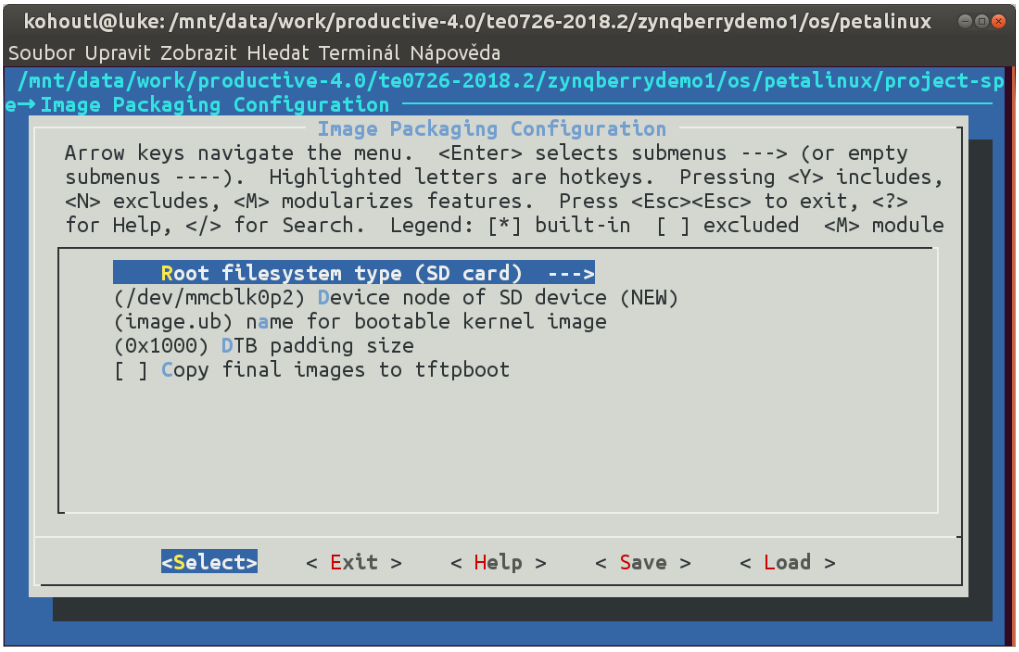
```
petalinux-config --get-hw-description . -p .
```




5. Change the PetaLinux filesystem location from the ramdisk to the extra partition on the SD card, select:

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

Optionally, the option *Copy final images to tftpboot* can be switched off.
Leave the configuration, 3x *Exit* and *Yes*.




6. Build PetaLinux, from the bash terminal execute

```
petalinux-build
```

7. Copy *image.ub* and *u-boot.elf* files to the zynqberrydemo1/prebuilt/os/petalinux/default folder

```
cp -f images/linux/image.ub ../../prebuilt/os/petalinux/default/image.ub
cp -f images/linux/u-boot.elf ../../prebuilt/os/petalinux/default/u-boot.elf
```

# 5 Debian for ARM

The file system is based on the latest stable version of Debian 9.8 Stretch distribution (03. 25. 2019). The precompiled image file can be found in the *te0726-arrohead-client/debian/te0726-debian.img.zip* file. For those who want to create their own image follow the steps below, otherwise skip this section a go to directly to section 6.

1. From the package *te0726-arrowhead-client/debian* copy *mkdebian.sh* file to the PetaLinux folder.

```
cp –f te0726-arrohead-client/debian/mkdebian.sh \
      zynqberrydemo1/os/petalinux/mkdebian.sh
```

