



AlphaVM for Windows User Manual

Date: 19-Sep-2014

Author: Artem Alimarin

Version: 1.4.14

© 2014, EmuVM.

TABLE OF CONTENTS

1	Installation.....	3
1.1	Host platform requirements.....	3
1.1.1	Requirements for AlphaVM-Pro.....	3
1.1.2	Requirements for AlphaVM-Free	4
1.1.3	Requirements for multiple instances	5
1.2	Obtaining the software	5
1.3	Installation procedure.....	5
1.4	Installed components.....	6
1.4.1	.NET Framework 4	6
1.4.2	PuTTY terminal emulator	6
1.4.3	WinPcap Ethernet driver	6
1.5	Uninstallation.....	7
2	Configuration.....	7
2.1	System configuration	7
2.2	CPU Configuration.....	9
2.3	Memory configuration	10
2.4	SCSI Controller Configuration	11
2.5	Disk configuration	11
2.5.1	Disk names in AlphaVM SRM console	13
2.5.2	Performance considerations.....	13
2.6	CDROM configuration	14
2.7	SCSI Tape configuration.....	14
2.8	SCSI Pass-Through (aka Direct SCSI, aka Generic SCSI) Configuration	16
2.8.1	SCSI Tapes	18
2.8.2	iSCSI devices.....	18
2.8.3	Non-SCSI disks.....	18
2.8.4	Booting from a SCSI Pass-Through device	19
2.9	Serial port configuration.....	19
2.10	Ethernet configuration	20
2.10.1	Communication between the host and AlphaVM.....	22
2.11	VM logging configuration	23

2.12	VM launching configuration	23
2.13	Licensing information.....	25
2.13.1	Configuring for evaluation	27
2.13.2	Configuring with USB dongle on the local machine	27
2.13.3	Configuring with USB dongle on a remote machine	27
2.14	Configuration of multiple instances.....	27
3	Emulator operation.....	29
3.1	Running a VM from the GUI launcher.....	29
3.1.1	Starting the emulation.....	29
3.1.2	Stopping the emulation	29
3.2	Running a VM from command line	29
3.3	Running a VM as a service	29
4	Usage example	30
4.1	OS installation on a new disk.....	30
5	Migration.....	31
5.1	Migration by copying disks	31
5.1.1	Copying disks using a Live Linux CD.....	31
5.1.2	Migration of OpenVMS using backup /image.....	31
5.1.3	Copying disks on OpenVMS	32
5.1.4	Copying disks on Tru64	32
5.2	Migration by reinstalling software from scratch	32

1 Installation

1.1 Host platform requirements

1.1.1 Requirements for AlphaVM-Pro

The general requirements are as follows:

- The operating system must be Windows x64 6.0 or higher:
 - Windows Server 2008,
 - Windows Server 2008R2

- Windows Server 2012,
- Windows Vista,
- Windows 7,
- Windows 8.

We recommend a server operating system Windows 2008 or 2012. We do not recommend Vista.

- The host system must have a CPU, which supports CMPXCHG16B, SSE3 and SSSE3 instruction set features. These features are not available only in very old CPUs.
- The host CPU performance has a direct influence on the emulated system performance. The actual host CPU type depends on your performance requirements. It is recommended to use fast host CPUs (at least 3GHz).
- The product requires a reserved host CPU core for each emulated CPU. This means that a single CPU emulator requires at least a dual-core host system. A dual CPU emulator requires a 3 core system. We advise a double amount of cores with respect to the emulated CPUs.
- Requirements with enabled hyper-threading are not yet available. ES40 with four CPUs has been tested to run fine on an i7 with 8 hyper-threads.
- The host memory requirement depends on the emulated Alpha memory size and other emulator settings. The general requirement is as follows: if the emulated system has N GB memory, than the host system must have at least N + 2GB memory. Besides the emulated memory size, the following factors may influence the required amount: number of CPUs, number of disks, caching settings for disks. We advise at least N + 4GB. Note that JIT2 is memory consuming.
- Emulated Ethernet controllers are mapped to host Ethernet adapters. Note that mapping to Wireless Ethernet controllers does not always work. It is advisable to have a separate host NIC for each emulated NIC.

The actual requirements are very specific for a given configuration and workload. Contact our support to get an advice about hardware required for your configuration.

1.1.2 Requirements for AlphaVM-Free

The product can run on a system with the following characteristics:

- The operating system must be Windows x64 6.0 or higher. It advised to use W7 or its server analogue. Note that the last release that supports 32-bit and Windows XP is 1.3.x. Starting from 1.4.0 we do not support XP and 32-bit any more.
- The host system must have a CPU, which supports CMPXCHG16B, SSE3 and SSSE3 instruction set features. These features are not available only in very old CPUs.
- The host CPU performance has a direct influence on the emulated system performance. AlphaVM-Free has a limited performance comparing to AlphaVM-Pro. Consider AlphaVM-Pro if performance is important. Its CPU is around 10 times faster than that of AlphaVM-Free.
- The host system must have at least 2 cores.

- The minimal host memory is 2GB.
- The emulated Ethernet controllers are mapped to host Ethernet adapters. Note that mapping to Wireless Ethernet controllers does not always work. It is advisable to have a separate host NIC for each emulated NIC. However, the emulator can share NIC with the host system.

1.1.3 Requirements for multiple instances

It is possible to run multiple instances of the emulator on the same machine. The requirements in this case just add up.

For instance, if you run two similar virtual machines on the same host, the requirements are double of the single instance requirements.

1.2 Obtaining the software

To obtain the software please visit our website

<http://emuvvm.com>

or contact us per e-mail

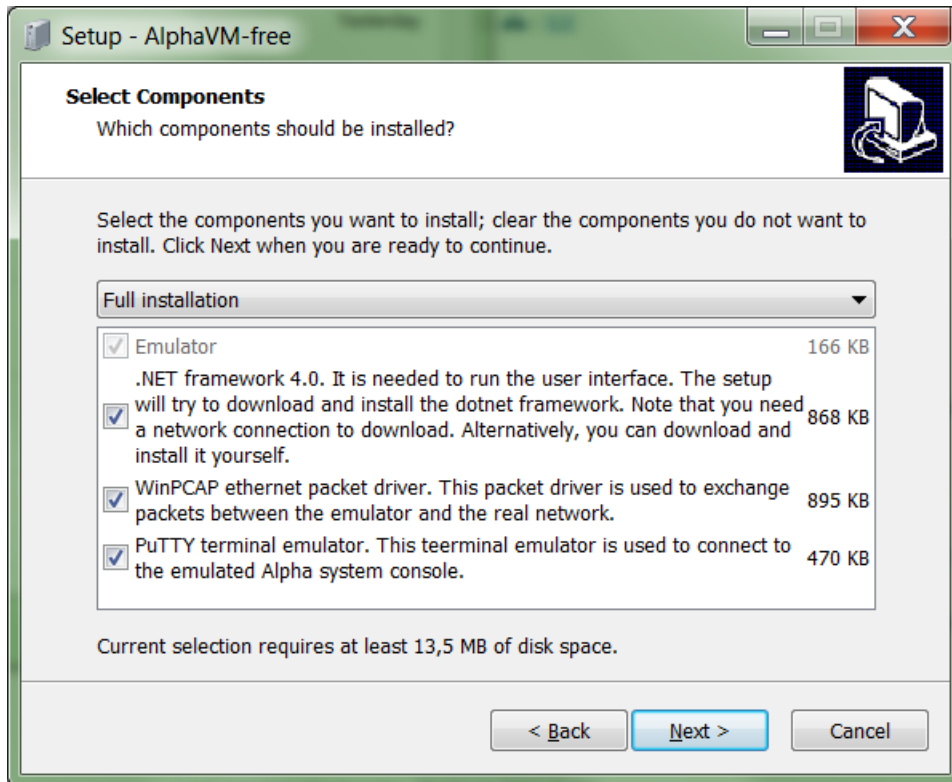
mailto://emuvvm_sales@emuvvm.com

The free version can be downloaded from the website. The commercial version requires a USB license key to run. The key and the software will be sent to you when you purchase the software.

