acontis technologies GmbH

SOFTWARE

Hypervisor

User's Manual

Version 1.0

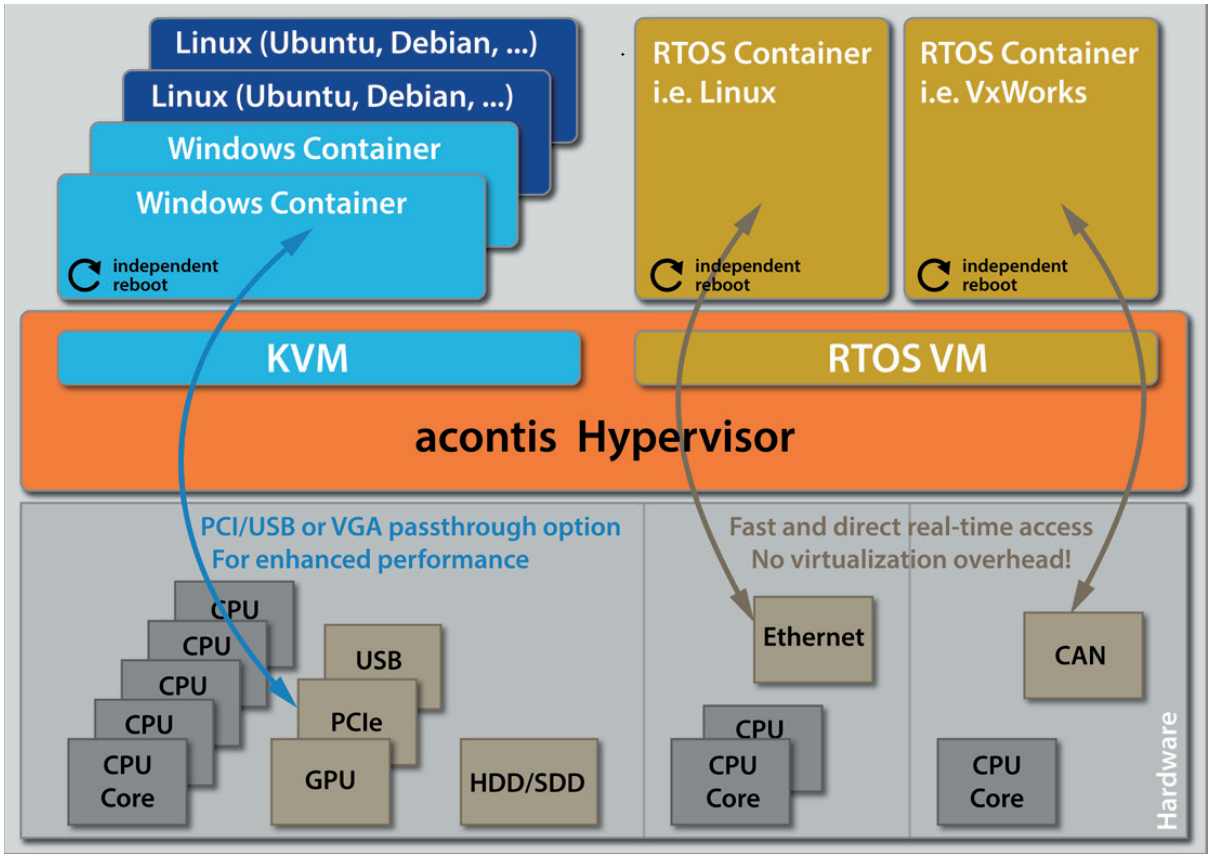
Edition: November 3, 2022

Contents

1 Introduction

1.1 Overview

Using the existing, industry proven acontis real-time RTOS-VM virtualization technology, multiple hard real-time operating systems (Real-time Linux, VxWorks, etc.) can run in native speed. In addition, based on the well-known and proven KVM virtualization solution, multiple standard operating systems like Windows and Linux (Ubuntu, Debian, Fedora, etc.) operate in parallel. Besides virtual hardware, KVM provides para-virtualization, PCI/USB and VGA pass-through for highest possible performance. Each guest OS is fully independent and separated and can be rebooted or shutdown while the other guests continue without being affected.



RTOS Virtual Machine Hypervisor

The RTOS Virtual Machine hypervisor technology provides an independent layer to run any RTOS in native speed. No virtualization overhead is introduced and all RTOS drivers as well the operating systems and applications have direct and fast hardware access. The same technology is successfully used by customers all over the world since more than 10 years in the existing acontis real-time solution. The new acontis Hypervisor can utilize this technology without modification so existing customer applications can be re-used without modification.

KVM Hypervisor

KVM is one of the most popular Type 1 hypervisors used in many high-availability and security critical environments like the cloud solutions from Google, Amazon or Oracle.

To increase performance, for example for fast USB, Ethernet or Graphics operation, the respective devices can be completely passed through to Windows or Linux guests. Alternatively, para-virtualized devices for the hard disk, the Ethernet or graphics controller reduce the amount of necessary hardware without compromising throughput.

Key technical features

The acontis Hypervisor is a perfect symbiosis between the wide-spread KVM virtualization solution and the industry proven RTOS Virtual Machine hypervisor for real-time operating systems.

General

- Supports Multiple OSes: real-time Linux, On Time RTOS-32, VxWorks® RTOS, Standard Linux, Microsoft® Windows®, proprietary Roll-your-own, Bare metal, any unmodified OS
- RTOS containers including applications run on bare metal core with no virtualization overhead and direct hardware access
- Fully separated and independent guest operation
- User defined guest startup sequence
- Utilize any number of CPU cores per single guest
- Independent reboot of all guests while others continue operation
- Virtual Network between all guests
- Inter-OS Communication: Shared Memory, Events, Interlocked data access, Pipes, Message Queues and Real-time sockets for high speed application level communication
- Hypervisor provided fileserver for all guests

KVM

Windows and Standard Linux operating systems like Ubuntu or Debian run under control of the KVM hypervisor. This hypervisor provides plenty of sophisticated features:

- Multiple Windows and/or standard Linux instances
- Windows/Linux containers with snapshot support to easily switch between different application situations without the need to install multiple OS instances. Snapshots create a view of a running virtual machine at a given point in time. Multiple snapshots can be saved and used to return to a previous state, in the event of a problem.
- Pass-through support: To increase performance, for example for fast USB, Ethernet or Graphics operation, the respective devices can be completely passed through to Windows or Linux guests.
- Paravirtualization: Para-virtualized devices for the hard disk, the ethernet or graphics controller reduce the amount of necessary hardware without compromising throughput.
- Graphics virtualization to provide 3D accelerated graphics to multiple guests.

RTOS-VM

- Multiple real-time Operating Systems (Linux, VxWorks, On Time RTOS-32, etc.)
- Fast real-time interrupt handling and short thread latencies
- Direct hardware access with no virtualization overhead
- Compatible with the existing acontis Windows real-time extension (applications can be shared or cross-migrated between both solutions)

1.2 Linux for Windows users

The following points may help a Windows user with the first Linux steps.

- Windows Command Prompt is called `Terminal` in Linux. The shortkey to open one is `Ctrl` + `Alt` + `T`.
- Instead of `notepad.exe` you can use `gedit`. If you have no graphical desktop available `nano` will work.
- `cd \MyDirMySubDir` is `cd /MyDir/MySubDir` and `cd..` must be `cd ...`
- `dir` is `ls` and `ls -l` will show additional information.
- `run as administrator` has its equivalent in `sudo`, which has to be put before a command. For example `sudo gedit` starts the editor as administrator (called `root` in Linux). You can use `sudo -s` to switch console to root user and `exit` to return.
- Most programs give helpful information being called with parameter `--help` or by calling the manual `man`. In case of `gedit` this would be `gedit --help` and `man gedit`.
- `chmod` can change the rights (*read/write/executable*) of a file. In Windows you just create a file `test.bat` and you can already execute it. In Linux you name it `test.sh`, but it won't run without being made an `executable`.
- `chown` can change the owner of a file.

2 System Setup

2.1 Hardware Requirements

Ressource	Recommended
CPU	Minimum: • `1` Core for the host, which can be shared with non-real-time OS • `+1` Core for *each* real-time OS. • `+1` Core for *each* non-real-time OS if **not** shared with host. **Important:** Do **not** enable Hyperthreading. **Important:** CPU **must** support `VT` technology!
Memory	`4GB` for host and RTOS + minimum `4GB` for Windows (as a guest).
GPU	
2nd GPU	
Disk Space	`50GB` for host + minimum `50GB` for Windows (as a guest).

Hint: For installation purposes is it recommended to use an USB stick with at least `4GB`. Further informations could be found at the quickstart guide!

2.2 Installation and Configuration

Hint: The hypervisor could be installed either exclusively on a PC or side by side with an existing OS. The **default** case is exclusively. The side by side installation is described at next chapter *Side By Side Installation*!

2.2.1 Side By Side Installation

If the hypervisor has been installed in parallel to Windows or Linux, the file `setrootuuid.sh` has to be adjusted. The original `defaultBoot` values need to be incremented by 1, e.g. set to `3` and `3`, respectively.

Enter `gedit setrootuuid.sh` to adjust these values, they are near the end of the file:

```
# update default boot configuration
###################################
defaultbootRE='\(GRUB_DEFAULT=\)[0-9]*'
[ -d /sys/firmware/efi ] && defaultBoot=3 \|\| defaultBoot=3 # 2 for UEFI,␣
↪2 for Legacy BIOS
```

Caution: The above numbers (2, 3) may change depending on the hypervisor version!

Restart the system by entering:

```
$ sudo reboot
```

Attention: Start/Stop of the guests after installation is described in chapter *Run shipped RTOS container*!

2.3 Hypervisor Files and Directories

2.3.1 Directories

Basic configuration settings including helper scripts are stored in directory `/hv/config`.

Directory	SubDir	Content
`\hv\`	`.`	Root dir
	`config\.`	Contains all global configuration files
	`hvctl\.`	Contains all global configuration files
	`VMs\vm[]\.`	Contains the vm guests 1..n
`\hv\`	`lx\.`	Contains the LxWin files, scripts and executables
	`rtos-32\.`	Contains the RTOS-32 files, scripts and executables
	`vx\.`	Contains the VxWorks files, scripts and executables

2.3.2 Configuration Files

Basic configuration settings including helper scripts are stored in directory `/hv/config`.

The following table shows the locations of the main `.config` files of the supplied rtos and demos.

Directory	SubDir	.config file	Description
`\hv\`	`lx\`	`hv.config`	Main configuration file to start the RT-Linux RTOS
	`rtos-32\`	`rtos-32demo.config`	Contains the RTOS-32 demo configuration file
		`realtimedemo.config`	Contains the realtime demo configuration file
		`ecmasterdemo.config`	Contains the EC-Master demo configuration file
	`vx\`	`vxworks.config`	Contains the VxWorks main configuration

2.3.3 Brand Labeling

In case if you need to do brand labeling the shipped product, here is the list of image files for wallpapers, boot logos etc

Directory	SubDir	graphic file	Description
`\usr\share\`	`plymouth\themes\xubuntu-logo`	`logo.png` `logo_16bit.png`	OS Boot logo
	`xfce4\backdrops`	`xubutnu-jammy.png`	Background image
`\etc\`		lsb-release	Contains name RTOSVisor

2.4 Additional device handling possibilities to RTOS

2.4.1 Remove device (from the RTOS)

You should remove include of `<RTOS device name>.config` from the main configuration file (`/hv/vx/vxworks.config`, etc).

Remove reference of `<RTOS device name>.sh` file in `/hv/config/hvpart.sh`.

After system restart device will be automatically used by the host. If you want to return the device to the host without rebooting you should call `<RTOS device name>.sh` script with parameter delete:

```
$ rtos_eth.sh delete
```

Now you can delete `<RTOS device name>.sh` and `<RTOS device name>.config` files.

2.5 USB device access for RT-Linux guests

A single USB device can be accessed from within RT-Linux using usbip.
This will expose a USB device from the hypervisor host to the RT-Linux guest via the virtual network.

The standard Linux image shipped with the hypervisor does not support usbip.
You need to exchange this image by the separately provided `rtlinux515.x64-usbip.bin` image file.
Copy this image into folder /hv/lx and then set the appropriate link:

```
$ cd /hv/lx
$ rm rtlinux.bin
$ ln -s rtlinux515.x64-usbip.bin rtlinux.bin
```

In a first step you will need to determine the vendor and device ID of the USB device you want to expose.
Insert the USB device and execute the following commands.

```
$ cd /hv/hvctl
$ sudo ./hvusbip.sh -init
$ sudo ./hvusbip.sh -list
```

Hint: The call with the -init parameter is only required once.

A list of USB devices will be shown.

```
device list
***********
 - busid 3-2 (046d:c52b)
   Logitech, Inc. : Unifying Receiver (046d:c52b)

 - busid 3-3 (046d:c52f)
   Logitech, Inc. : Unifying Receiver (046d:c52f)

 - busid 3-5 (0e8d:0608)
   MediaTek Inc. : unknown product (0e8d:0608)

 - busid 3-8 (064f:2af9)
   WIBU-Systems AG : CmStick (HID, article no. 1001-xx-xxx) (064f:2af9)

 - busid 4-4 (18a5:0243)
   Verbatim, Ltd : Flash Drive (Store'n'Go) (18a5:0243)
```

In this example, we will expose the WIBU CmStick device to the guest.
According to the above output, its vendor device id is **064f:2af9**

Then you need to insert the vendor device ID into the respective configuration file.

```
$ cd /hv/config
$ gedit usbip_exposed_device.sh
```

Insert the appropriate vendor device ID.
Don't miss to remove the comment at the beginning of the line!

```
# one single device currently can be exposed to the RTOS
export vendev=064f:2af9  # WIBU CodeMeter USB dongle
```

Hint:

When starting RT-Linux, the `/hv/lx/usbip_gen_init.sh` script will dynamically create the `/hv/lx/rtfiles/usbip_init.sh` script.
This sub script will be called when RT-Linux is booted via the `/hv/lx/rtfiles/autostart.sh` script.

Caution: You must not adjust the `/hv/lx/rtfiles/usbip_init.sh` script manually, it will be overwritten when RT-Linux is started!