2. Go to the folder with PetaLinux:

```
cd zynqberrydemo1/os/petalinux
```

3. Debian image is created with *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
```

Next table summarizes all the tools with a corresponding package name.

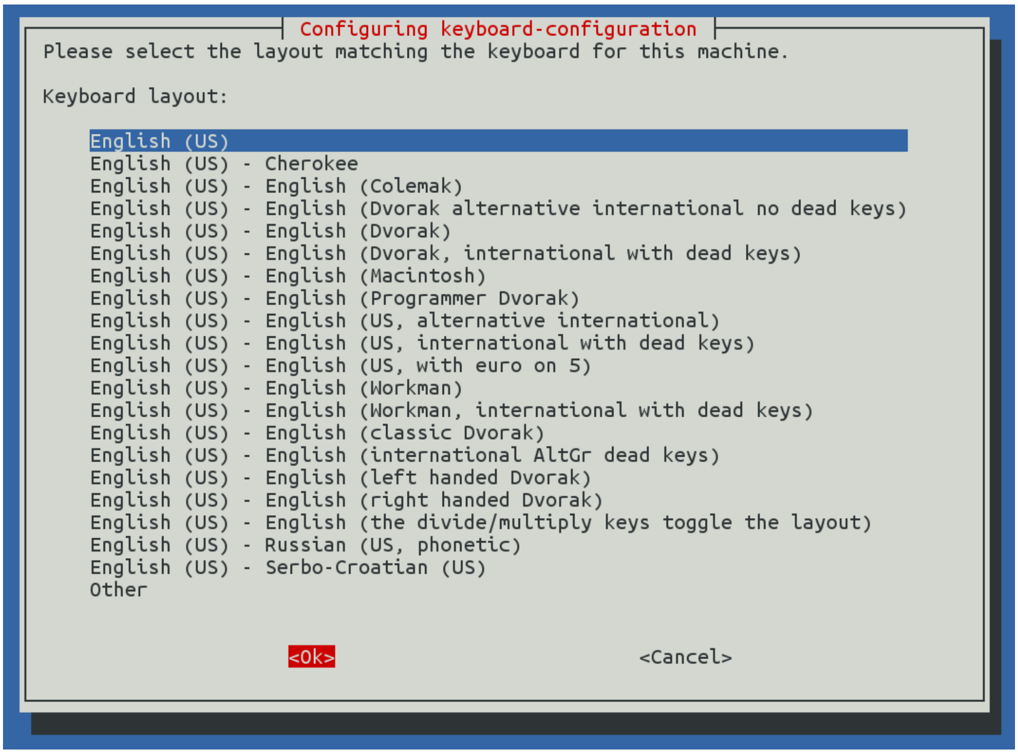
| Tool | Package |
|---|---|
| dd | coreutils |
| losetup | mount |
| parted | parted |
| lsblk | util-linux |

| Tool | Package |
|---|---|
| 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.

```
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 much 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.

# 6 SD Card

1. Insert a micro SD card to the reader. The minimal size of the card should be 8 GB and the card should be as fast as possible (class 10 for example). The speed of the

card strongly affects the speed of the running system (PetaLinux kernel with Debian) as well as the time needed to write the image to the card.

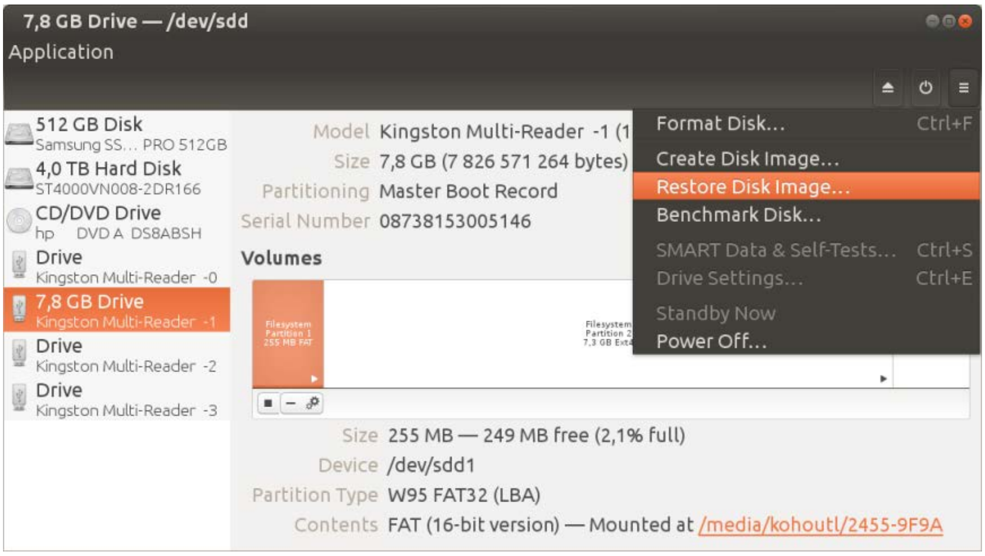
2. On PC go to the folder with PetaLinux:

```
cd zynqberrydemo1/os/petalinux
```