1.3 Installation procedure

To install the emulator run the installer (setup.exe). Then just follow the installation procedure.

The installer offers to choose the installation type. We recommend the full installation. In this case the emulator will be able to offer all of its features. The optional components include PuTTY terminal emulator a WinPcap Ethernet packet driver. Below are some details on these sub-packages.



1.4 Installed components

1.4.1 .NET Framework 4

The installer will check and install the .NET Framework. It is required for proper operation of the user interface. The installer does not contain the full redistributable package of the framework. Instead, it will try to perform a network download and installation. If needed, you can install the framework before the installation.

1.4.2 PuTTY terminal emulator

PuTTY terminal emulator is used to connect to the emulated Alpha serial console. The user can use another terminal emulator. However, we recommend installing PuTTY anyway as a fallback. The putty executable will be put in the same directory where the emulator is installed.

1.4.3 WinPcap Ethernet driver

WinPcap Ethernet Packet Driver is used to pass packets between the emulator and the real network. This is widely used driver certified to work in Windows operating systems. You may want to disable installation of WinPcap if it is already installed. This is especially useful if you want to keep another version of software.

1.5 Uninstallation

To uninstall the emulator use the "uninstall" icon provided in the program menu reachable from the Start menu or launch the un-installation otherwise.

2 Configuration

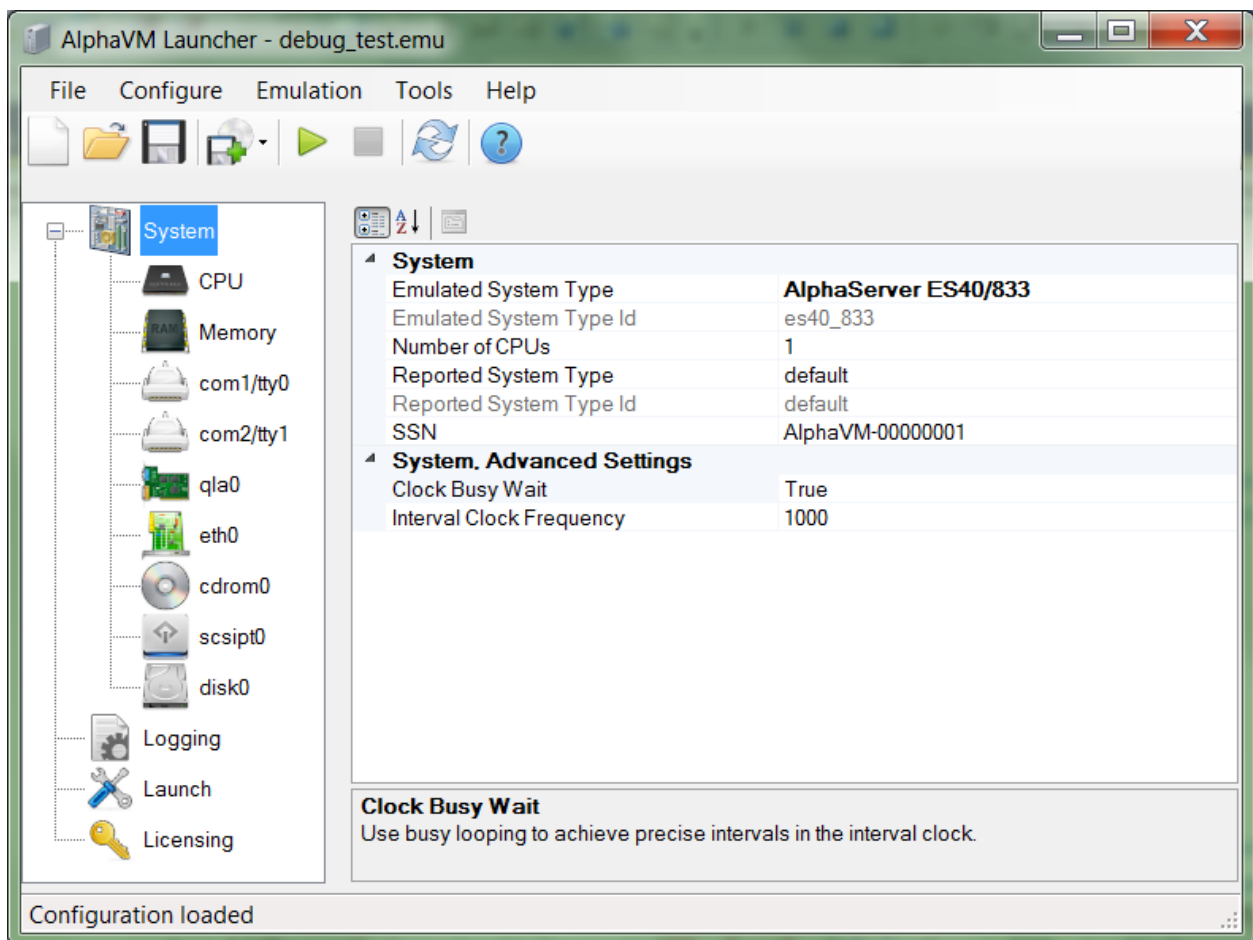
The emulator must be configured before it can be used. The configuration specifies properties of the emulated system.

The emulator is configured by means of the elements of the user interface. The configuration tree allows selecting objects to configure. The property grid on the right side allows specifying properties of the selected object.

The configuration can be loaded and saved via the file menu or via the toolbar.

2.1 System configuration

System configuration screen enables configuration of the emulated Alpha system.



Configuration properties:

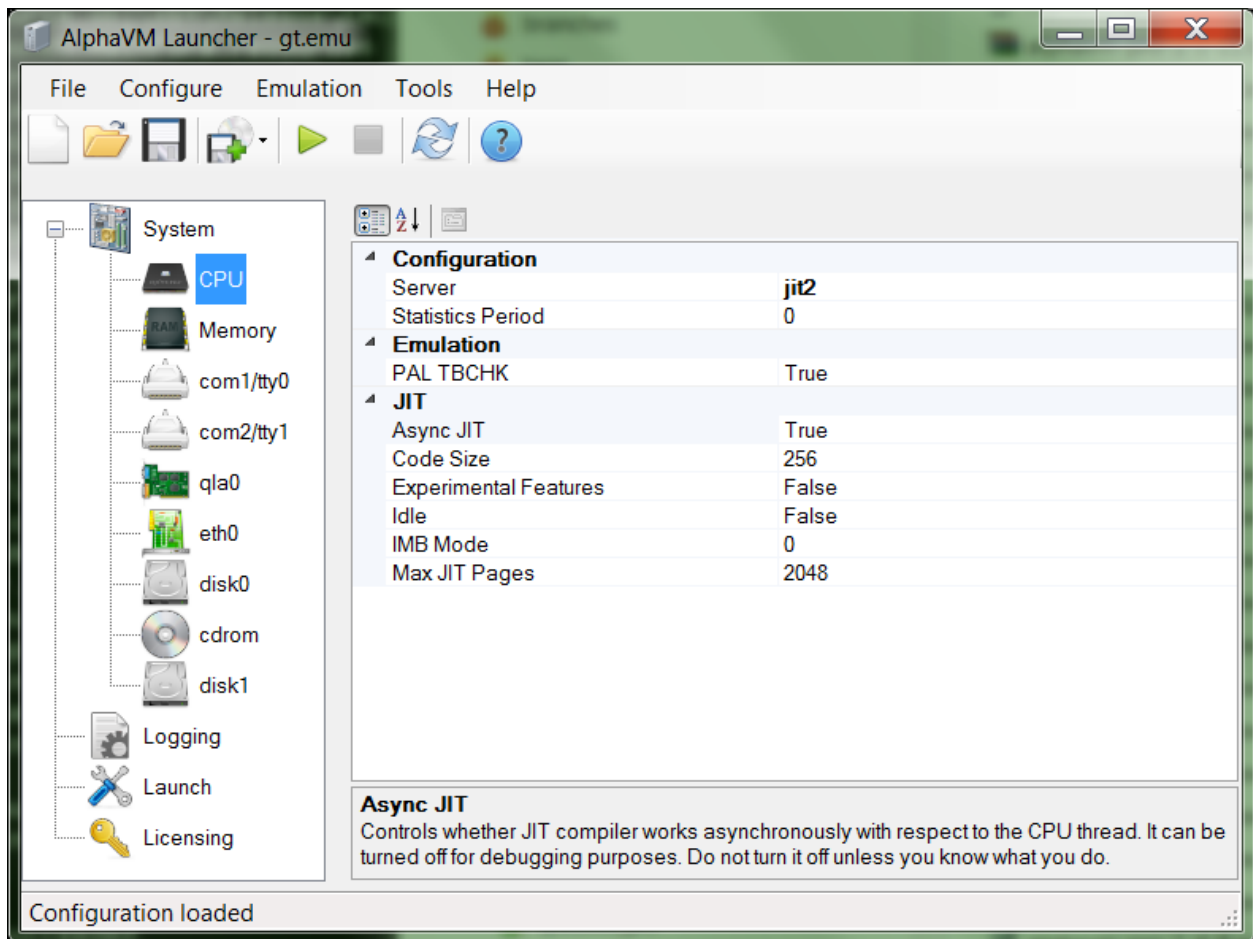
- **SystemType** specifies the type of the emulated system. Currently the following systems are supported:
 - AlphaServer DS10 466MHz, model 1839 (ds10_466)
 - AlphaServer DS10 616MHz, model 1970 (ds10_616)
 - AlphaServer DS10L 466MHz, model 1961 (ds10l_466)
 - AlphaServer DS10L 616MHz, model 1962 (ds10l_466)
 - AlphaServer DS20 500MHz, models 1839, 1920 (ds20_500)
 - AlphaServer DS20E 500MHz, models 1840, 1921 (ds20e_500)
 - AlphaServer DS20E 667MHz, models 1939, 1940 (ds20e_667)
 - AlphaServer DS20E 833MHz, models 1982, 1983 (ds20e_833)
 - AlphaServer DS20L 833MHz, model 2006 (ds20l_833)
 - AlphaServer ES40 500MHz, models 1813 -1816 (es40_500)
 - AlphaServer ES40 500MHz, models 1817 -1820 (es40_667)
 - AlphaServer ES40 833MHz, models 1984 – 1987 (es40_833)
 - AlphaStation XP900 466MHz, model 1879 (xp900_466)
 - AlphaStation XP900 500MHz, model 1821 (xp1000_500)
 - AlphaStation XP900 667MHz, model 1822 (xp1000_667)
 - AlphaStation XP900 750MHz, model 1922 (xp1000_750)
- **SystemTypeid** shows an internal system type identifier for **SystemType**. Launcher passes this identifier to the VM to select a system to emulate.
- **ReportedSystemType** specifies the type of the system reported to the OS informational routines (like licensing). This option allows specifying systems that are not actually implemented. By default the same system information is returned as specified by **SystemType**. The value **default** instruct the system to use the same type as the specified by **type**. See values in the drop down menu of the Launcher. Other systems can be added on request.
- **ReportedSystemTypeid** shows an internal system type identifier for **ReportedSystemType**. Launcher passes this identifier to the VM to select a system.
- **NumCPUs** specifies the number of CPUs in the emulated system. Please note that on a 32-bit Windows multi-processor configuration is not supported. The emulator reserves one core of the host system for each emulated CPU. The emulator needs at least one core for some bookkeeping and IO processes. Thus, you need a dual core system to run with one emulated CPU, and at least 3-cores to run a dual CPU configuration. The maximal number of CPUs depends on the emulated system and on the product license. The default number of CPUs is 1.
- **SSN** specifies the system serial number of the emulated system. SSN is often used by third party software to identify the hardware for licensing purposes. This value is a string of max 16 characters long.
- **IntervalClockFreq** specifies the interrupt clock frequency in Hz. Interrupt clock frequency specifies the number of timer interrupts per second. Please do not change this value unless you know what you are doing. This value can affect stability of the system. The standard Alpha frequency is 1000 interrupts/second. However, this frequency can be changed for performance tuning reasons. It is communicated to the operating system via HWRPB. OpenVMS and Tru64

adjust to this value. For performance reasons, it could be better to set this value to 100. Currently Linux does not seem to work correctly with non-standard values.

- **Busy Clock Wait** specifies whether to use a busy loop to make the clock intervals precise. The default value is True, which corresponds to the behavior of the product prior to introduction of this option. This option is introduced to prevent excessive CPU usage by the timer when the host is not capable of providing reliable intervals. Disabling busy loops can be useful when running on a host with scarce CPU resources or on a virtual host that is poorly scheduled.

2.2 CPU Configuration

CPU configuration node is used to configure all CPUs in the system. All CPUs in the system will get the same properties.



Configuration properties:

- **Server** specifies a CPU server to be used. Currently there are three servers available:
 - **Basic** server is a server with a basic performance. This server is the only available for AlphaVM-free.
 - **JIT1** is the server with the performance on the level of fast EV4. It is based on JIT technology.

- **JIT2** is the server with the performance on the level of EV6.
- **Statistics Period** specifies the period between the CPU statistics dumps. For the workload profiling purposes the CPU can dump statistics over a time period. The default value is zero, which means that the dumps are disabled. Do not turn this option on for a production system.
- **TBCHK** specifies whether PAL TBCHK instruction is implemented. By default it is not implemented.
- **Async JIT** specifies whether JIT compilation process is synchronous with respect to the CPU or asynchronous. By default it is asynchronous. The synchronous mode is needed mostly for debugging.
- **Experimental Features** enables some JIT optimizations that are not yet released. These features are available on the field test basis and are not subject for product support. Do not enable the experimental features in production environment.
- **Idle** when enabled, instructs the emulator to release the host CPU when the guest OS is idle. Most version of OpenVMS and Tru64 5.1b are currently supported. Note that this feature may negatively affect the performance of some IO-bound loads. The feature is currently experimental. Do not use it for production environment.
- **IMB Mode** this is an advanced feature for performance tuning. Do not change it unless advised to do so by EmuVM.
- **Max JIT Pages** specifies the maximal number of JIT pages that can be simultaneously active in the system. Each JIT page corresponds to a single page of Alpha code. Please do not change unless you know what you are doing.
- **Code Size** specifies the default size in KB for memory allocation of code chunks. The default is 256k. Too big chunks can cause excessive memory consumption. Too small chunks can degrade the system performance due to frequent allocation. The advised values are in range 128 - 1024. The value is rounded up to 64k.