You may insert specific autostart activities when the RT-Linux guest starts.

```
$ cd /hv/lx
$ gedit usbip_gen_init.sh
```

At the very bottom an example of such autostart activity is shown. Please adjust the script `/hv/lx/usbip_gen_init.sh` accordingly.

```
# insert usbip specific autostart activities here
if [ $vendev == "064f:2af9" ]; then
    # echo "start codemeter daemon"
    echo "start-stop-daemon --start --quiet --chuid root --exec /usr/sbin/
↪CodeMeterLin" >>./rtfiles/usbip_init.sh
    echo "to verify CodeMeter status run 'cmu --cmdust'" >>./rtfiles/usbip_
↪init.sh
fi
```

Then, the usbip expose service needs to be enabled which will start the usbip server on the host:

```
$ sudo systemctl enable /hv/services/hv_usbip_expose.service
```

Finally you should start the service to make your changes effective. You may also reboot to do so.

```
$ sudo systemctl start hv_usbip_expose
```

Hint:

This service will run the script /hv/config/usbip_expose.sh which will finally expose the device.
You can run the command 'systemctl status hv_usbip_expose.service' to verify if exposing worked correctly.

Caution:

If the USB device is removed and re-inserted, it has to be exposed again.

Furthermore, in RT-Linux the appropriate steps (executed in `usbip_init.sh`) also have to be executed again.

After you have started the service, you can check if the device is correctly exposed - before starting RT-Linux. Connect the USB device and then run the following command.

```
$ usbip list -r 127.0.0.1
```

A list of exportable USB devices will be shown. The one that you have set before should be shown.

```
Exportable USB devices
======================
 - 127.0.0.1
        3-3: WIBU-Systems AG : CmStick (HID, article no. 1001-xx-xxx)␣
↪(064f:2af9)
           : /sys/devices/pci0000:00/0000:00:14.0/usb3/3-3
           : (Defined at Interface level) (00/00/00)
           :  0 - Human Interface Device / No Subclass / None (03/00/00)
```

Finally you can start RT-Linux, the USB device should be visible then. To verify this, you can run the following command.

```
$ usbip port
```

A list of imported USB devices will be shown. The one that you have set before should be shown.

```
Imported USB devices
====================
Port 00: <Port in Use> at Full Speed(12Mbps)
       WIBU-Systems AG : CmStick (HID, article no. 1001-xx-xxx) (064f:2af9)
       1-1 -> usbip://192.168.157.1:3240/3-3
           -> remote bus/dev 003/007
```

Hint:

To stop exposing USB devices, you have to comment the device in `/hv/config/usbip_exposed_device.sh`.

```
# one single device currently can be exposed to the RTOS
# export vendev=064f:2af9  # WIBU CodeMeter USB dongle
```

Then you should disable the service.

```
$ sudo systemctl disable hv_usbip_expose
```

2.5.1 Hypervisor host usbip low level services

The usbip low level services are used by the scripts and services of the Hypervisor host.
This background information may be helpful for debugging and diagnosis purposes.

Loading kernel modules:

```
$ sudo modprobe usbip_host
$ sudo modprobe usbip_core
```

Start the usbip daemon:

```
$ sudo usbipd -D
```

Show all USB connected devices:

```
$ usbip list -l
```

Expose a specific USB device:

```
$ sudo usbip bind -b busid
```

Show all currently exposed USB devices that are not in use:

```
$ usbip list -r 127.0.0.1
```

Stop exposing a specific USB device:

```
$ sudo usbip unbind -b busid
```

2.5.2 RT-Linux usbip low level services

The usbip low level services are used by the scripts and services of the RT-Linux guest.
This background information may be helpful for debugging and diagnosis purposes.

Loading kernel modules:

```
$ modprobe vhci-hcd
```

Show exposed devices:

```
$ usbip list -r 192.168.157.1
```

Attach exposed device:

```
$ usbip attach -r 192.168.157.1 -b "Device bus ID"
```

Show ports in use by attached devices:

```
$ usbip port
```

Detach a specific port:

```
$ usbip detach -p "port"
```

2.6 Network Forwarding from Windows to the RTOS

If the RTOS (or any other OS connected to the virtual network) shall be accessed via TCP/IP from a **single** external system, traffic can be forwarded to the virtual network. Execute the following steps to forward traffic from a specific external computer to the RTOS:

- enable network forwarding in the Hypervisor host:

  ```
  $ sudo sysctl -w net.ipv4.ip_forward=1
  ```

- determine the IP address of the Hypervisor host. You can use the `ifconfig` command to accomplish this.
- open a Command Window with Administrator rights on your Windows PC
- run the following command (replace AAA.BBB.CCC.DDD with the appropriate IP address of the Hypervisor host):

  ```
  route add 192.168.157.0 mask 255.255.255.0 AAA.BBB.CCC.DDD
  ```

Caution:

Assure the Default Gateway in the RTOS is set to the Hypervisor host virtual network IP address (192.168.157.1)!
For RT-Linux it is set by default. For other RTOS you need to check the RTOS documentation how to accomplish this.

2.7 Bridge virtual and physical network

If the RTOS (or any other OS connected to the virtual network) shall be accessed via TCP/IP from **any** external system, the virtual network and the respective physical network has to be bridged.

In the folder `/hv/hvctl` you can find the template configuration file `brvnetconfig.sh` for the bridge configuration. Note, the IP address of the virtual network inside the RTOS guest need to be adjusted appropriately, see below for more details.

- Create the bridge

First, start the RTOS to assure the virtual network is available.
Change into the guest directory (`/hv/hvctl`) and execute the `brvnetset.sh` script.

- **Remove the bridge**
  Change into the guest directory (`/hv/hvctl`) and execute the `brvnetclr.sh` script.

2.7.1 Bridge configuration

First step: determine, which network adapter should be bridged. Search for `<link>` entry and get the adapter name.

```
$ ifconfig -a
```

In this case it's `enp2s0`. The current `$IP$` address of `enp2s0` is `inet 172.17.10.53` and the network mask is `255.255.0.0`.

```
rtv@rtv-TEST:~$ ifconfig -a
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 172.17.10.53  netmask 255.255.0.0  broadcast 172.17.255.255
    inet6 2a02:590:801:2c00:7170:3747:f835:a1cb  prefixlen 64  scopeid 0x0
↪<global>
    inet6 fe80::fe6f:c5f8:c5cd:e3cd  prefixlen 64  scopeid 0x20<link>
    inet6 2a02:590:801:2c00:96b0:b8a:2c58:6c91  prefixlen 64  scopeid 0x0
↪<global>
    ether 90:1b:0e:18:c9:83  txqueuelen 1000  (Ethernet)
    RX packets 116751  bytes 22127837 (22.1 MB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 74453  bytes 551331072 (551.3 MB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp3s5: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
    ether 74:ea:3a:81:4b:1d  txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop  txqueuelen 1000  (Local Loopback)
    RX packets 201  bytes 14798 (14.7 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 201  bytes 14798 (14.7 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vnet0: flags=99<UP,BROADCAST,NOTRAILERS,RUNNING>  mtu 1500
    inet 192.168.157.1  netmask 255.255.255.0  broadcast 192.168.157.255
    ether 00:60:c8:00:00:00  txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 59  bytes 10381 (10.3 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Next step: determine the default gateway.

```
$ ip route ls
```

You will get an output like `default via 172.17.5.2 dev enp2s0 proto dhcp metric 100`.

Next step: determine the DNS server.

```
$ resolvectl status | grep "Current DNS Server"
```

You will get an output like `Current DNS Server: 172.17.5.9`.

Next step: Adjust `brvnetconfig.sh` with the detected values of `ifconfig`:

```
$ gedit /hv/hvctl/brvnetconfig.sh
```

Values:

- `netif="enp2s0"`
- `defaultgw="172.17.5.2"`
- `dns="172.17.5.9"`
- `vnetbrip="172.17.10.53"`
- `vnetbrnm="255.255.0.0"`
- `#vnetbrmac=` comment in and adjust value only if there are collisions with **'same'** `MAC-IDs` on the network.

```
#!/bin/bash

# Ethernet network interface to bridge with VM.
# ethernet interface to bridge with vnet
netif="enp2s0"

# Default gateway
# How to determine the default gateway:
#       Use the command ip route ls
#           default via 172.17.5.2 dev enp2s0 proto dhcp metric 100
#           172.17.0.0/16 dev enp2s0 proto kernel scope link src 172.17.
↪10.4 metric 100
#       The default gateway here is "172.17.5.2"
defaultgw="172.17.5.2"        # default gateway

# DNS server
# How to determine the default gateway:
#       Use the following command: resolvectl status | grep "Current DNS␣
↪Server"
#            Current DNS Server: 172.17.5.9
#       The DNS server here is "172.17.5.9"
dns="172.17.5.9"

# Bridge settings
# The bridge replaces the former network device used by the hypervisor to␣
↪connect to the network.
# See above results provided by the ifconfig -a command
#       enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
#               inet 172.17.10.53  netmask 255.255.0.0  broadcast 172.17.
↪255.255
#       In this example, the bridge IP address is 172.17.10.53 and the␣
↪network mask is 255.255.0.0
vnetbrip="172.17.10.53"
vnetbrnm="255.255.0.0"

# the values below are default values, typically not to be changed
vnetip="192.168.157.1"
vnetnm="255.255.255.0"
#vnetbrmac="54:52:00:ac:30:10       # by default, the MAC address of the␣
↪physical network is used
vnet="vnet0"
```

(continues on next page)

(continued from previous page)

```
vnetbr="vnetbr"
```

RT-Linux Guest IP address settings

If the network is bridged, the IP address of RT-Linux must be adjusted properly. The settings are stored in /hv/lx/linux.config. The IpAddress has to be set to a unique address in your company network, assure the entries are uncommented! The MacAddress has to be adjusted to a unique value only if more than one RT-Linux guest is bridged, in that case, please adjust the last value from `12` to `13`, `14` etc.

```
; This must be set correctly if the vnet device is bridged in the␣
↪Hypervisor host
[Rtos\Vnet\0]
    "IpAddress"="172.17.10.239"
    "MacAddress"="AA:BB:CC:DD:E0:12"
```

2.8 Windows or Linux Guest basic settings

Attention: See chapter *Installation and Configuration* for tutorial **HV-WindowsGuest-Guide**.

The following subitems describes some technical background and edge cases.

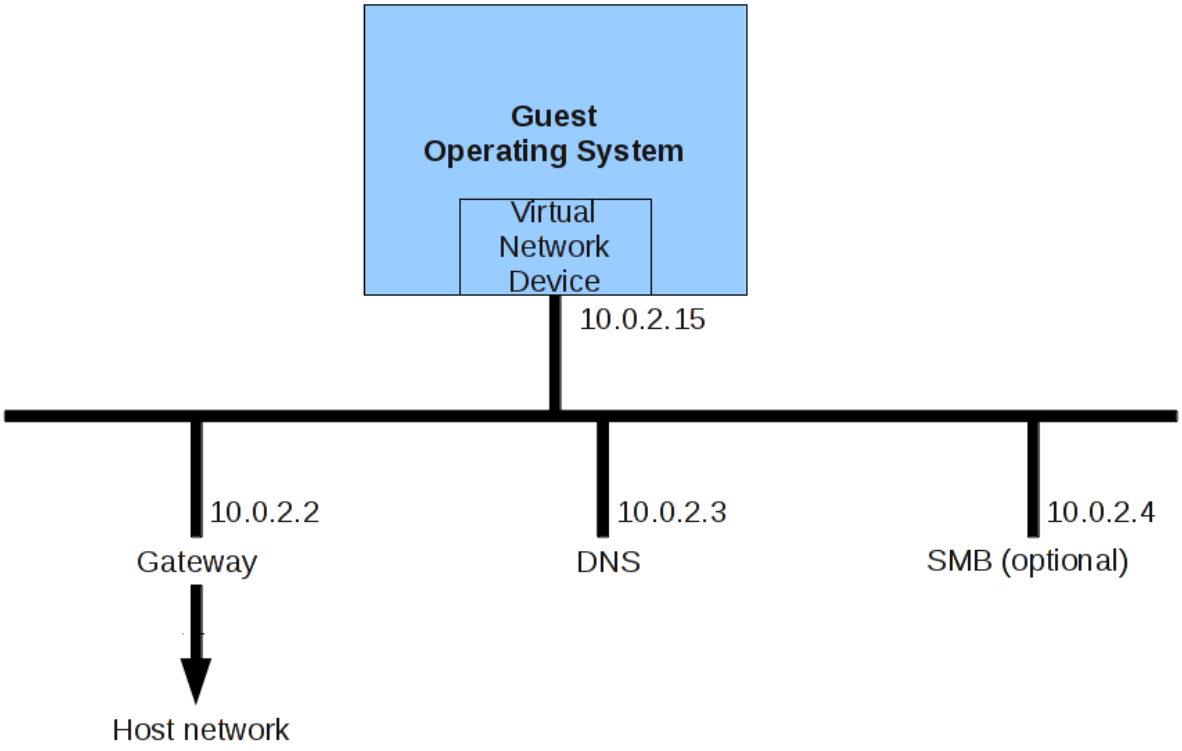
2.8.1 Guest Ethernet

In the `vmconfig.sh` script, two guest Ethernet controllers can be configured.

One is used for directly connecting to the external physical network via network bridging.

To enable this network, set `external_nw` to a value of `1`.

The second is used as a private network, which is connected to the external physical network via NAT. This connection is safer, but much slower. It is also used for file sharing with the hypervisor host. The architecture of the private network looks as follows:



Guest Ethernet MAC address

In the folder `/hv/VMs/vm1` the script `vm1_setmac.sh` defines the Ethernet MAC address for the virtual network adapters inside the guest. By default, if this file does not exist, this script file is automatically generated using random local administered addresses. You will have to change these addresses by an official address related to your company.

The content of this file looks as follows:

```
export ethmacVM1=0A:C0:FD:20:39:01
export ethmacVM2=0A:C1:FD:20:39:01
```

The `ethmacVM1` address belongs to the bridged Ethernet controller (bridged to the external network).

The `ethmacVM2` address belongs to the private (internal) Ethernet controller (using NAT).

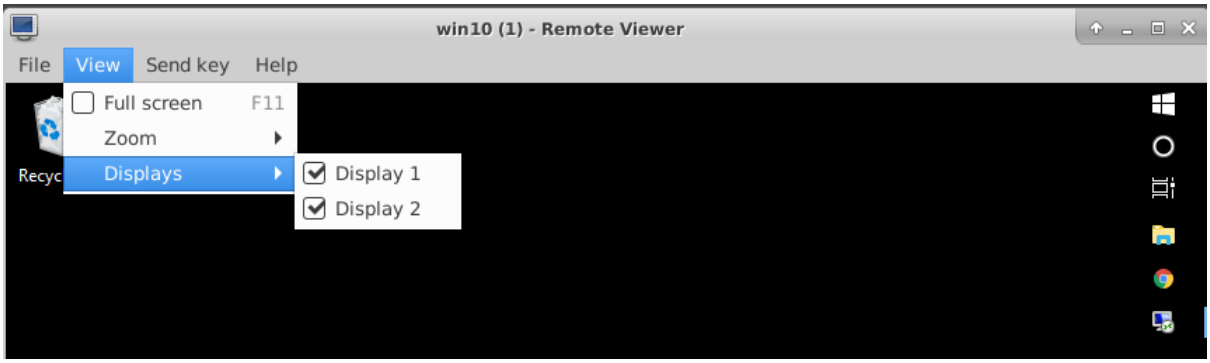
2.8.2 Guest Multiple monitors

If more than one monitor is connected to the system, these can also be used for VM guests (up to a maximum of `4` monitors).

For Windows guests, the following settings are required in `vmconfig.sh`:

```
export windows_guest=1
export num_monitors=# (where # is the number of monitors)
```

To enable additional monitors being displayed, select the displays via the View – Displays menu in the viewer application.

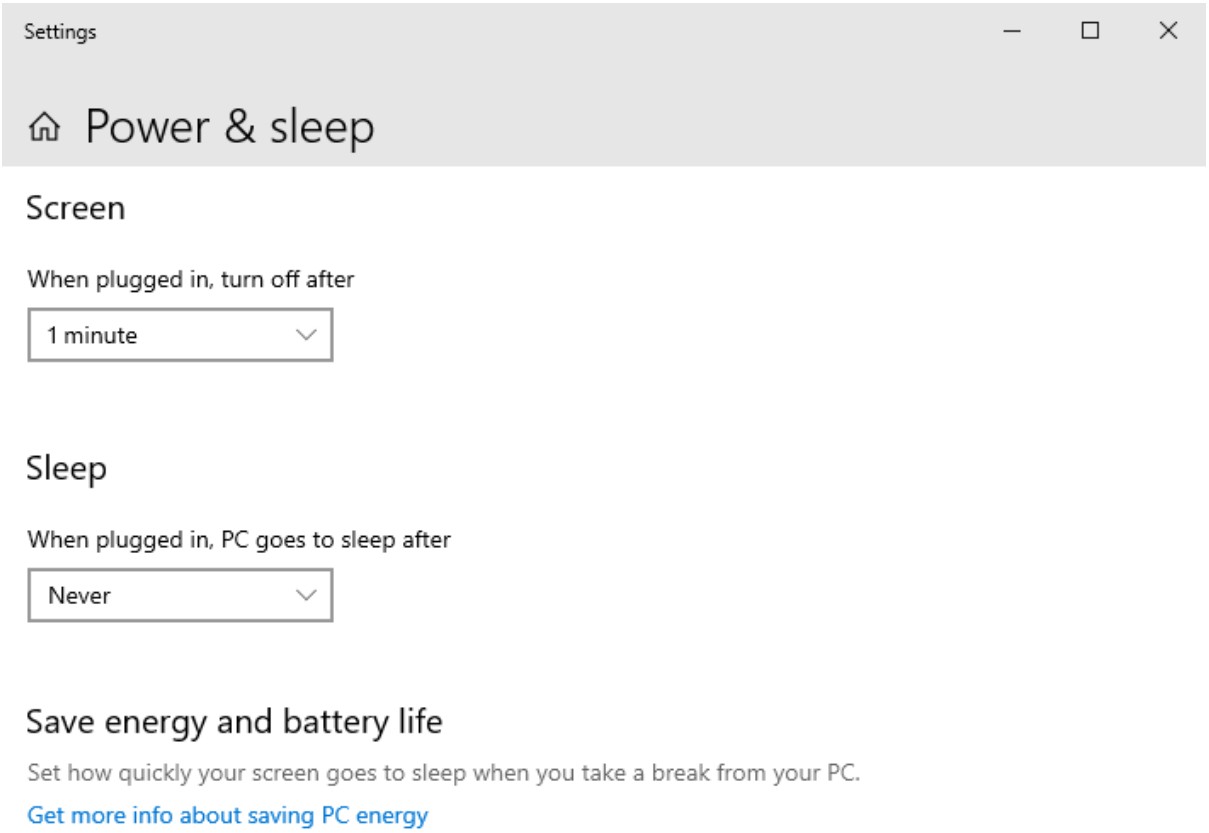


2.9 Windows installation

Attention: See chapter *Installation and Configuration* for tutorial **HV-WindowsGuest-Guide**.

2.9.1 Additional action on the Windows guest:

Disable Sleep (*Windows Settings -> System -> Power and sleep*):



2.10 PCI Device passthrough (Windows/Linux)

2.10.1 Why pass-through

Typically, Windows or Linux guest operating systems will run in a sandbox like virtual machine with no direct hardware access. There are scenarious when this is not sufficient, for example some PCI devices (e.g. CAN cards) will not be virtualized and thus would not be visible in such guest OS. There may also be significant performance impacts in some cases if virtual hardware is used (especially for for graphics hardware). To overcome these limitations, the guest will have to use the real physical hardware instead of virtual hardware. PCI Device passthrough will directly assign a specific PCI device to a Windows or Linux guest.

It is madatory to have hardware support for IOMMU (VT-d). VT-d should be supported by your Processor, your motherboard and should be enabled in the BIOS.

Virtual Function I/O (VFIO) allows a virtual machine to access a PCI device, such as a GPU or network card, directly and achieve close to bare metal performance.

The setup used for this guide is:

- Intel Core I5-8400 or I3-7100 (with integrated Intel UHD 630 Graphics) - this integrated graphics adapter will be assigned to the Windows VM.
- AMD/ATI RV610 (Radeon HD 2400 PRO) – optional, as a second GPU, only needed to have display output for Linux host. Later, the host is reached only via SSH.
- Intel I210 Gigabit Network Card - optional, to demonstrate how to pass through a simple PCI device to a Windows VM

2.10.2 Ethernet PCI Card/Custom PCI device assignment

Some manual work is required to pass through the PCI Device to a Windows VM.

Understanding IOMMU Groups

In order to activate the hardware passthrough we have to prevent the ownership of a PCI device by its native driver and assign it to the vfio-pci driver instead.

In a first step, an overview of the hardware and related drivers is required. In the below example we want to passthrough the I210 Ethernet Controller to the guest VM.

```
rte@rte-System-Product-Name:~$ lspci
00:00.0 Host bridge: Intel Corporation 8th Gen Core Processor Host Bridge/
↪DRAM Registers
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v5/E3-1500 v5/6th Gen␣
↪Core Processor
00:02.0 VGA compatible controller: Intel Corporation Device 3e92
00:14.0 USB controller: Intel Corporation Cannon Lake PCH USB 3.1 xHCI␣
↪Host Controller
00:14.2 RAM memory: Intel Corporation Cannon Lake PCH Shared SRAM (rev 10)
00:16.0 Communication controller: Intel Corporation Cannon Lake PCH HECI␣
↪Controller
00:17.0 SATA controller: Intel Co`rporation Cannon Lake PCH SATA AHCI␣
↪Controller (rev 10)
```

(continues on next page)

(continued from previous page)

```
00:1c.0 PCI bridge: Intel Corporation Device a33c (rev f0)
00:1c.5 PCI bridge: Intel Corporation Device a33d (rev f0)
00:1c.7 PCI bridge: Intel Corporation Device a33f (rev f0)
00:1f.0 ISA bridge: Intel Corporation Device a303 (rev 10)
00:1f.4 SMBus: Intel Corporation Cannon Lake PCH SMBus Controller (rev 10)
00:1f.5 Serial bus controller [0c80]: Intel Corporation Cannon Lake PCH␣
↪SPI Controller
01:00.0 VGA compatible controller: [AMD/ATI] RV610 [Radeon HD 2400 PRO]
01:00.1 Audio device: [AMD/ATI] RV610 HDMI Audio [Radeon HD 2400 PRO]
03:00.0 Ethernet controller: Intel Corporation I210 Gigabit Network␣
↪Connection (rev 03)
04:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/
↪8411 PCI Express Gigabit Ethernet Controller (rev 15)
```

The I210 device is determined as `03:00.0`

Caution: The IOMMU has to be activated, you can verify this as follows:

```
sudo dmesg | grep -e "Directed I/O"
DMAR: Intel(R) Virtualization Technology for Directed I/O
```

Hint: In case the IOMMU is activated, all devices are divided into groups. The IOMMU Group is an indivisible unit. All devices in the same group must be passed through together, it is not possible to only pass through a subset of the devices.

In the next step, we need to get an overview of the IOMMU architecture and determine to which group the device we want to pass through belongs to.

```
for a in /sys/kernel/iommu_groups*; do find $a -type l; done | sort --
↪version-sort
```

```
/sys/kernel/iommu_groups/0/devices/0000:00:00.0
/sys/kernel/iommu_groups/1/devices/0000:00:01.0
/sys/kernel/iommu_groups/1/devices/0000:01:00.0
/sys/kernel/iommu_groups/1/devices/0000:01:00.1
/sys/kernel/iommu_groups/2/devices/0000:00:02.0
/sys/kernel/iommu_groups/3/devices/0000:00:14.0
/sys/kernel/iommu_groups/3/devices/0000:00:14.2
/sys/kernel/iommu_groups/4/devices/0000:00:16.0
/sys/kernel/iommu_groups/5/devices/0000:00:17.0
/sys/kernel/iommu_groups/6/devices/0000:00:1c.0
/sys/kernel/iommu_groups/7/devices/0000:00:1c.5
/sys/kernel/iommu_groups/8/devices/0000:00:1c.7
/sys/kernel/iommu_groups/9/devices/0000:00:1f.0
/sys/kernel/iommu_groups/9/devices/0000:00:1f.4
/sys/kernel/iommu_groups/9/devices/0000:00:1f.5
/sys/kernel/iommu_groups/10/devices/0000:03:00.0
/sys/kernel/iommu_groups/11/devices/0000:04:00.0
```

The I210 device (`03:00.0`) belongs to the IOMMU group `10`. It is important to know that all devices in a single group are shared. No other device belongs to this IOMMU group so we can pass through the I210

device to the guest.

We need to determine the PCI vendor and device ID of the I210 device now:

```
rte@rte-System-Product-Name:~$ lspci -s 03:00.0 -vvn
03:00.0 0200: 8086:1533 (rev 03)
    ...
    Kernel driver in use: igb
    Kernel modules: igb
```

Add to the linux kernel command line parameters the following: `vfio-pci.ids=8086:1533`. To edit kernel parameters edit `/etc/grub.d/40_custom` file and then execute update-grub and reboot.

```
sudo gedit /etc/grub.d/40_custon

menuentry 'Hypervisor' --class ubuntu --class gnu-linux --class gnu --
↪class os $menuentry_id_option 'gnulinux-simple-8b4e852a-54b3-4568-a5a2-
↪dec7b16b8e07' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
        insmod part_gpt
        insmod ext2
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --
↪hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2  8b4e852a-54b3-4568-a5a2-
↪dec7b16b8e07
        else
          search --no-floppy --fs-uuid --set=root 8b4e852a-54b3-4568-a5a2-
↪dec7b16b8e07
        fi
        linux   /boot/vmlinuz-5.15.0-41-generic root=UUID=8b4e852a-54b3-
↪4568-a5a2-dec7b16b8e07 ro  quiet splash $vt_handoff find_preseed=/
↪preseed.cfg auto noprompt priority=critical locale=en_US memmap=8k\
↪$128k memmap=8M\$56M memmap=256M\$64M memmap=16M\$324M maxcpus=3␣
↪intel_pstate=disable acpi=force idle=poll nohalt pcie_port_pm=off pcie_
↪pme=nomsi cpuidle.off=1 intel_idle.max_cstate=0 noexec=off noexec32=off␣
↪nox2apic intel_iommu=on iommu=pt intremap=off vfio_iommu_type1.allow_
↪unsafe_interrupts=1 vfio-pci.ids=10ec:8168
        initrd  /boot/initrd.img-5.15.0-41-generic
}
```

```
sudo update-grub
sudo reboot
```

Hint: Normally no other steps are needed to to replace the native driver by the vfio-pci driver. If you have conflicts between these two drivers, it may be required to disable the loading of the native driver. Add the parameter `module_blacklist=igb` to a kernel command line in that case.

Hint: As mentioned before, device passthrough requires IOMMU support by the hardware and the OS. It is needed to add `intel_iommu=on iommu=pt` to your kernel command line. These parameters are

automatically added by Hypervisor when executing the `inithv.sh` script.

Typically, one or multiple PCI devices will also be assigned to a RTOS, for example Real-time Linux. In such case (one or more PCI devices are passed through to a Windows or Ubuntu guest as well as one or multiple PCI devices are assigned to a RTOS) it is required to deactivate IOMMU Interrupt Remapping in the Linux Kernel.

The following kernel command line parameters usually are added as well by the `inithv.sh` script into the GRUB Entry "Hypervisor" in the `/etc/grub.d/40_custom` file. You may verify and add these parameters if they are missing. In that case it is also required to execute update-grub and then reboot.