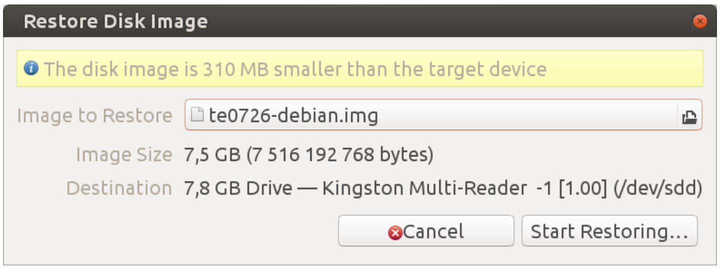
3. Write the image to the card. For the purpose, there is a tool within the Ubuntu distribution called *gnome-disks*.

```
gnome-disks
```

4. In the left column of the tool, select the drive corresponding to the inserted card. From the menu of the tool, select *Restore Disk Image…*




5. Select the image file to write, *zynqberrydemo1/os/petalinux/te0726-debian.img* and click on *Start Restoring…* button. Confirm the restoration with your superuser password.




6. After writing is finished, plug the card to the ZynqBerry board.

# 7 BOOT.bin

1. Go back to the Vivado tool.
2. Create *BOOT.bin* file. From the Vivado TCL console execute command:

```
TE::sw_run_hsi
```

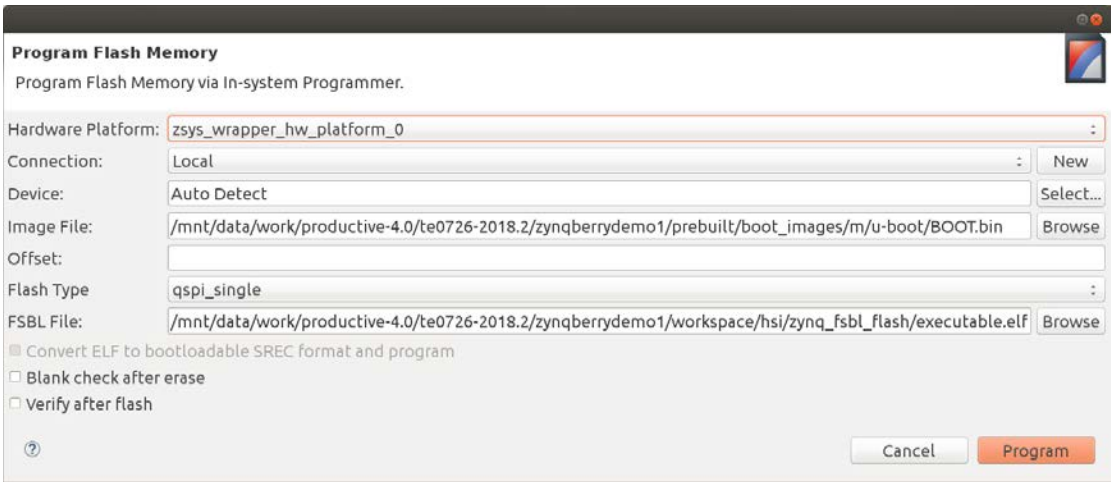
The resultant file will be located in *zynqberrydemo1/prebuilt/boot_images/m/u-boot/* folder.

3. Start Xilinx SDK. From the Vivado TCL console execute command:

```
TE::sw_run_sdk
```

4. Connect ZynqBerry board with your PC via micro USB cable. It provides the power supply and the programming interface.

5. Write *BOOT.bin* file to ZynqBerry FLASH memory to enable booting the board. From the SDK menu select *Xilinx→Program Flash*. Select image to write, browse to *zynqberrydemo1/prebuilt/boot_images/m/u-boot/BOOT.bin*. Select FSBL file, browse to *zynqberrydemo1/workspace/hsi/zynq_fsbl_flash/executable.elf*. Click on *Program* button.