2.3 Memory configuration

Configuration properties:

- **Memory Size** specifies the RAM size in megabytes of the emulated system. The amount of memory you can use here depends on the amount of memory on your host computer. It is recommended to have at least 2GB of host memory. Maximal memory size depends on the emulated system and on the product licensing. The default size is 128M.
- **Lock** specifies whether to lock the guest memory in the host memory. Locking means that the pages are not unloaded from the memory by swapping. Locking can degrade performance, because other pages will be offloaded. This is an advanced option added for experimentation purposes. Locking is by default off. Note that you can only lock pages when the working set is large enough. If the VM fails to lock pages and logs an error and continues without locking.

2.4 SCSI Controller Configuration

SCSI controllers can be added using the “Configure” menu and removed using a menu that appears on a right-click. The controllers have consecutive fixed names: *qla0*, *qla1*, ... Only the last controller can be removed to preserve this naming sequence.

The number of plugged controllers determines the number of SCSI buses available. You can select a SCSI bus in the SCSI device configuration using the bus number option.

Some Alpha systems have one or two built in SCSI controllers. These controllers will be present in the resulting system even if they are not present in the Launcher GUI configuration. Since the number of available buses in the GUI depends on the number of controllers available in the GUI, the controllers must be present to make the buses available.

The configuration of built-in controllers is given by *qla0* and *qla1*. If the system does not have built-in controllers, then *qla0* and *qla1* cause controllers to be loaded in the available system slots.

Configuration properties:

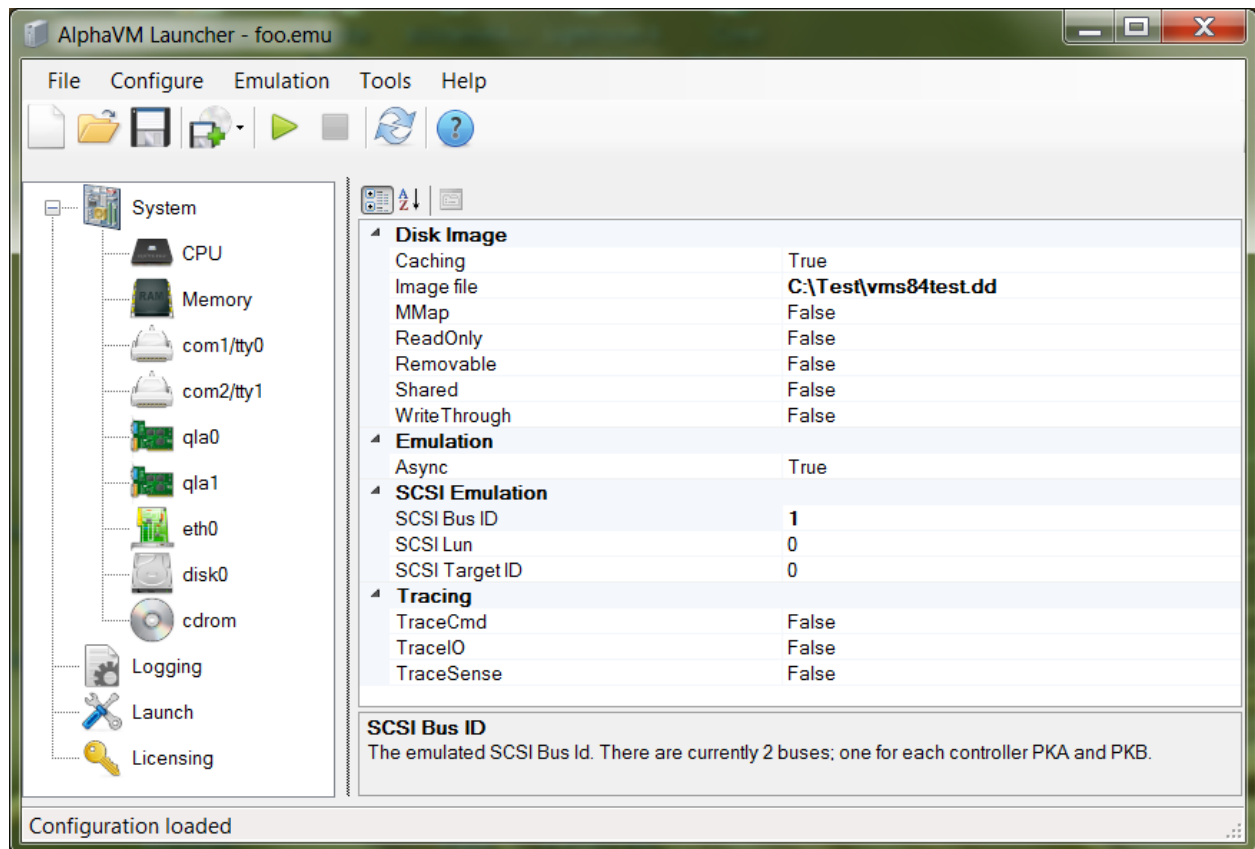
- The option **SCSI ID** specifies the SCSI ID of the controller. Values are 0 .. 15. The default value is 7.
- The option **Slot** specifies the PCI slot to which the adapter is plugged. The default behavior is automatic; thus, you do not have to specify this option or know about slots. The automatic behavior covers most cases with OpenVMS. Tru64, however, is very sensitive to changes in the hardware configuration. When you copy your disk images from the real system, it can be required to specify the slots in such a way to reflect the configuration of the real. The number of slots depends on the actual emulated system. The mapping of the slots to PCI hoses and IDSELS also depends on the system.

2.5 Disk configuration

New disks can be added in the Configuration menu. It can be removed or renamed by means of a context sensitive menu available on right-click on the device in the configuration tree. Note that the device name has no meaning for the system; it is only used for convenience. For instance, you can choose it to be the same as your disk label or with the disk name in the SRM or in your guest OS.

The disk image must exist before you can attach it to the emulator. A fresh disk image can be created by means of the Make Disk tool available in the Tools menu. The Make Disk tool just creates an empty disk image file. It does not attach it to the emulator. Therefore, you have to attach it yourself after creation.

Disk configuration table enables configuration of the emulated disk properties.



Configuration properties:

- **Image** specifies a file name of the disk image file used to store disk data. An empty disk image can be created with “Tools/Make disk”. Please note that creation of a disk image does not connect it to the system. After creation, you still need to specify it in the **Image** property of one of the disks. The default value is empty, which means that there is no disk.
- **SCSI ID** specifies the SCSI target ID of the disk. The SCSI ID can have values 0 .. 6, 8 - 15. SCSI ID 7 is reserved for the SCSI controller. All SCSI devices in the configuration must have unique SCSI IDs. The default value is the disk number, which ensures that the disks have unique IDs.
- **SCSI BUS** specifies a SCSI bus to which the device is connected. The buses are numbered from zero. The number of SCSI buses depends on the amount and type of the SCSI controllers in the system. The number of SCSI buses depends on the amount and type of the SCSI controllers in the system. If you see an empty drop down box, it can mean that you have no SCSI controllers loaded. The default value is zero.
- **SCSI LUN** specifies SCSI logical unit of the disk device. The value can be 0..7. SCSI allows several logical devices to be associated with a single bus device. The default is 0.
- **Async** enables asynchronous operation of the disk with respect to the SCSI controller. It is by default **True**. It can make sense to turn it off for very fast memory-mapped IO, when extra context switches may cause extra overhead.