```
intremap=off vfio_iommu_type1.allow_unsafe_interrupts=1
```

VM configuration

Last step is to edit the `/hv/VMs/VM1/vmconfig.sh` file and add the PCI Etherner Card information here.

Uncomment `#export OTHER_HW` variable and set it to:

```
export OTHER_HW=" -device vfio-pci,host=03:00.0"
```

2.10.3 Intel Integrated Graphics (iGVT-d) assignment

To use graphics passthrough, less steps compared to standard PCI hardware passthrough are required, because Hypervisor automates most of the steps.

Understanding GPU modes UPT and Legacy

There are two modes "legacy" and "Universal Passthrough" (UPT).

Hypervisor uses only Legacy mode, but it could be important to understand the difference.

UPT is available for Broadwell and newer processors. Legacy mode is available since SandyBridge. If you are unsure, which processor you have, please check this link https://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures

In Legacy it is meant that IGD is a primary and exclusive graphics in VM. Additionally the IGD address in the VM must be PCI 00:02.0, only 440FX chipset model (in VM) is supported and not Q35. The IGD must be the primary GPU for host as well (please check your BIOS settings).

In UPT mode the IGD can have another PCI adress in VM and the VM can have a second graphics adapter (for example qxl, or vga).

Please read here more about legacy and UPT mode: https://git.qemu.org/?p=qemu.git;a=blob;f=docs/igd-assign.txt

There a lot of other little things, why IGD Passthrough could not work. For ex. In legacy mode it expects a ISA/LPC Bridge at PCI Adress `00:1f.0` in VM and this is a reason, why Q35 chip does not work, because it has another device at this adress.

In UPT mode, there is no output support of any kind. So the UHD graphics can be used for accelerating (for ex. Decoding) but the Monitor remains black and there is a non-standard experimental qemu vfi-pci command line parameter x-igd-opregion=on, which can work.

Blacklisting i915 driver on Host

The standard `Intel Driver i915` is complex and it is not always possible to safely unbind the device from this driver, that is why this driver is blacklisted by Hypervisor when executing `inithv.sh` script.

Deactivating Vesa/EFI Framebuffer on Host

Please also know, when i915 driver is disabled, there are other drivers which are ready to jump on the device to keep the console working. Depend on your BIOS settings (legacy or UEFI) two other drivers can occupy a region of a video memory: `efifb` or `vesafb`.

Hypervisor blacklists both by adding the following command line parameter:

```
video=vesafb:off,efifb:off
```

Please also check if it works: `cat/proc/iomem`. If you still see that one of this driver still occupies a part of a video memory, please try manually another combination:

```
video=efifb:off,vesafb:off.
```

Legacy BIOS, pure UEFI and CSM+UEFI in Host

It plays also significant role, in which mode your machine is booted: Legacy BIOS, pure UEFI or UEFI with CSM support. In pure UEFI (on host) QEMU cannot read video ROM. In this case you could extract it manually (for ex. Using *Cpu-Z* utility or just boot in CSM mode, when iGPU is a primary GPU in BIOS), patch it with correct device id and provide it to qemu as *romfile*= parameter for `vfi-pci`. Please google for *rom-parser* and *rom-fixer* for details.

SeaBIOS and OVMF (UEFI) in VM

It also plays role which BIOS you use in the VM itself. For QEMU there are two possibilities: SeaBIOS (legacy BIOS, which is default for qemu) and OVMF (UEFI). You can download and build OVMF itself, but easier to install precompiled binaries from here: https://www.kraxel.org/repos/.

For your convenience, Hypervisor installs precompiled OVMF binaries to /hv/bin directory: `OVMF_CODE.fd` `OVMF_VARS.fd`

By default, OVMF UEFI does not support OpRegion Intel feature, which is required to have a graphics output to a real display. There are three possibilities how to solve this problem and the easiest one seems to be the using special vbios rom `vbios_gvt_uefi.rom`, please read more here https://wiki.archlinux.org/index.php/Intel_GVT-g.

For your convenience, we have already included this file into the Hypervisor package and it can also be located in `/hv/bin` directory.

Hypervisor uses OVMF (UEFI) for graphics pass-through.

How to do it in Hypervisor

The final working configuration which we consider here:

- CPU Graphics is a primary GPU in BIOS (host)
- Host boots in pure UEFI mode
- OVMF is used as BIOS in Windows VM
- `vbios_gvt_uefi.rom` is used as VBIOS in `romfile` parameter for `vfio-pci`
- Legacy mode for IGD, so the Windows VM has only one graphics card Intel UHD 630

Which commands should be executed in Hypervisor to do a pass-through of a Intel Integrated Graphics to a Windows VM?

None! Almost everything is done automatically. When executing `inithv.sh` script (which is required to install real-time linux kernel and to reserve kernel memory for hypervisor needs), a separate GRUB entry "Hypervisor + iGVT-d" is created.

This entry contains already all necessary linux kernel parameters required to do a graphics pass-through: blacklisting intel driver, disabling interrupt remapping, assigning a VGA device to a `vfio-pci` driver and other stepts.

But one step should be done once, configuring your VM.

Change `/hv/VMs/VM1/vmconfig.sh` script. Two variables should be uncommented and activated:

```
export uefi_bios=1
export enable_vga_gpt=1
```

Just reboot your machine, choose "Hypervisor+iGVT-d" menu item. If everything is correct, the display of your Host should remain black. Connect to the machine using SSH connection or use a second graphics card for Host (read next chapter) and then execute `/hv/VMs/vm1/vmrun.sh`

Wait 30-60 seconds and.. display remains black? Of course. Windows does not have Intel Graphics drivers.

Remember we configured Windows for a Remote Desktop Access in previous steps? Connect to Windows VM via RDP and install latest Intel Drivers https://downloadcenter.intel.com/product/80939/Graphics

If everyting is done correctly, your display should now work and display a Windows 10 Desktop.

Using Second GPU Card for Host

X-Windows on the hypervisor host does not work properly, when the primary GPU in the System is occupied by the `vfi-pci` driver. It detects the first GPU, tries to acquire it, fails and then aborts. We should let it know, that it should use our second GPU card instead.

Log in to the hypervisor host (Press `Ctrl` + `Alt` + `F3`, for example), shutdown LightDM manager sudo service lightdm stop.

Execute sudo X –configure, it creates `xorg.conf.new` file in the current directory. When this command is executed, X server enumerates all hardware and creates this file . By default, in modern systems, Xserver does not need the `xorg.conf` file, because all hardware is detected quite good and automatically. But the config file is still supported.

Look at this file, find a section "Device" with your second GPU (look at the PCI Adress). Copy content of this section to a separate file `/etc/X11/xorg.conf.d/secondary-gpu.conf`. Save and reboot.

```
Section "Device"
    ### Available Driver options are:-
    ### Values: <i>: integer, <f>: float, <bool>: "True"/"False",
    ### <string>: "String", <freq>: "<f> Hz/kHz/MHz",
    ### <percent>: "<f>%"
    ### [arg]: arg optional
    #Option "Accel" # [<bool>]
    #Option "SWcursor" # [<bool>]
    #Option "EnablePageFlip" # [<bool>]
    #Option "SubPixelOrder" # [<str>]
    #Option "ZaphodHeads" # <str>
    #Option "AccelMethod" # <str>
    #Option "DRI3" # [<bool>]
    #Option "DRI" # <i>
    #Option "ShadowPrimary" # [<bool>]
    #Option "TearFree" # [<bool>]
    #Option "DeleteUnusedDP12Displays" # [<bool>]
    #Option "VariableRefresh" # [<bool>]
    Identifier "Card0"
    Driver "amdgpu"
    BusID "PCI:1:0:0"
EndSection
```

Assigning external PCI Video Card to Windows VM

External PCI Video Cards are not automatically recognized by `inithv.sh` script (unlike the CPU integrated video), so this entry is not added to the grub boot menu.

So if you want to pass the external GPU to a VM through, you first need to create a separate GRUB entry or modify an existing one.

Let'a assume we've already executed `inithv.sh` script (as described in previous chapters) and it created a "Hypervisor" boot entry.

Boot computer using this "Hypervisor" entry.

Open `/boot/grub.cfg`, find its corresponding

```
menuentry 'Hypervisor'
```

section

Edit it and rename the menu entry name to:

```
menuentry 'Hypervisor + Nvidia Quadro Passthrough'
```

The kernel command line in this section should like like:

```
linux   /boot/vmlinuz-5.4.17-rt9-acontis+ root=UUID=ebbe5511-f724-4a1d-
↪b5a5-e8dafecaf451 ro  quiet splash $vt_handoff find_preseed=/preseed.cfg␣
↪auto noprompt priority=critical locale=en_US
        memmap=8k\$128k memmap=8M\$56M memmap=256M\$64M memmap=16M\$384M␣
↪maxcpus=7 intel_pstate=disable acpi=force idle=poll nohalt pcie_port_
↪pm=off pcie_pme=nomsi cpuidle.off=1
        intel_idle.max_cstate=0 noexec=off nox2apic intremap=off vfio_
↪iommu_type1.allow_unsafe_interrupts=1 intel_iommu=on iommu=pt
```

1) First thing, we should discover the topology of PCI devices

find it out, which PCI slot is used for our external PCI card. type `lspci`

```
00:00.0 Host bridge: Intel Corporation Device 9b53 (rev 03)
00:02.0 VGA compatible controller: Intel Corporation Device 9bc8 (rev 03)
00:08.0 System peripheral: Intel Corporation Xeon E3-1200 v5/v6 / E3-1500␣
↪v5 / 6th/7th Gen Core Proc
00:12.0 Signal processing controller: Intel Corporation Device 06f9
00:14.0 USB controller: Intel Corporation Device 06ed
00:14.1 USB controller: Intel Corporation Device 06ee
00:14.2 RAM memory: Intel Corporation Device 06ef
00:16.0 Communication controller: Intel Corporation Device 06e0
00:17.0 SATA controller: Intel Corporation Device 06d2
00:1b.0 PCI bridge: Intel Corporation Device 06c0 (rev f0)
00:1b.4 PCI bridge: Intel Corporation Device 06ac (rev f0)
00:1c.0 PCI bridge: Intel Corporation Device 06b8 (rev f0)
00:1c.4 PCI bridge: Intel Corporation Device 06bc (rev f0)
00:1c.5 PCI bridge: Intel Corporation Device 06bd (rev f0)
00:1f.0 ISA bridge: Intel Corporation Device 0685
00:1f.4 SMBus: Intel Corporation Device 06a3
00:1f.5 Serial bus controller [0c80]: Intel Corporation Device 06a4
02:00.0 VGA compatible controller: NVIDIA Corporation GM206GL [Quadro␣
↪M2000] (rev a1)
02:00.1 Audio device: NVIDIA Corporation Device 0fba (rev a1)
04:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. Device 8125␣
↪(rev 05)
05:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network␣
↪Connection
```

Ok. `02:00.0` is our device. Let's discover, which driver occupies this device.

```
type "lspci -s 02:00.0 -vv"

02:00.0 VGA compatible controller: NVIDIA Corporation GM206GL [Quadro␣
↪M2000] (rev a1) (prog-if 00 [VGA controller])
      Subsystem: NVIDIA Corporation GM206GL [Quadro M2000]
      ...
      Kernel modules: nvidiafb, nouveau
```

So we know now, which drivers should be blacklisted in kernel: `nvidiafb` and `noueveau`.

But, it looks like we have an integrated audio device in `02:00.1`. Most probably we should deactivate its driver as well.

Let's investigate our IOMMU groups, type `find /sys/kernel/iommu_groups/ -type l`:

```
/sys/kernel/iommu_groups/7/devices/0000:00:1b.0
/sys/kernel/iommu_groups/15/devices/0000:05:00.0
/sys/kernel/iommu_groups/5/devices/0000:00:16.0
/sys/kernel/iommu_groups/13/devices/0000:02:00.0
/sys/kernel/iommu_groups/13/devices/0000:02:00.1
/sys/kernel/iommu_groups/3/devices/0000:00:12.0
/sys/kernel/iommu_groups/11/devices/0000:00:1c.5
/sys/kernel/iommu_groups/1/devices/0000:00:02.0
/sys/kernel/iommu_groups/8/devices/0000:00:1b.4
/sys/kernel/iommu_groups/6/devices/0000:00:17.0
/sys/kernel/iommu_groups/14/devices/0000:04:00.0
/sys/kernel/iommu_groups/4/devices/0000:00:14.1
```

(continues on next page)

(continued from previous page)

```
/sys/kernel/iommu_groups/4/devices/0000:00:14.2
/sys/kernel/iommu_groups/4/devices/0000:00:14.0
/sys/kernel/iommu_groups/12/devices/0000:00:1f.0
/sys/kernel/iommu_groups/12/devices/0000:00:1f.5
/sys/kernel/iommu_groups/12/devices/0000:00:1f.4
/sys/kernel/iommu_groups/2/devices/0000:00:08.0
/sys/kernel/iommu_groups/10/devices/0000:00:1c.4
/sys/kernel/iommu_groups/0/devices/0000:00:00.0
/sys/kernel/iommu_groups/9/devices/0000:00:1c.0
```

so, we see, that our NVidia card belongs to a IOMMU group 13, together with its audio device.

So, repeat steps for audio device, type `lspci -s 02:00.1 -vv`:

```
02:00.1 Audio device: NVIDIA Corporation Device 0fba (rev a1)
...
Kernel modules: snd_hda_intel
```

Now all these `3` drivers should be blacklisted in your system.

add the following to your "Hypervisor" entry kernel command line: `module_blacklist=nouveau,nvidiafb,snd_hda_intel`

2) Assign devices to KVM.

In order to make it possible to pass NVidia VGA and Audio device to a Windows VM through we should assign a special driver `vfi-pci` to each our drvice.

Let's determine Vendor and Device IDs:

```
type "lspci -s 02:00.0 -n"
02:00.0 0300: 10de:1430 (rev a1)

type "lspci -s 02:00.1 -n"
02:00.1 0403: 10de:0fba (rev a1)
```

add the following to our kernel command line: `vfio-pci.ids=10de:1430,10de:0fba`

so our final command line should now look like:

```
linux   /boot/vmlinuz-5.4.17-rt9-acontis+ root=UUID=ebbe5511-f724-4a1d-
↪b5a5-e8dafecaf451 ro  quiet splash $vt_handoff find_preseed=/preseed.cfg␣
↪auto noprompt priority=critical locale=en_US
       memmap=8k\$128k memmap=8M\$56M memmap=256M\$64M memmap=16M\$384M␣
↪maxcpus=7 intel_pstate=disable acpi=force idle=poll nohalt pcie_port_
↪pm=off pcie_pme=nomsi cpuidle.off=1
       intel_idle.max_cstate=0 noexec=off nox2apic intremap=off vfio_
↪iommu_type1.allow_unsafe_interrupts=1 intel_iommu=on iommu=pt module_
↪blacklist=nouveau,nvidiafb,snd_hda_intel
       vfio-pci.ids=10de:1430,10de:0fba
```

if you boot the computer into the this new grub entry, you could check if vfio-pci driver successfully acquired our devices:

```
type "dmesg | grep vfio"

[    5.587794] vfio-pci 0000:02:00.0: vgaarb: changed VGA decodes:␣
↪olddecodes=io+mem,decodes=io+mem:owns=none
```

(continues on next page)

(continued from previous page)

```
[    5.606445] vfio_pci: add [10de:1430[ffffffff:ffffffff]] class 0x000000/
↪00000000
[    5.626451] vfio_pci: add [10de:0fba[ffffffff:ffffffff]] class 0x000000/
↪00000000
```

everything is ok.

3) Check if your PCI Card is not a primary graphics device in your system. Boot into BIOS and find the corresponding settings. For every BIOS there are own namings for this option.

Often there is no such option in BIOS and it then detects which HDMI ports are connected and if both Integrated GPU and the external PCI card are connected to monitors, then the integrated card is choosen by BIOS as a primary device. If not, then your PCI card is selected as the primary device.

If your card is not a primary device and integrated CPU graphics is used for host, skip the reading this section. But if not, you should disable frame buffer, because its drivers occupy video card PCI regions.

Add then `` video=efifb:off,vesafb:off disable_vga=1`` to your kernel command line.

This trick is also useful, when your PC has no integrated CPU graphics and the external PCI Video card is the only device in your system.

4) assign your VGA and Audio devices to your Windows VM (Qemu)

usually the VM configuation is located in this file: `/hv/VMs/vm1/vmconfig.sh`

find and change `OTHER_HW` line from

```
export OTHER_HW=$OTHER_HW
```

to

```
export OTHER_HW=$OTHER_HW" -device vfio-pci,host=02:00.0 -device vfio-pci,
↪host=02:00.1 -nographic"
```

Note: Please note, you need to install your video card manufacturer drivers to a Windows VM, to make the graphics working.

Create a temporary VGA device or a Windows RDP/Qemu VNC connection to your VM to install graphics drivers.

2.10.4 Keyboard and Mouse assignment

Normally your Linux Host with RTOSVisor with Windows VM and Integrated Graphics passed through works in head-less mode. Windows VM outputs to a monitor durch DVI-D/HDMI connection and the Linux Host is controlled via SSH connection. Windows has a look and feel as it works without an intermediate hypervisor layer.

So, Windows needs a keyboard and mouse.

Go to `/dev/input/by-id/` and find something that looks like a keyboard and mouse and it should containt **"-event-"** in its name.