6. Close SDK, close Vivado.

# 8 Start ZynqBerry

1. Connect or reconnect ZynqBerry board with your PC via micro USB cable. ZynqBerry starts its booting sequence.

2. To see the booting sequence or to control ZynqBerry device, use the serial terminal connected via micro USB cable. On PC, there can be used *putty* tool for instance. The settings of the serial terminal are in the table below.

| Parameter | Value |
|---|---|
| Speed | 115200 |
| Data bits | 8 |
| Stop bits | 1 |
| Parity | None |
| Control Flow | None |

3. Login: root/root

4. Get or set an IP address of the ZynqBerry.

   a. In case the ZynqBerry board is connected to the network that provides DHCP (preferred), from the serial terminal command line execute:

   ```
   ifconfig
   ```

It returns the listing similar to this:

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.42.0.103 netmask 255.255.255.0 broadcast 10.42.0.255
inet6 fe80::20a:35ff:fe00:1e53 prefixlen 64 scopeid 0x20<link>
ether 00:0a:35:00:1e:53 txqueuelen 1000 (Ethernet)
```

Or on user host machine execute:

```
sudo arp-scan --interface=eth0 –localnet
```

where *eth0* is the ethernet interface of PC connecting subnet with ZynqBerry device. This command returns all IP addresses within the subnet.

b. When DHCP is not provided set the local address manually. From the serial terminal command line execute:

```
ifconofig eth0 10.42.0.103
```

5. Use SFTP to copy installation script to the ZynqBerry, from PC command line execute:

```
cd te0726-arrohead-client/zynq
sftp root@10.42.0.103 <<< $'put install-arrohead-cli-dep.sh'
```

The script will be copied to */root/* folder on the ZynqBerry board.

6. Install dependencies required by the Arrowhead client compilation, from the ZynqBerry command line (SSH or serial terminal) execute:

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

7. Zynqberry provides the Debian desktop screen using HDMI connector, the resolution is 1280x720p60. It can be controlled with USB keyboard and mouse. To start the desktop, execute from the terminal:

```
startx&
```

8. To properly shutdown the Zynqberry board, use command *halt* before the power off. It avoids unfinished writes to the filesystem on the SD card.

# 9 Arrowhead Provider on ZynqBerry

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

1. Get the arrowhead client source codes. 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 *Provider*.

```
cd client-cpp/ProviderExample
make
```

3. Configure the *Provider*, the name of the configuration file is *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 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 *Provider*

```
./ProviderExample
```

The *Provided* registers itself in the Arrowhead framework database, on *Consumer* request, it returns artificial temperature, it is fixed value 26 degrees of Celsius.

# 10 Arrowhead Consumer on ZynqBerry

For the testing purpose, the Arrowhead *Consumer* can be compiled and run from the same ZynqBerry device as the *Provider*.

1. Compile Arrowhead *Consumer*.

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

2. Configure the *Consumer*, 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.
- requestedService/serviceMetadata/security – change string "token" to empty string "".
- 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:

```
{
  "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 editor (F10).

3. Run the *Consumer*

```
./ConsumerExample
```

The program should show the response from the *Provider* and exit.