- **MMap** enables memory-mapped IO. The default is **False**. This mode makes sense for some IO loads when the host system has a lot of memory available for mapping.
- **Caching** specifies whether caching of the disk image file is enabled on the host operating system level. The default value is off. In some situations caching can improve IO performance. However, we noticed that disk IO intensive usage with caching enabled leads Windows to use most memory for disk caching, which results in excessive page faults, system trashing, and as the result CPU sanity checks on OpenVMS SMP configurations. There are complicated inter-dependencies of this setting with the amount of available memory and working set settings.
- **WriteThrough** specifies whether write-through mode is enabled on the host operating system level. The default value is off. See the comment on the option Caching.
- **Shared** specifies whether the VM opens the disk image in shared mode. Normally it should be opened in exclusive mode to prevent multiple usage of the same file. The default value is false (exclusive mode), which guarantees that the disk can be modified only by the emulator.
- **ReadOnly** specifies whether the emulated disk is read-only. In this case the VM opens the image in read-only mode. The default value is false (writable)
- **Removable** specifies emulation of a removable disk. The default value is **no**. This option only makes sense for an image file. Essentially, if the image file cannot be open, the “no medium” status is returned instead of “offline”. The file is closed upon the SCSI STOP command with LOEJ bit set. Unfortunately OpenVMS mount /unload does not set LOEJ bit. The following commands can be used on OpenVMS to force unload of a disk (or CDROM):

```
$ rzt:==$sys$etc:rztools_alpha
$ rzt dka100: /stop
```

For Tru64 you can use

```
# scu -f /dev/rdisk/rz4c eject
```

When such disk is mounted, AlphaVM tries to open the file.

2.5.1 Disk names in AlphaVM SRM console

The SRM or VMS disk device name, e.g. dkb1201, is formed as follows:

- The First two letters **dk** designate SCSI disk
- The third letter designate the SCSI controller number **a=0, b =1, ...**
- The number **n** defines SCSI id and logical unit: id=**n**/100, lun = **n** % 100

Thus dkb101 means that the disk is connected to the bus of the second SCSI controller (bus=1), SCSI ID is 12, SCSI Lun is 1.

2.5.2 Performance considerations

For most workloads it is recommended to use asynchronous IO with caching off and write through off.

Caching causes performance degradation for large disks and some operations like large file copies. In this case it causes excessive swapping the host OS level.

Caching can be beneficial when the disk is relatively small comparing to the host RAM.

Disk IO performance depends on multiple factors. Some of them are given below.

- AlphaVM process working set size. The working set limits can be set in the Launch configuration section.
- Host system file cache size. It can be adjusted by the emulator at startup. See the Launch configuration section.
- Whether or not Windows has a paging file.

2.6 CDROM configuration

CDROM configuration is similar to disk configuration.

New CDROM can be added to the system using the Add CDROM button. It can be removed or renamed by means of a context sensitive menu available on right-click on the device in the configuration tree.

The image property here normally specifies a CDROM image file. However, it can also be a physical CDROM name like [\\.\Cdrom0](#).

CDROM does not have write-related properties. ISO images are always opened in read-only mode.

To programmatically eject CD in OpenVMS use the following commands:

```
$ rzt:==$sys$etc:rztools_alpha
$ rzt dka400: /stop
```

2.7 SCSI Tape configuration

AlphaVM supports virtual (logical) SCSI tapes. The tape is emulated using tape image file. A virtual tape drive can be added from the configuration menu. As with other SCSI devices, the SCSI path should be unique.

Currently the virtual tape drive has no button in the UI to load/unload the medium. On OpenVMS please use rztools:

```
$ rzt:==$sys$etc:rztools_alpha
```

Send load command to the tape:

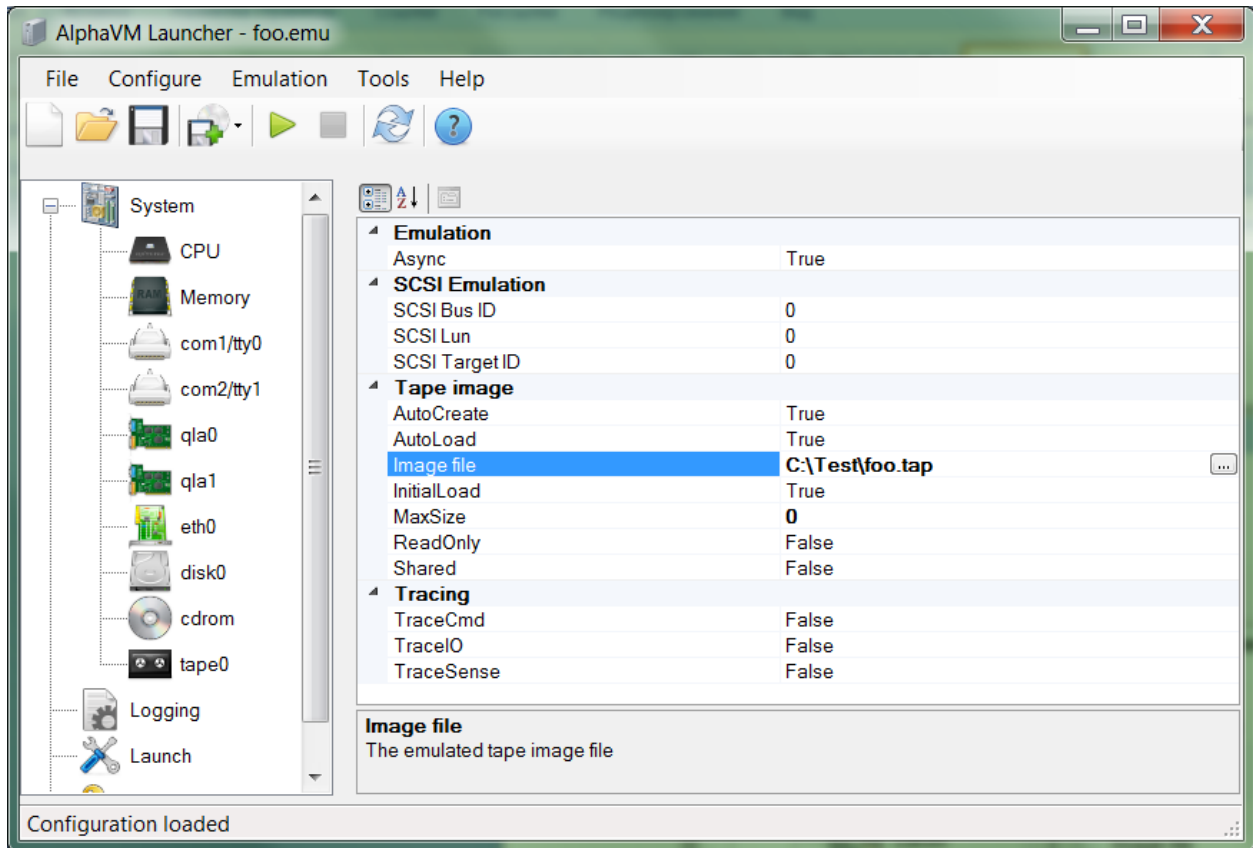
```
$ rzt mka600: /start
```

Send unload command to the tape:

```
$ rzt mka600: /stop
```

On Tru64 the tape can be operated with:

```
# scu -f /dev/rmt0h
```



Configuration properties:

- **Image file** specifies a file name of the tape image file used to store data. An empty tape image can be created by creating an empty file. The default value is empty, which means that there is no medium in the tape drive.
- **SCSI ID** specifies the SCSI target ID of the tape. The SCSI ID can have values 0 .. 6, 8 - 15. SCSI ID 7 is reserved for the SCSI controller. All SCSI devices in the configuration must have unique SCSI IDs. The default value is the disk number, which ensures that the disks have unique IDs.
- **SCSI BUS** specifies a SCSI bus to which the device is connected. The buses are numbered from zero. The number of SCSI buses depends on the amount and type of the SCSI controllers in the system. If you see an empty drop down box, it can mean that you have no SCSI controllers loaded. The default value is zero.
- **SCSI LUN** specifies SCSI logical unit of the disk device. The value can be 0..7. SCSI allows several logical devices to be associated with a single bus device. The default is 0.
- **Async** enables asynchronous operation of the tape with respect to the SCSI controller. It is by default **yes**. Tape is a very slow device, which can block IO when in synchronous mode. This option is provided for debugging.

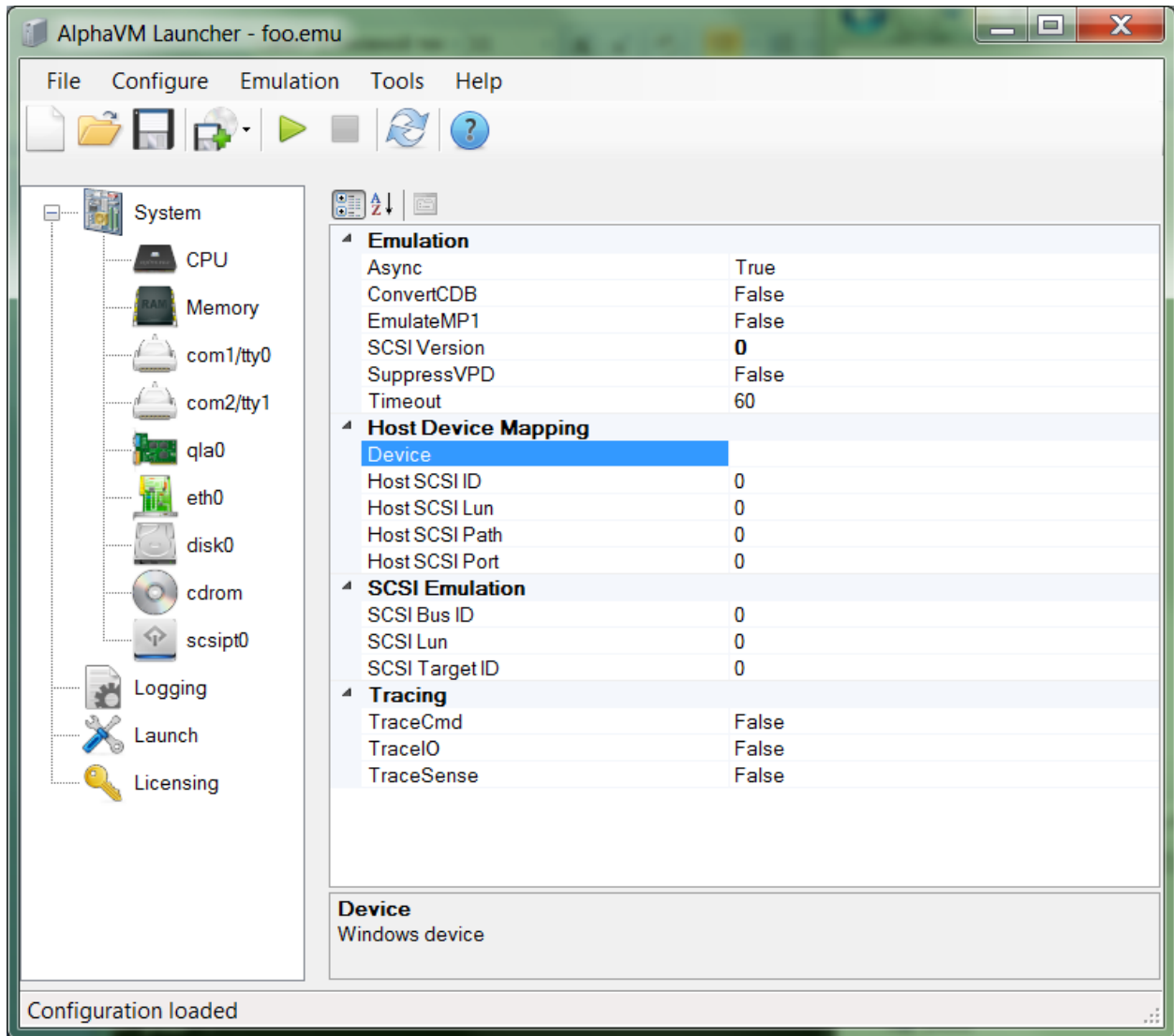
- **Initial Load** specifies whether the tape medium is loaded in the drive when the emulator starts. This is applicable only if the tape image file exists. If the image does not exist, it is considered that there is no medium in the drive.
- **AutoLoad** specified with the tape is automatically loaded on access. This means that the tape file is opened on access. When this option is off, a special load command must be issued to load the tape (see *rztools* commands earlier in this section). When auto-load is on, you do not need those commands. Note that multi-volume backups do not work with auto-load, because you do not have a chance to swap the media: the tape will automatically reopen the same file when it is done with the first volume.
- **AutoCreate** specifies whether an empty tape file created if it does not exist. It is convenient because you do not have to create empty tape files yourself.
- **Max Size** specifies the maximal size of the tape image file. This parameter can be used to create a multi-volume tape backup. The default value is zero, which means no limit.
- **ReadOnly** can be used to protect the tape from writing.
- **Shared** indicates the shared open mode of the tape drive.

2.8 SCSI Pass-Through (aka Direct SCSI, aka Generic SCSI) Configuration

AlphaVM-Pro is capable of accessing the host system SCSI devices by means of so called SCSI Pass Through mechanism. SCSI commands and data are passed between the guest and the hosts systems “as is”. This feature allows to access devices that are not available via the emulation layer. Examples of where the SCSI Pass Through is useful include access to the following devices:

- SCSI tape, which is not available otherwise
- SCSI disk, which can be taken from the real system and attached to the emulator to simplify migration
- ATAPI CDROM, which works as expected
- iSCSI disks
- SCSI devices of other types

Although the intentions of SCSI Pass Through mechanism is to pass commands and data “as is”, some options are available to adjust commands and data in such a way that some useful devices are not rejected by OpenVMS or Tru64.