```
ls -la

usb-18f8_USB_OPTICAL_MOUSE-event-if01 -> ../event5
usb-18f8_USB_OPTICAL_MOUSE-event-mouse -> ../event3
usb-18f8_USB_OPTICAL_MOUSE-if01-event-kbd -> ../event4
usb-18f8_USB_OPTICAL_MOUSE-mouse -> ../mouse0
usb-SEM_USB_Keyboard-event-if01 -> ../event7
usb-SEM_USB_Keyboard-event-kbd -> ../event6
```

configure your VM with these parameters by editing `vmconfig.sh`:

```
export vga_gpt_kbd_event=6
export vga_gpt_mouse_event=3
```

Please also note, disconnecting evdev devices, such as keyboard or mouse, can be problematic when using qemu and libvirt, because it does not reopen device when the device reconnects.

If you need to disconnect/reconnect your keyboard or mouse, there is a workaround, create a udev proxy device and use its event device instead. Please read more hier https://github.com/aiberia/persistent-evdev.

If everything works, you'll find new devices like `uinput-persist-keyboard0` pointing to `/dev/input/eventXXX` use use these ids as usual in:

```
export vga_gpt_kbd_event=XXX
export vga_gpt_mouse_event=ZZZ
```

2.11 Intel(c) VT and acontis Hypervisor

Intel(c) VT-x and VT-d are hardware virtualisation extensions for Intel Processors. Please take a look at official Intel page for full description of the technology: https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

In acontis Hypervisor, VT techonogy allows to activate so called "Shared Mode". In Shared Mode is is possible to run a RTOS on the same CPU as Host (so, CPU is shared between two OS). Please note, that hard real time for RTOS is also achieved in this mode.

2.11.1 Activating VT

VT should be activated manually. In this example we configure `2` CPU system, where first CPU is used to run Linux Host and RTOS1 (so, in shared CPU mode) and second CPU will be used to run RTOS2 (as exclusive core).

Note: Before you start to change anything in your system, please make sure, you have executed `inithv.sh` script at least once.

This script installs necessary `.deb` packages, configures memory settings, reads ACPI tables and etc.

Linux Kernel grub parameters

IOMMU must be *disabled* in order to activate VT-D.

Add the following parameters to the linux kernel in `grub.cfg`: `intremap=off intel_iommu=off maxcpus=1`

Make sure these parameters have been removed: `iommu=pt vfio_iommu_type1.allow_unsafe_interrupts=1`

Currently **only** one shared cpu is supported. This is the reason, why `maxcpus=1` is used.

Adapt `.config` files

1. **make sure this line presents in your `config`:**
   `#include "../config/memmap820.config"`

   it is a memory map of your system, created by `inithv.sh` script.
2. **`[Upload]`**
   `"VersionDrv"=dword:9070000`
3. **`[Vmf]`**
   `"VtAllowed"=dword:1`
4. **`[Rtos]`**
   `"MemoryType"=dword:3`
5. **`[Rtos1]`**
   `"MemoryType"=dword:3`
6. **`[Rtos\Vmf]`**
   `"MapSystemTables"=dword:1`
7. **`hwdevbase.config` and `hvdevbase_rtos2.config`**
   Please make sure, that *Destination* parameter for first and second rtos has **correct** processor mask. Each interrupt should be forwarded to the correct CPU.
8. **`rtos_eth.config`**
   If you use a network card, please make sure, that all interrupts from each card are forwarded to the correct CPU. Please make sure, that *Destination* parameter for first and second rtos has **correct** processor mask.

2.12 Windows or Linux Guest Operation

2.12.1 General Guest control

Boot

Change into the guest directory (`/hv/VMs/vm1`) and execute the `vmrun.sh` script. When executed the very first time, the virtual hard disk image is created (e.g. file `vm1.qcow2`).

- **Shutdown**
  Change into the guest directory (`/hv/VMs/vm1`) and execute the `vmshtdn.sh` script.

- **Reset**
  Change into the guest directory (`/hv/VMs/vm1`) and execute the `vmrst.sh` script.
- **Destroy the guest (in case it crashed)**
  Change into the guest directory (`/hv/VMs/vm1`) and execute the `vmkill.sh` script.
- **View guest output (in case the output Windows crashed or was closed)**
  Change into the guest directory (`/hv/VMs/vm1`) and execute the `vmview.sh` script.
- **Other control and monitoring commands**
  Change into the guest directory (`/hv/VMs/vm1`) and execute the `vmmon.sh` script. Monitor commands are described in here: https://en.wikibooks.org/wiki/QEMU/Monitor

2.13 Windows/Linux USB guest access (non automatic mode)

In case the USB device is plugged in while the guest is already running, you need to start the guest monitor using the `vmmon.sh` command in the vm directory.

Using the info command, a list of connected USB devices can be found:

```
info usbhost
    Bus 1, Addr 11, Port 1, Speed 480 Mb/s
        Class 00: USB device 0951:1665, DataTraveler 2.0
    Bus 2, Addr 3, Port 4.2, Speed 5000 Mb/s
        Class ff: USB device 0b95:1790, AX88179
    Bus 1, Addr 8, Port 8, Speed 12 Mb/s
        Class ff: USB device 06cb:009a
    Bus 1, Addr 7, Port 7, Speed 480 Mb/s
        Class ef: USB device 04f2:b604, Integrated Camera
    Bus 1, Addr 9, Port 6, Speed 12 Mb/s
        Class e0: USB device 8087:0a2b
    Bus 1, Addr 4, Port 5, Speed 12 Mb/s
        Class 00: USB device 058f:9540, EMV Smartcard Reader
    Bus 1, Addr 2, Port 2, Speed 12 Mb/s
        Class 00: USB device 046d:c52b, USB Receiver
```

The following command will dynamically connect the USB device on host bus 1 and address 11 (an USB stick) with the guest:

```
device_add usb-host,id=MyUsbDevice,hostbus=1,hostaddr=11
```

The id value has to be unique in case multiple USB devices are connected.

Hint: More information can be found here:

- https://github.com/qemu/qemu/blob/master/docs/usb2.txt
- https://unix.stackexchange.com/questions/426652/connect-to-running-qemu-instance-with-qemu-monitor/476617
- https://wiki.archlinux.de/title/QEMU#USB_Peripherie

2.14 Windows/Linux USB guest access (passthrough mode) for non Real-time guests

If a specific physical USB port shall be permanently used by the guest, this can be accomplished using USB passthrough mode.
In this case, the hypervisor will monitor a specific USB port and automatically passthrough a USB device (for ex. an USB 3.0 Stick) to an active guest.

In the first step, you need to determine the USB hostbus and hostport value pairs for the selected physical USB port.

Caution: Depending on the USB type (USB1/2 or USB3) there will be different values even if the same physical port is used.

Caution:

If a USB mouse is connected to a specific USB port, you must not passthrough this USB port unless you are using graphics passthrough mode.
This means, the physical USB port where such mouse is connected must not be used for other devices.
Also, the USB mouse must not be connected to any physical USB port which is passed through to the guest.
If this port is passed through, the mouse pointer will not be visible!

Hint: More information can be found here:

- https://qemu.weilnetz.de/doc/6.0/system/usb.html
- https://qemu-project.gitlab.io/qemu/system/devices/usb.html
- https://www.kraxel.org/blog/2018/08/qemu-usb-tips/

2.14.1 USB1/2 devices

Connect an USB2 stick to the USB port you want to passthrough and execute the following on the Linux Host:

```
$ lsusb -t
```

You will get a result similar to

```
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 10000M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
    |__ Port 4: Dev 18, If 0, Class=Mass Storage, Driver=usb-storage, 480M
    |__ Port 6: Dev 4, If 0, Class=Wireless, Driver=btusb, 12M
    |__ Port 6: Dev 4, If 1, Class=Wireless, Driver=btusb, 12M
```

The USB hostbus value for this physical port using USB1/USB2 devices is 1 (see line 2)
The USB hostport value for this physical port using USB1/USB2 devices is 4 (see line 3)

2.14.2 USB3 devices

Connect an USB3 stick to the USB port you want to passthrough and execute the following on the Linux Host:

```
$ lsusb -t
```

You will get a result similar to

```
1 /:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 10000M
2     |__ Port 5: Dev 18, If 0, Class=Mass Storage, Driver=usb-storage, 5000M
3 /:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
4     |__ Port 6: Dev 4, If 0, Class=Wireless, Driver=btusb, 12M
5     |__ Port 6: Dev 4, If 1, Class=Wireless, Driver=btusb, 12M
```

The USB hostbus value for this physical port using USB3 devices is 2 (see line 1)
The USB hostport value for this physical port using USB3 devices is 5 (see line 2)

2.14.3 USB hubs

If devices are connected behind an USB hub, you will see multiple nested ports behind which the device can be found.

Example 1: USB3 device behind a hub

```
1 /:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 10000M
2     |__ Port 1: Dev 15, If 0, Class=Hub, Driver=hub/4p, 5000M
3         |__ Port 2: Dev 16, If 0, Class=Vendor Specific Class,␣
  ↪Driver=ax88179_178a, 5000M
4 /:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
5     |__ Port 6: Dev 4, If 0, Class=Wireless, Driver=btusb, 12M
6     |__ Port 6: Dev 4, If 1, Class=Wireless, Driver=btusb, 12M
```

The USB hostbus value for this physical port behind a hub is 2 (see line 1)
The USB hostport value for this physical port behind a hub is 1.2 (see lines 2 and 3)

Example 2: USB2 device behind two hubs

```
1 /:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 10000M
2 /:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
3     |__ Port 1: Dev 26, If 0, Class=Hub, Driver=hub/4p, 480M
4         |__ Port 1: Dev 29, If 0, Class=Hub, Driver=hub/4p, 480M
5             |__ Port 2: Dev 30, If 0, Class=Mass Storage, Driver=usb-
  ↪storage, 480M
6     |__ Port 6: Dev 4, If 0, Class=Wireless, Driver=btusb, 12M
7     |__ Port 6: Dev 4, If 1, Class=Wireless, Driver=btusb, 12M
```

The USB hostbus value for this physical port behind a hub is 1 (see line 2)
The USB hostport value for this physical port behind a hub is 1.1.2 (see lines 3,4 and 5)

Caution: Some USB devices contain an internal USB hub to expose multiple USB device instances. Here, the same rules apply.

2.14.4 Guest configuration

The USB hostbus and hostport value pairs need to be used in the guest configuration file `vmconfig.sh`. Below you will find the required entries for the above examples.

```
# USB host passthrough (automatic passthrough for any device connected to␣
↪these ports).
#   Note: on the same physical USB port, different values for hostbus,
↪hostport pairs will show up for different USB speed!
export USB_HOST_ADAPTER1_PASSTHROUGH=""
export USB_HOST_ADAPTER1_PASSTHROUGH="$USB_HOST_ADAPTER1_PASSTHROUGH -
↪device usb-host,bus=$USB_HOST_ADAPTER1_NAME.0,hostbus=1,hostport=4"
export USB_HOST_ADAPTER1_PASSTHROUGH="$USB_HOST_ADAPTER1_PASSTHROUGH -
↪device usb-host,bus=$USB_HOST_ADAPTER1_NAME.0,hostbus=2,hostport=5"
export USB_HOST_ADAPTER1_PASSTHROUGH="$USB_HOST_ADAPTER1_PASSTHROUGH -
↪device usb-host,bus=$USB_HOST_ADAPTER1_NAME.0,hostbus=2,hostport=1.2"
export USB_HOST_ADAPTER1_PASSTHROUGH="$USB_HOST_ADAPTER1_PASSTHROUGH -
↪device usb-host,bus=$USB_HOST_ADAPTER1_NAME.0,hostbus=1,hostport=1.1.2
↪"
```

Launch the guest and try to physically connect and disconnect a USB devices to the configured ports. The guest then should recognize such device automatically.

2.15 Windows/Linux COM port guest access

It is possible to passthrough a `COM` port to a guest VM. It works and for both, legacy serial ports and for USB-to-Serial converters.

modify the `vmconfig.sh` file and change

```
export OTHER_HW=$OTHER_HW

to

export OTHER_HW=$OTHER_HW"  -serial /dev/ttyUSB0 -serial /dev/ttyS0"
```

this example creates `COM1` Port in the VM and it uses USB-to-Serial converter on a Linux Host and creates `COM2` port in the VM and it is the real COM1 port on Linux Host.

2.16 Hypervisor host network configuration

The hypervisor host network can be configured using automatic IP address configuration (DHCP), manual IP address configuration or disabled network. To simplify the process, the netconf command is provided.

Hint: netconf is a link to the script `/hv/config/netconf.sh`.

Caution:

The host network configuration has to match with the guest network settings.
The guest network settings are determined by parameter netif_mode in the guest configuration file (e.g. `/hv/VMs/vm1/vmconfig.sh`).
If the hypervisor host and guest settings do not match, the behaviour is undefined.

2.16.1 Automatic network configuration

This is the default mode. Nothing has to be changed if this mode shall be used. In case, the mode previously had been different, you can switch back to the automatic configuration as follows.

First, set the hypervisor host configuration appropriately.

```
$ cd /hv/VMs/vm1
$ netconf -auto
```

Then, configure the guest.

```
$ cd /hv/VMs/vm1
$ gedit vmconfig.sh
```

Change the configuration value.

```
netif_mode=1
```

2.16.2 Manual network configuration

If you want the hypervisor host to be configured manually, you need to adjust the settings accordingly.

```
$ cd /hv/VMs/vm1
$ netconf %DEVICE% -man IP-address netmask-bits gateway-IP dns-IP
```

For example:

```
$ netconf enp1s0 -man 192.168.178.188 24 192.168.178.1 8.8.8.8
```

Then, configure the guest.

```
$ cd /hv/VMs/vm1
$ gedit vmconfig.sh
```

Change the respective configuration values.