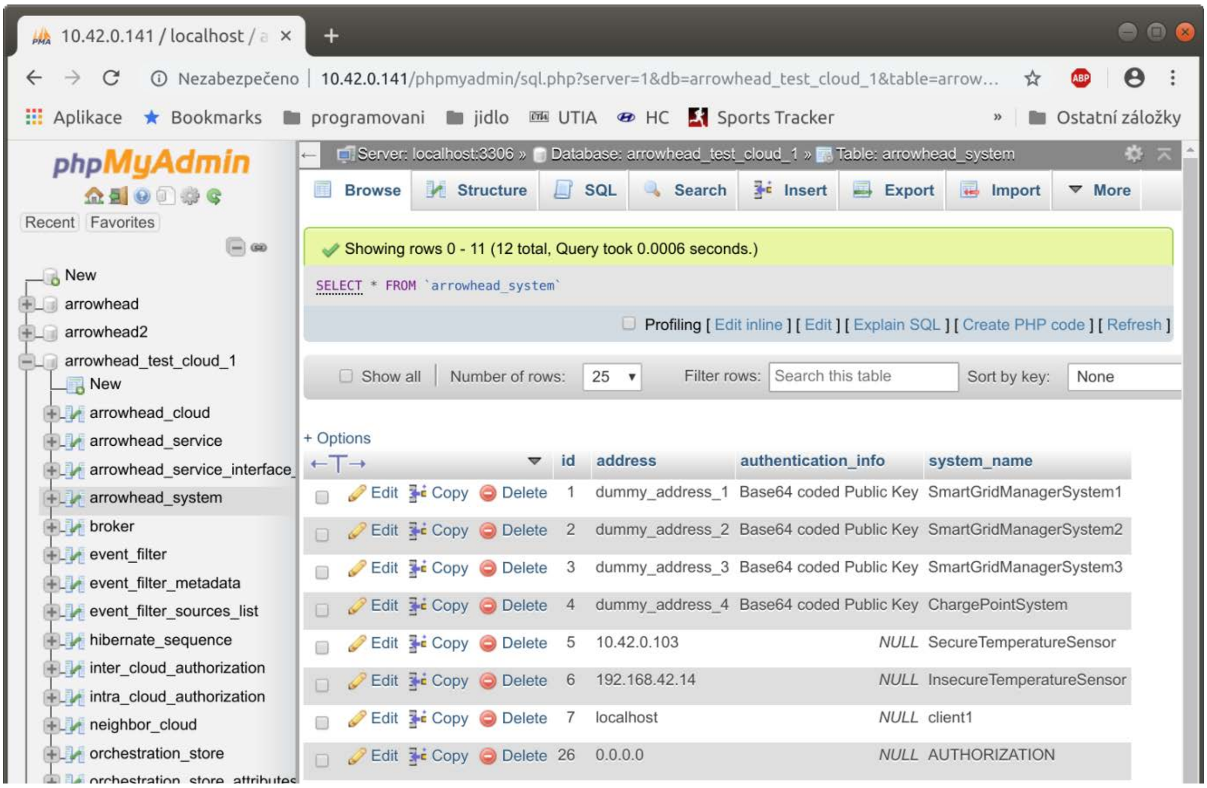
```
Provider Response:
{"e":[{"n": "this_is_the_sensor_id","v":26.0,"t": "1553675692"}],"bn":
"this_is_the_sensor_id","bu": "Celsius"}
```

If it fails, the database of the Arrowhead framework has to be modified. To fix it, follow steps in Section 11.

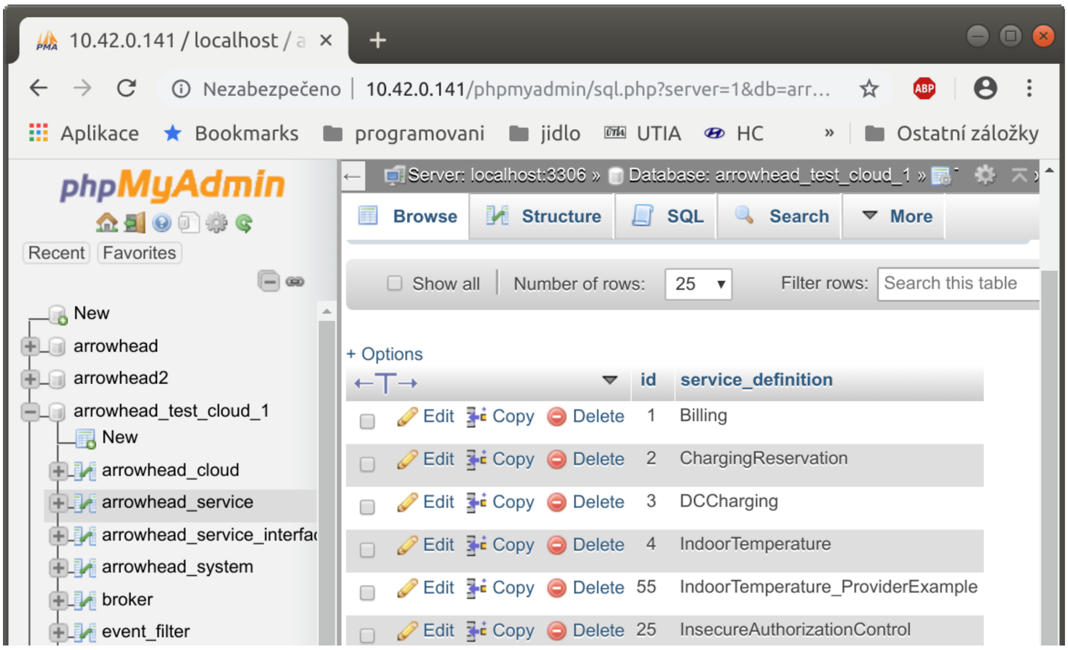
## 11 Arrowhead Database

The Arrowhead framework running on RPi3 provides *phpMyAdmin* to control its database. To allow the *Consumer* to get the *Producer* service response, follow next steps.