Configuration properties:

- **SCSI ID** specifies the SCSI target ID of the disk. The SCSI ID can have values 0 .. 6, 8 - 15. SCSI ID 7 is reserved for the SCSI controller. All SCSI devices in the configuration must have unique SCSI IDs. The default value is the disk number, which ensures that the disks have unique IDs.
- **SCSI BUS** specifies a SCSI bus to which the device is connected. The buses are numbered from zero. The number of SCSI buses depends on the amount and type of the SCSI controllers in the system. The number of SCSI buses depends on the amount and type of the SCSI controllers in the system. If you see an empty drop down box, it can mean that you have no SCSI controllers loaded. The default value is zero.
- **SCSI LUN** specifies SCSI logical unit of the disk device. The value can be 0..7. SCSI allows several logical devices to be associated with a single bus device. The default is 0.

- **Async** enables asynchronous operation of the unit with respect to the SCSI controller. It is by default **False**. A slow device in a synchronous mode can block IO. This option is provided for debugging.
- **Device** specifies the Windows device used to access by the SCSI Pass Through. It can be for instance “\\.\Tape0” or “\\.\PhysicalDrive2”. If the value is empty, the following properties are used to identify the device: **Host SCSI Port**, **Host SCSI Path**, **Host SCSI Id**, **Host SCSI Lun**. These parameters are used to access the device through \\.\ScsiX. The default value is empty.
- **Host SCSI Port**, **Host SCSI Path**, **Host SCSI Id**, **Host SCSI Lun** specify the SCSI path of the host device. These options are ignored when the **Device** is specified. Their default values are all zero.
- **Suppress VPD** enables suppression of Vital Product Data in the SCSI Inquiry.
- **Emulate MP1** enables emulation of SCSI Mode Page 1 (read-write error recovery) in case the device does not provide it.
- **ConvertCDB** enables conversion of 6-byte SCSI commands to 10-byte SCSI commands. This mode is can be used to access some ATAPI devices. The default value is **False**.
- **Trace Command** enables logging of commands with sense data. Normally sense data is associated with errors or non-standard situations, so you may wish to enable it to see if something is going wrong. The default value is **False**.
- **Trace Sense** enables logging of all SCSI commands. This option is for debugging. The default value is **False**.

2.8.1 SCSI Tapes

One of the typical uses of the SCSI Pass Through is to access a physical SCSI tape attached to the host.

The tape device name in Windows has the form of \\.\Tape0, which should be used to configure the tape in the emulator.

Alternatively, if the tape device is disabled in the device manager, one can access it using **Host SCSI Port**, **Host SCSI Path**, **Host SCSI Id**, **Host SCSI Lun**. In that case the **Device** option should be empty.

2.8.2 iSCSI devices

The SCSI Pass Through can be used with devices available via iSCSI. This means that the iSCSI initiator must be configured for the device in Windows.

iSCSI target can naturally be on the same host or on a different host.

Linux iSCSI target is compatible with the emulator. It can be directly used.

Windows 2012 SCSI target can require **Suppress VPD** and **Emulate MP1** to be enabled.

2.8.3 Non-SCSI disks

Windows converts SCSI commands to ATAPI device commands, in such a way that SCSI Pass Through can be used to access ATAPI devices. For instance, an ATAPI CDROM drive or a USB stick can be accessed this way.

Note that not all devices respond in a way accepted by OpenVMS or Tru64.

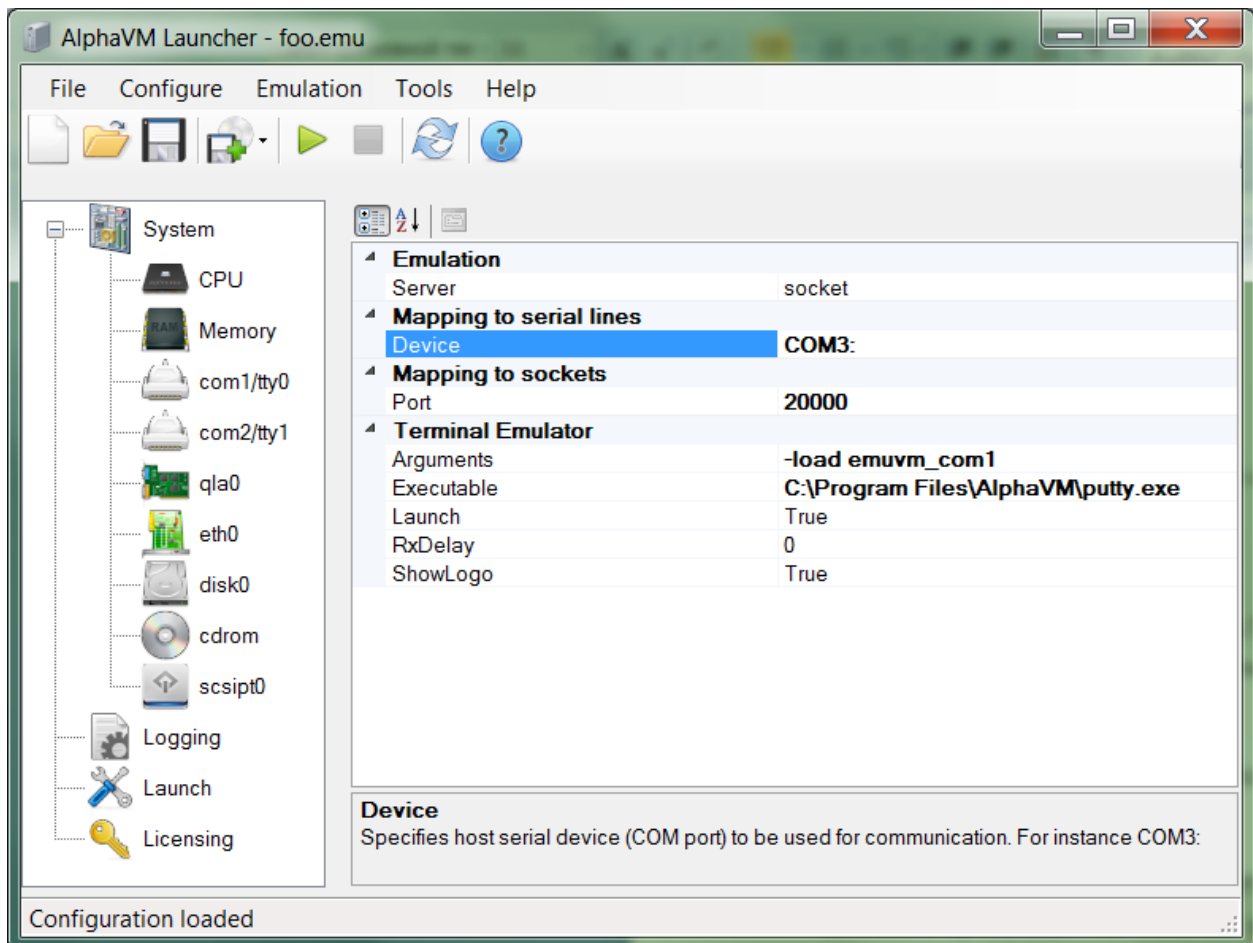
2.8.4 Booting from a SCSI Pass-Through device

Alpha can boot only from a device that supports the block size of 512 bytes. Most CDROM drivers have block size of 2048. However, Alpha CDROM drives are able to switch the logical block size to 512, which enables Alpha booting from these devices.

If your device is not capable of switching to logical block size of 512, AlphaVM will not be able to boot from it.

2.9 Serial port configuration

Serial port configuration pane enables specification of how the port is connected. Currently the port connected only to a virtual terminal. Virtual terminal can be connected to a terminal emulator. We provide a free terminal emulator PuTTY (written by Simon Tatham), which is widely used. This is the default terminal emulator used by AlphaVM. You can choose another terminal emulator and configure it here.