```
netif_mode=0
netif_m=...
defaultgw_m=...
dnsgw_m=...
brip_m=...
brnm_m=...
```

2.16.3 Disabled network

If you want the hypervisor host not to use the network, you need to adjust the settings accordingly. In case the PC currently is connected with the LAN and you want to use this connection for a guest, you need to determine the device name before disabling the host network. You may use the `ifconfig` command for that purpose.

```
$ sudo ifconfig
```

Next, disable the network of the hypervisor host.

```
$ cd /hv/VMs/vm1
$ netconf -off
```

You need to turn off IPv6 as follows.

```
$ sudo gedit /etc/sysctl.conf
```

Insert the following lines at the bottom of this file:

```
net.ipv6.conf.all.disable_ipv6=1
net.ipv6.conf.default.disable_ipv6=1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Make this effective:

```
$ sudo sysctl -p
```

Then, configure the guest.

```
$ cd /hv/VMs/vm1
$ gedit vmconfig.sh
```

Change the respective configuration values.

```
netif_mode=2
netif_m=...
```

Caution: You must define the network device that shall be used in the guest by setting the parameter netif_m to the name you have determined above.

Hint: To re-enable the host network in automatic mode, run the netconf command again:

```
$ cd /hv/VMs/vm1
$ netconf -auto
```

And change the netif_mode configuration value.

```
$ cd /hv/VMs/vm1
$ gedit vmconfig.sh
```

```
netif_mode=1
```

2.16.4 Virtual network configuration

The hypervisor provides a virtual network. The hypervisor host and all the guests are connected to this virtual network.

The IP addresses are set to fixed values.

The default IP addresses are:

Hypervisor host: 192.168.157.1

First RTOS: 192.168.157.2

Windows guest: 192.168.157.3

Hint:

The host virtual network IP address is initially set when calling the inithv.sh script.

There is a call to the netconf command to set the IP addresses an manage these manually.

2.17 Backup

You can create a backup of your hypervisor installation.

This backup will store all modifications you have made in the /hv folder, it will not backup the complete hypervisor disk!

All files (except some pre-installed *.iso, *.deb files etc.) will be stored in an archive.

Optionally, this archive may also include guest virtual machine images (*.qcow2), though it is recommended to backup those files separately.

The backup script `bkup.sh` takes 3 arguments:

```
arg 1: -vmimg    optional: also include guest VM images
                 (backup will take a long time to finish and the backup␣
↪file will be big!)
arg 2: -noinst   optional: do not include all hypervisor installation files
                 (e.g. specific Linux kernel, Windows guest support␣
↪installation)
arg 3: bkupfile  backup file (only the prefix, e.g. mybackup, do not use␣
↪the full filename like backup.tar.gz)
```

Run the following command to create a backup file.

```
$ cd /hv/config
$ sudo ./bkup.sh /tmp/mybackup
```

This will create a backup file `/tmp/mybackup.tar.gz` (in case guest images are included using -vmimg, the extension would be tar).

To restore a previously taken backup, copy the backup file and the restore script to a folder on the PC.
The script has to be made executable.
The restore script `rstr.sh` takes 3 arguments:

```
arg 1: bkupfile   backup file (only the prefix, e.g. backup, do not use␣
↪the full filename like backup.tar.gz)
arg 2: restdir    folder where to restore the previously taken backup
arg 3: -force     if this option is given, existing files will be␣
↪overwritten without any confirmation
```

Run the following command to restore a previously created backup.
It is recommended to remove or rename the /hv folder on the disk where you want to restore the backup before executing the restore command.
The example below assumes the files `/hv/config/rstr.sh` and `mybackup.tar.gz` have been copied to the Downloads folder in the home directory.

```
$ cd ~/Downloads
$ chmod +x rstr.sh
$ sudo ./rstr.sh mybackup /hv
```

Caution: You may have to re-run the auto configuration script after the backup has been restored

3 Remote Debugging

3.1 RT-Linux (Visual Studio)

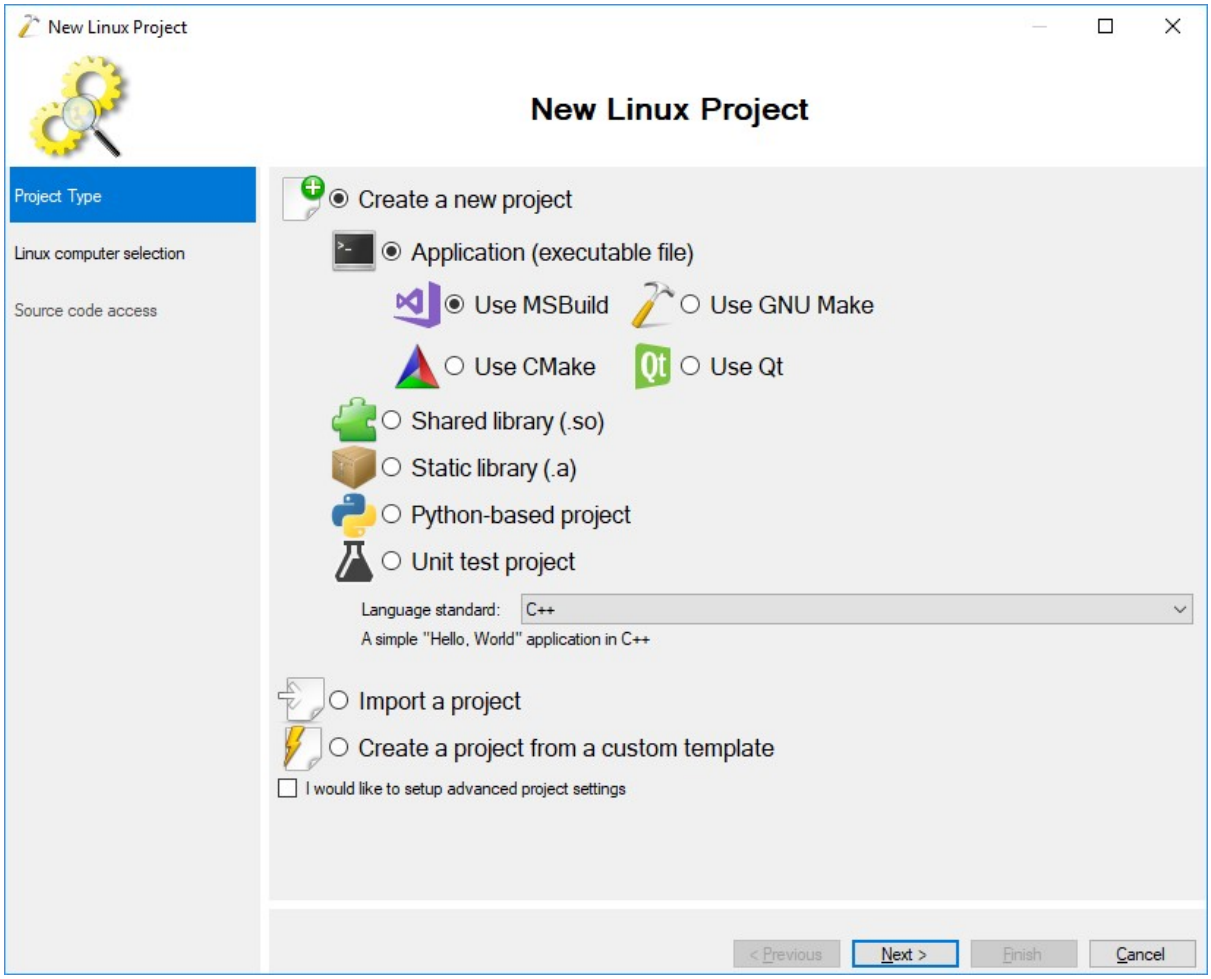
You may use the third party VisualGDB solution for development and debugging of RT-Linux applications using Microsoft Visual Studio.
An evaluation license can be obtained here: http://visualgdb.com/download

After installing VisualGDB restart Visual Studio to get the latest VisualGDB package updates.
The toolchain to be used can be downloaded from http://software.acontis.com/LxWin/MinGW.zip
For x64 development download the following toolchain
http://software.acontis.com/LxWin/MinGw64.zip

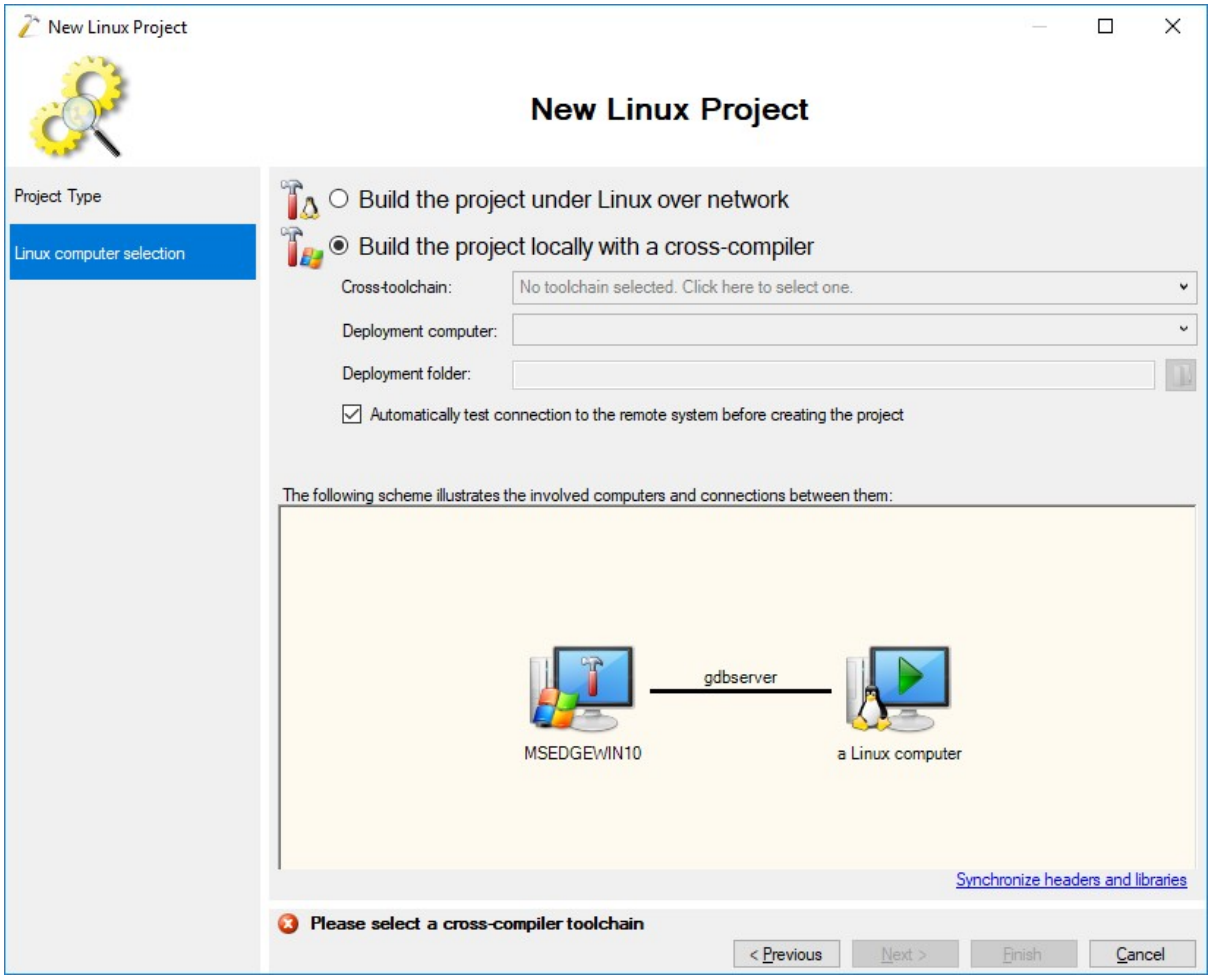
> Extract the zip file into `C:\` (If you want extract into another directory, ensure there are no blanks!)

3.1.1 Create a new project

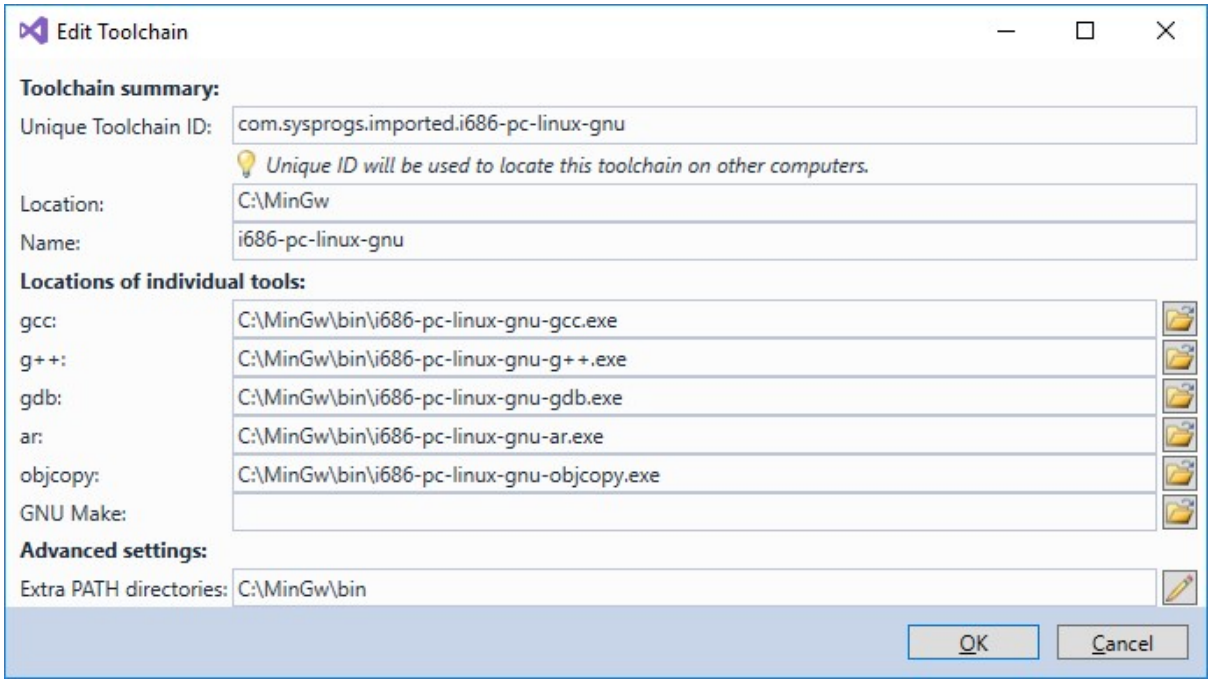
- Start the Hypervisor, configure to run the Linux RTOS and start RT-Linux (this is described in the Quick Start Tutorial).
- Set up network bridging and assure you can reach the RT-Linux OS from your Windows development machine. See chapter *Bridge virtual and physical network* for details. Alternatively you can also use network forwarding, see *Network Forwarding from Windows to the RTOS*
- Start Visual Studio
- Create a new VisualGDB project by using the `Linux Project Wizard`
- Set up the project as Application and use MSBuild
- Set the `Language standard` to `C++`