1. On your PC start web browser and go to RPi3 *phpMyAdmin* web page, *http://10.42.0.141/phpmyadmin* (use IP address of your RPi3). User name is *root*, password is *root*.
2. Get an ID of the *Producer*. 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 *Consumer*. Select table *arrowhead_test_cloud_1→ arrowhead_system* and locate the line containing system_name *client1*. In our case it is 7.

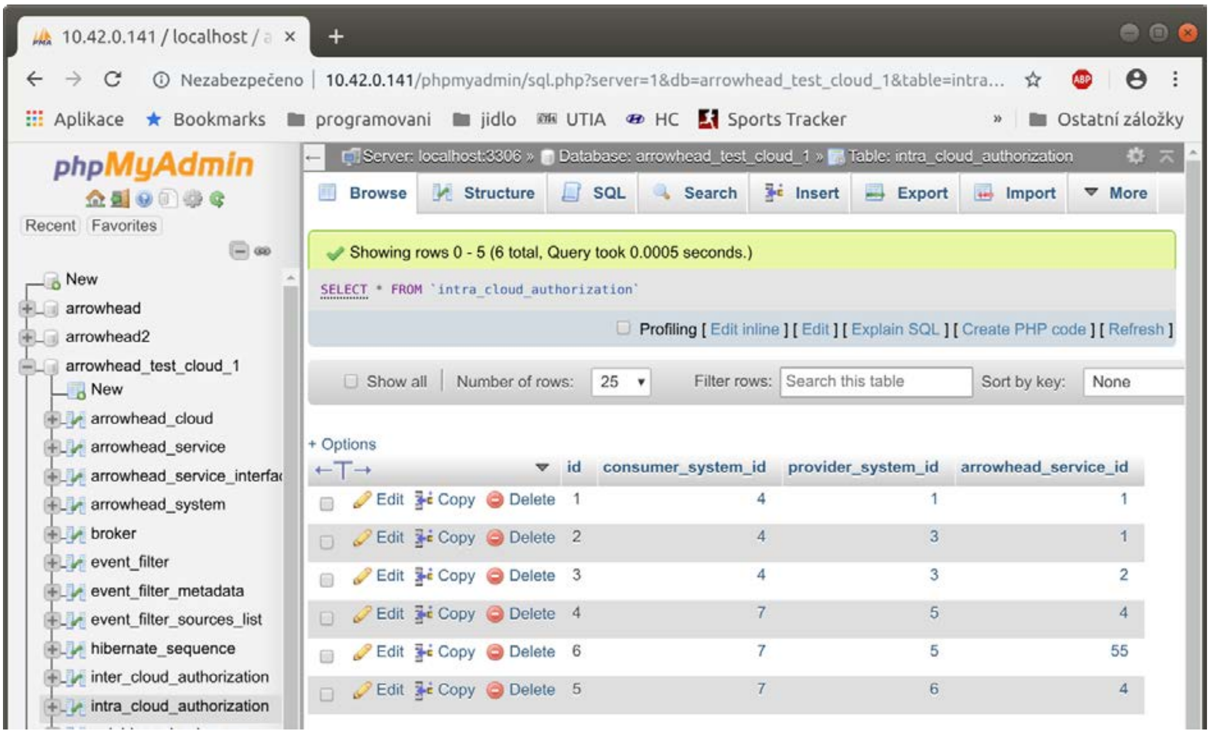



4. Get an ID of the *Producer* service. Select table *arrowhead_test_cloud_1→ arrowhead_service* and locate the line containing service_definition called *IndoorTemperature_ProviderExample*. In our case the ID is 55.




In table *service_registry* it can be checked, that the *Provider* is linked with its service.

5. Link the *Provider*, its service and the *Consumer* together. In table *intra_cloud_authorization* add a new line containing *consumer_system_id* 7, *provider_system_id* 5 and *arrowhead_service_id 55*. Now the *Consumer* should get the proper response from the *Provider*.



## 12 Package content

```
.
├── debian
│   ├── mkdebian.sh
│   └── te0726-debian.img.zip
└── zynq
    └── install-arrohead-cli-dep.sh
```

## 13 References


[1]. Trenz Electronic, "TE0726 TRM," [Online].
Available: https://wiki.trenz-electronic.de/display/PD/TE0726+TRM

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

## 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.