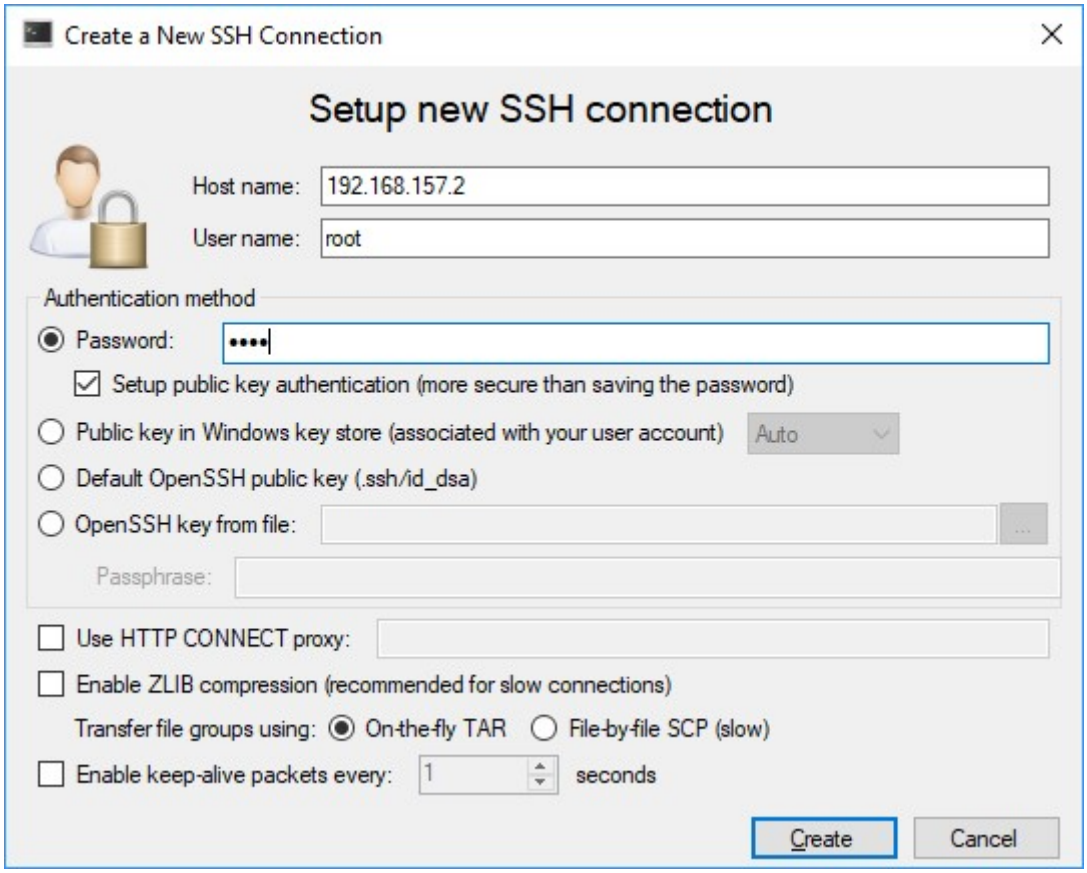
- Select `Build the project locally with a cross-compiler`



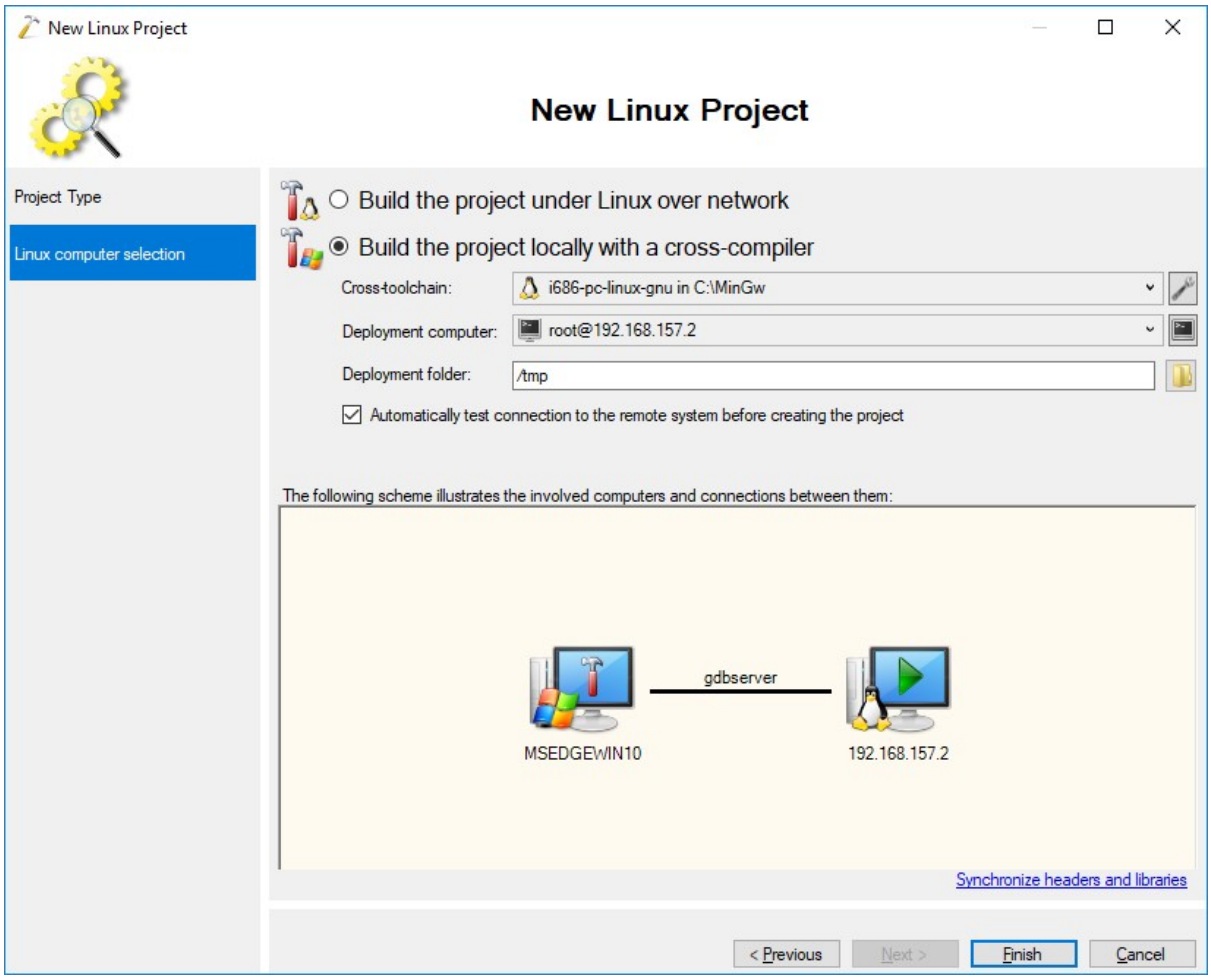
- In the Cross-toolchain field select `Locate a cross-toolchain by finding its gdb.exe` and select C:MinGwbini686-pc-linux-gnu-gdb.exe
- For the 64 bit (x64) toolchain select C:MinGw64binx86_64-pc-linux-gnu-gdb.exe
- Edit the Toolchain dialog looks like:



- In the `New Linux Project`-View, click the drop-down-field `Deployment computer` to create a new SSH connection
- Assure RT-Linux is started before you create the SSH connection! As host name use the IP address of the Linux target. User name and password are both `root`.



- Assure, before finishing the New Linux Project dialog looks like this:



Then press Finish. Accept the `Mismatching environment detected` message with the OK Button.

- Now you can debug the project

3.2 On Time RTOS-32

There are separate tutorials which describe how to set up remote debugging for On Time RTOS-32.

4 Additional Tutorials

4.1 Run shipped RTOS container

Caution: This tutorial is only for completeness and handles some edge cases like VxWorks! See chapter *Installation and Configuration* for the general **Hypervisor-Quickstart-Guide**!

4.1.1 General

RTOS images (containers) are located in the appropriate directory beyond `/hv` (e.g. `/hv/lx` for RT Linux containers).

The first time, when a RTOS container is started, the RTOS Virtual Machine is loaded and the configuration for the RTOS container(s) are loaded. In case a configuration entry has changed, all RTOS containers need to be stopped and the RTOS Virtual Machine to be reloaded.

Configuration files are stored in config files. The following operations are supported via calling the appropriate scripts:

- start RTOS (e.g. `lx.sh` to start the RT Linux image)
- stop RTOS: `rtosstop.sh`
- stop RTOS and RTOS Virtual Machine: `stopall.sh`
- open Debug Console: `dbgcon.sh`

4.1.2 Real-time Linux container

Important: you need at least **256 Mbyte** RAM for the shipped Linux image. Assure the `inithv.sh` script was executed with the appropriate RAM size.

Real-time Linux containers are stored in `/hv/lx`

Start the container using the `lx.sh` script.

From within the real-time Linux OS you may access the Hypervisor filesystem via the `/mnt/rtfiles` mount point. The root directory will point to the Hypervisor directory `/hv/lx/rtfiles`.

4.1.3 VxWorks container

VxWorks containers are stored in `/hv/vx`

Start a VxWorks container using the `vx.sh` script.

File system access

To access the Hypervisor filesystem from within VxWorks, you may use FTP.

For security reasons, by default, there is no FTP server installed.

Follow these instructions to install an FTP server in the Hypervisor.

- `sudo apt-get install vsftpd`
- activate the entry `write_enable=YES` in `/etc/vsftpd.conf`
- Restart the FTP server: `/etc/init.d/vsftpd` restart
- **Create an FTP user target with password vxworks:**
  - adduser target
  - passwd vxworks
  - You may use visudo to add the user target to the sudoers group (duplicate the root entry)
- After executing the above steps, from within VxWorks, you should be able to access the directory `/home/target`

4.1.4 On Time RTOS-32 container

RTOS-32 from “On Time” company is a hard real-time operation system. It is one of the operating systems supported by RTOSVisor.

- **Directories:**

  `/hv`
  this root directory contains all RTOSVisor files and executables.

  `/hv/rtos-32`
  RTOS-32 configuration files and start/stop scripts as well as OS binaries.

  `/hv/rtos-32/rtfiles`
  directory for your `.dlm` files.

- **Most important bash scripts:**

  `/hv/rtos-32/realtimedemo.sh`
  starts RTOS-32 VM and the acontis tool measuring context switch and interrupt latencies.

  `/hv/rtos-32/realtimedemo-debug.sh`
  starts debug monitor that awaits requests from a remote debugger on a Windows PC with installed EcWin, Visual Studio, and RTE Plugin.

/hv/rtos-32/ecmasterdemo.sh
starts RTOSVisor RTOS-32 VM and EtherCAT MasterStack demo.

/hv/rtos-32/stopall.sh
stops VM and RTOSVisor.

/hv/rtos-32/dbgcon.sh
opens interactive RTOS-32 VM console, so you can interact with your app here.

- **Specific scripts used for debugging:**

/hv/hvctl/brvnetset.sh
creates a virtual network bridge on Linux host to forward debugger *TCP/IP/UDP* packets from `LAN1` to RTOS-32 VM. It is required to start this script if you need to perform remote debugging of a RTOS-32 app from another machine.

Hint: See chapter "Bridge virtual and physical network" in the Hypervisor manual for details how to configure the bridge.

/hv/hvctl/brvnetclr.sh
deletes bridge, after the RTOS-32 VM has been stopped.

Start a RTOS-32 container using the `rtos-32.sh` script.

PC Configuration Prerequisites

It is assumed that you already have installed RTOSVisor and configured it as it is described in this `Hypervisor.pdf` file.

It is also assumed, that your Hypervisor PC has two `LAN` ports and you already have assigned `LAN2` to RTOSVisor, as it is described in `Hypervisor.pdf` in the **PCI/PCIe device assignment** section. Ths step is only required if you want to start `EcMasterDemo`, because it requires one `LAN` port to communicate with EtherCAT slaves.

If you want just run other demoes (ex. `realtimedemo`), or want to remotely debug your RTOS-32 app, you can skip this step.

Achieving Hard Real-Time Capabilities

It is important to understand, which factors have influence to the real-time capabilities of your hardware. Most important of them are BIOS Settings (no CPU power saving modes should be activated, no CPU throttling, no variable CPU Frequency, USB Legacy support should also be disabled). Please refer to the RTOSVisor documentation). SMI must be also avoided.

Video Card has usually a huge impact on the real-time capabilities of the real-time system, because it can not only generate SMI but also perform huge background DMA transfers from/to DRAM memory.

For example, if you have Intel i915 video card on your Linux host, its Linux driver can produce significant interrupt/context latencies.

First step, in order to achieve better latencies, is to disable LightDM graphics manager in your Xubuntu Installation of RTOSVisor. Simply run the following command `sudo apt-get remove lightdm` and the reboot your system.

This step converts your system into console only mode, no GUI. But it is not enough, because a video card can do DMA Transfers even without GUI.

The easiest and fastest solution here is to kill its driver as follows:

- establish connection with Hypervisor PC via SSH (port 22)
- kill video driver `sudo rmmod -f i915`. Now the computer monitor cannot be used anymore.
- start realtime demo and make sure, that context switch/interrupt delays much better now (200%+):

```
$ cd /hv/rtos32
$ sudo /realtimedemo.sh
```

How to run a sample preconfigured RTOS-32 App (Realtimedemo)

RTOSVisor has a special tool (`RealtimeDemo`), which can accuratelly measure the real-time capabilities of a machine and the hypervisor. This tool can be also used as a first sample realtime app, to play with the RTOSVisor.

To start it, execute `/hv/rtos-32/realtimedemo.sh` script. This script loads/starts RTOSVisor VM, RTOS-32 OS Image and starts `RealtimeDemo.dlm`. Execute `/hv/rtos-32/stopall.sh` script to stop everything.

Most important parameters here are: *Interrupt Delay* and *Task Delay* (in microseconds). These parameters show the realtime capabilities of the Hypervisor PC and RTOSVisor.

You can press `Ctrl` + `C` to exit from the console into Linux Terminal. Please execute `dbgcon.sh` script to return to the RTOS-32 console again.

It is also easily possible to run the realtime demo in the background, completely detached from the console. So, use `dbgcon.sh` script in this case, to open interactive RTOS-32 terminal from a new console.

How to run `EcMasterDemo` that uses a network card and the EtherCAT stack

It is assumed that your Hypervisor PC has two ethernet ports `LAN1` and `LAN2`. The first one can be used for a *TCP/IP* traffic and/or to establish a debug connection with a DevPC.

It is also assumed, that your second `LAN` port was already assigned to RTOSVisor, as it is described in this `Hypervisor.pdf` in the **PCI/PCIe device assignment** section.

Execute the following steps:

1. Connect `LAN2` port to a EtherCAT slave.
2. `cd /hv/rtos32`
3. `sudo ./ecmasterdemo.sh`

All required files like `EcMasterDemo.dlm`, `EcMaster.dlm`, `emllRTL8169.dlm` are already in the `rtfiles/ecmaster` subdirectory.

How to remotely debug a RTOS-32 app from a Windows PC with Visual Studio

It is assumed, that you also have a second Windows PC (DevPC), where you installs Visual Studio + EcWin + RTE Plugin. DevPC is used to develop your RTOS-32 Application and perform a remote debugging of this app on the Hypervisor PC.

acontis has developed a special Visual Studio plugin that provides the possibility to create a RTOS-32 project, compile, debug it (local or remote) and configures a Visual Studio environment for you.

Requirements for a Windows PC:

* Visual Studio 2015 or newer should be installed

* acontis Rtos32Win/EcWin should be installed

Topics like creating a new RTOS-32 Project are out of scope of this document. Please refer to the official acontis RTOS-32 documentation to find details.

Before we start, it is important to understand, how RTOS-32 Apps work in the RTOS-32 VM. Every app consists of a Loader part (`/hv/rtos-32/Loader.bin`) and a dynamically loaded module in a `.dlm` format (`/hv/rtos-32/rtfiles/yourapp.dlm`). `Loader.bin` is provided with full source code and comes as a separate Visual Studio project. It is automatically generated by the RTE Visual Studio Plugin using the method described above.

But an application, that is is supposed to be debugged, should use a separate bootloader called **RTOS-32 Monitor** (`/hv/rtos-32/debug/Monvmf.bin`).

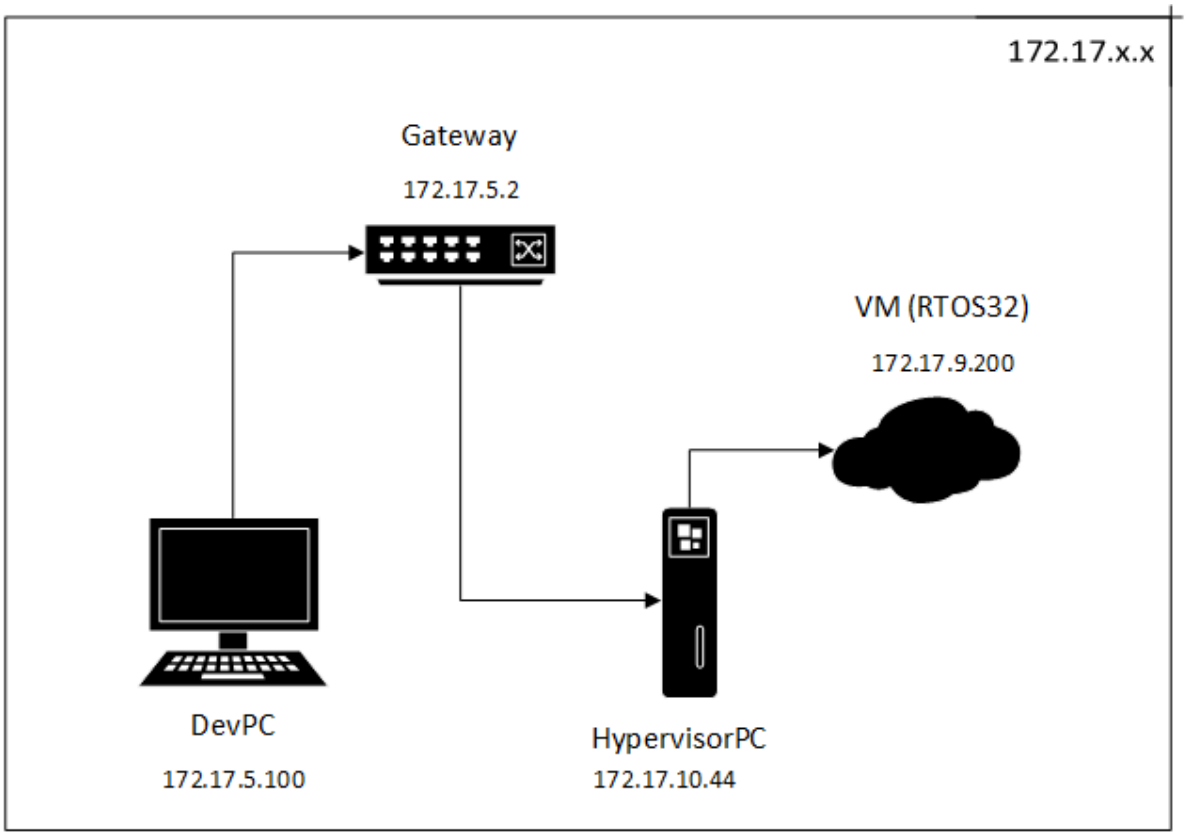
The monitor starts in RTOS-32 VM and listens to a *TCP/IP* traffic from the remote Visual Studio debugger.

The second important thing: all Monitor settings (like an *IP* address to listen to) are always embedded into the `Monvmf.bin` file. So, copy `Monvmf.bin` to the Hypervisor PC from DevPC every time when you change remote *IP* address in the System Manager (the part of EcWin/Rtos32Win).

Network Configuration:

The Hypervisor PC should be configured to peform the remote debugging; scripts and configuration files should contain specific *IP* addresses/masks/subnets. The requirement is that DevPC, Hypervisor PC and RtosVM are in the same subnet, in our example `172.17.x.x`.

Our sample network configuration:



All these *IP* addresses should be set here: `/hv/hvctl/brvnetconfig.sh`

In our example, the gateway is a simple router with running DHCP server, that assigns static `172.17.5.\*` addresses to development PCs in the office and dynamic `172.17.10.\*` addresses to all other computers connected to the router.

VM with address `172.17.9.200` is in a separate subnet, not physically connected to the router. But Hypervisor PC accepts `MASK 172.17.255.255` on its virtual network bridge (read below) and can forward all traffic to and from a VM from the `172.17.x.x` network.

By simple words, the DevPC communicates always with VM. In the VM a special debug stub is started (called **RTOS-32 Debug Monitor**) that listents to the `172.17.9.200` *IP* address.

Open project of a sample demo app (we describe here how to debug `RealtimeDemo`):

1. DevPC: open System Manager and right click `RTOS #1\Application` and choose `Create New Application Project (Debug Only)`
2. Choose `RealtimeDemo` and click *OK* button. Visual Studio projects are generated now and copied to `RtFiles` and other subdirectories of your System Manager workspace.

Prepare a debugger bootloader (RTOS-32 Monitor):

1. DevPC: Select *Remote Debugging* and click *Settings* button near to a grayed *IP* text field.
2. Enter `172.17.9.200` in the **IP** text field and press *OK* button. Please note, now the new RTOS-32 Monitor binary (`Monvmf.bin`) is generated and the target *IP* addresss is embedded into the binary.

Compile the `.dlm` project:

1. DevPC: Click *Open Project with Visual Studio* button.

2. In the Visual Studio two projects are available in the solution explorer: `Loader` and `RealtimeDemo`.
3. Right click every project and click *Build*

Copy binaries to the target (Hypervisor PC):

1. Copy `RealtimeDemo.dlm` from the `RtFiles` directory in your workspace on DevPC to `/hv/rtos-32/rtfiles/debug/` directory.
2. Copy `projects/monvmf/Monvmf.bin` to `/hv/rtos-32/debug/` directory
3. Make sure `/hv/rtos-32/realtimedemo-debug.config` file has correct settings. It should have a valid file server directory (`hv/rtos-32/rtfiles/debug/`) and it should of course include `.config` file for a PCI Card for `LAN2` port, that is assigned to the RTOS VM.

Start RTOS-32 VM and the RTOS-32 Monitor (Hypervisor PC):

1. Execute `/hv/rtos-32/realtimedemo-debug.sh`. Please note, the `.dlm` module was already added to the `realtimedemo-debug.config` in the `Rtos\Loader` section.
2. You should see a RTOS-32 Monitor output. It listens to the IP address `172.17.9.200` to receive the remote debugger traffic.

Please check all your network setings here: `/hv/hvctl/brvnetconfig.sh`

Please note, vnet0 virtual network adapter is created on Linux host, when the VM is started. Please consider this adapter as a virtual PCI network card inside the RTOS-32 VM, that is visible in Linux host as `vnet0`.

By default the `192.168.178.1` address is assigned to this adapter on a Linux side.

Configure a virtual bridge network adapter

Configure a virtual bridge network adapter to forward all incoming debugger traffic from `LAN1` to the virtual network adapter `vnet` in the RTOS VM.

Please execute `/hv/hvctl/brvnetset.sh` script. The virtbr network adapter is created on the linux side.

Start debugging (DevPC):

1. In Visual Studio open file `Loader.cpp`.
2. Locate `main()` function
3. Set a breakpoint.
4. Press `F5` key to start debug.

How to repeat/stop debugging:

1. There is no need to restart the RTOS-32 Monitor if you want to stop the debugging.

Simply click Stop in the Visual Studio and then press `F5` again.

1. It is not needed to copy the RTOS-32 Monitor binary (`Monvmf.bin`) every time you make changes in the loader or a `.dlm` project. Only when the *IP* address is changed.
2. **If you found a bug in your `.dlm` module and need to upload a new verison into the Hypervisor PC, please do the following:**

    1. compile project

2. copy changed `.dlm` binary to the `/hv/rtos-32/rtfiles/debug/` directory.

3. **If you want completely stop your VM, make it in the following sequence:**

   1. Execute `/hv/hvctl/brvnetclr.sh` to remove the bridge
   2. Execute `/hv/rtos-32/stopall.sh`

4.2 Install Windows or Linux guest (SecureBoot)

Attention: See chapter *Installation and Configuration* for the general tutorial **HV-WindowsGuest-Guide**.

This tutorial describes the case with SecureBoot.

4.2.1 UEFI and Legacy BIOS

4.2.2 UEFI

What is SecureBoot

Secure Boot is an interface between UEFI and Operating System. When SecureBoot is activated, it prevents the loading of unsigned boot loaders or drivers

Read More: https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface

Preparing keys for SecureBoot

The key thing in SecureBoot is a platform key (Platform). It establishes relationship between a platform owner and a platform firmware. PK is a self-generated certificated owned by OEM.

Another important key is a KEK key. This key is obtained from an OS manufacturer (for ex. Microsoft) and is used to establish trust relationship between the firmware and OS.

Generating the platform key:

```
openssl req -newkey rsa:2048 -nodes -keyout PKpriv.key -x509 -days 365 -
↪out PK.crt
Generating a 2048 bit RSA private key
....+++
.+++
writing new private key to 'PKpriv.key'
-----
You are about to be asked to enter information that will be incorporated␣
↪into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
```

(continues on next page)

(continued from previous page)

```
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Bayern
Locality Name (eg, city) []:Munic
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Beer Inc
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []: BayernBeer
Email Address []:
```

OVMF supports keys in DER format only. So we need to convert t:

```
$ openssl x509 -in PK.crt -outform der -out PK.der
```

Download Key Exchange Key (KEK): `MicCorKEKCA2011_2011-06-24.crt` https://go.microsoft.com/fwlink/p/?linkid=321185

Download Signature Database (allows Windows to boot): `MicWinProPCA2011_2011-10-19.crt` https://go.microsoft.com/fwlink/?LinkId=321192

Download Microsoft signer for third party UEFI binaries via DevCenter: `MicCorUEFCA2011_2011-06-27.crt` https://go.microsoft.com/fwlink/p/?LinkID=321194

Building OVMF with SecureBoot support

By default, OVMF is built without SecureBoot support.

So it is recommended to fetch this project from its repository and build OVMF yourselves.

Install required packages:

```
$ sudo apt-get install build-essential git uuid-dev iasl nasm -y
$ sudo apt-get install iasl -y
$
$ git clone git://github.com/tianocore/edk2.git
$ cd edk2
```

Prepare build tools:

```
$ git submodule update -init
$ make -C BaseTools
$ .edksetup.sh
$ make -C ./BaseTools
$ export EDK_TOOLS_PATH=/home/rte/edk2/BaseTools
$ .edksetup.sh BaseTools
```

Edit `Conf/target.txt`:

```
ACTIVE_PLATFORM = OvmfPkg/OvmfPkgX64.dsc
TARGET_ARCH = X64
TOOL_CHAIN_TAG = GCC5
```

Build OVMF with SecureBoot support:

```
$ OvmfPkg/build.sh

    -a IA32 -a X64

    -D SMM_REQUIRE -D SECURE_BOOT_ENABLE

    -D FD_SIZE_2MB -D EXCLUDE_SHELL_FROM_FD
```

Binaries can be found in the `Build` directory.

Embedding SecureBoot keys to OVMF

Create a `OVMF-SecureBoot` directory and copy `Build/OvmfX64/DEBUG_GCC5/FV/OVMF_CODE.fd` and `Build/OvmfX64/DEBUG_GCC5/FV/OVMF_VARS.fd` to this directory.

Create a `hda` subdirectory and copy all generated and downloaded keys to this subdirectory.

Run qemu:

```
$ cd OVMF-SecureBoot
$ qemu-system-x86_64 -L .

    -drive if=pflash,format=raw,readonly,file=OVMF_CODE.fd

    -drive if=pflash,format=raw,file=OVMF_VARS.fd

    -hda fat:hda

    -net none
```

After booting you get to a UEFI shell. Type `exit`.

1. Go to Device Manager / Secure Boot Configuration / Secure Boot Mode and change from **Standard Mode** to **Custom Mode**.
2. PK Options / Enroll PK / Enroll PK Using File and choose `PK.der`
3. KEK Options / Enroll KEK / Enroll KEK Using File and choose `MicCorKEKCA2011_2011-06-24.crt`
4. DB Options / Enroll Signature / Enroll Signature Using File and choose `MicWinProPCA2011_2011-10-19.crt`
5. Repeat last step and choose `MicCorUEFCA2011_2011-06-27.crt`

The Secure Boot Mode should be **Enabled** now.

Exit from BIOS, shutdown the machine.