Configuration properties:

- **Server** – selects the way the serial line emulation is served. Currently there are two possible servers: **socket** and **serial**. The socket server maps the serial line to a TCPIP connection.

Normally this connection has a terminal emulator (e.g. PuTTY) on the other side. The serial server maps the emulated serial line to a real host serial line (COM port). By default the value is **socket**. The **serial** server is available only in the professional version of the product.

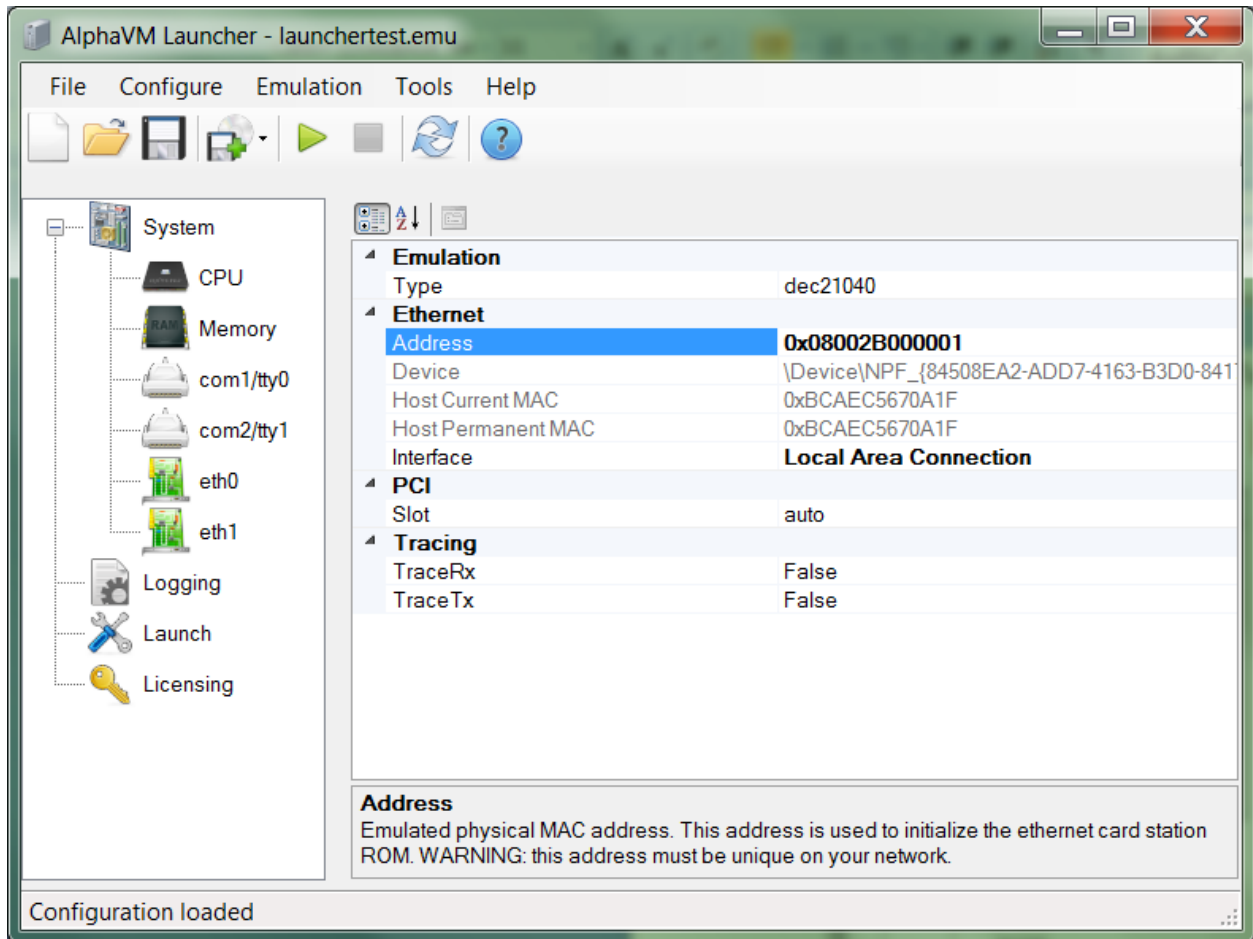
- **Device** specifies the host serial device (COM port) to be used when the server is serial. For instance, *COM3*. This value is ignored when the socket server. Currently the SRM emulator ignores the SRM variable changes related to the serial line. By default it initializes the serial line to 9600 baud, 8 bits, No flow control, no modem control.
- **Port** - is the TCPIP port number used to connect to the terminal emulator. The default value is 20000 for COM1 and 20001 for COM2. This value is ignored for the serial server.
- **Launch** - indicates whether to launch the terminal emulator automatically when the emulation starts. The default value is true.
- **Executable** - The terminal emulator executable. By default, the terminal emulator is PuTTY. PuTTY is delivered together with the AlphaVM product. You can choose another terminal emulator here. The default value is the path to putty in the AlphaVM product location. For instance, "C:\Program Files\AlphaVM\putty.exe".
- **Arguments** - The arguments passed to the terminal emulator executable. AlphaVM provides default putty configurations for COM1 and COM2. In this example the PuTTY configuration *emuvvm_com1* is loaded. Note that the port property used here is the same as the port used by the *emuvvm_com1*. The default value is "-load *emuvvm_com1*" for COM1 and "-load *emuvvm_com2*" for COM2
- **ShowLogo** specifies whether the VM prints logo text on the terminal when then terminal is connected. The default value is true. This can be disabled, which is useful in situations when the logo transmission breaks down the communication protocol.

2.10 Ethernet configuration

The AlphaVM system emulates Ethernet adapter based on DEC21x4x also known as Tulip. You can add these adapters in the "Configure" menu. The added adapters are named automatically: *eth0*, *eth1*, ... Only the last controller can be removed to preserve this naming sequence. The removal option is available on right click.

Some Alpha systems have one or two built-in Ethernet adapters. Configurations *eth0* and *eth1* correspond to the built-in adapters in this case. These adapters will be present in the VM even if not configured in the Launcher GUI.

The emulator communicates with the real Ethernet by means of WinPcap packet filter driver. The user has to provide the information about the connection. In particular, the user has to specify which Windows network interface will be used by the emulator.



Configuration properties:

- **Type** defines the emulated Ethernet controller type. Currently we have just two options available:
 - dec21040 – a 10Mbit controller also known as DE435
 - dec21143 – a 100Mbit controller also known as DE500
- **Address** specifies the permanent physical address of the Ethernet adaptor. The address is specified in a TCPDUMP format, as a hexadecimal number. The default address is 0xAA0004000402, which is AA:00:04:00:04:02. If there are several emulators on your network, make sure their Ethernet controllers have a unique MAC addresses. This is to avoid MAC address conflicts.
- **Interface** specifies the Windows network interface used to connect to the network. Normally you would use “Local Area Connection” or “Local Area Connection 2”. If you wish to disable network, use “No interface”, which disables mapping of the emulated Ethernet to any host network interface.

Note that not all wireless controllers seem to work with the emulator. Please select wired controllers.

The default value is “Local Area Connection”, if it exists; otherwise it is “No mapping”.

- **Device** shows the device name used by WinPcap. This property is read-only and is shown for informational purposes.
- **Slot** specifies the PCI slot to which the adapter is plugged. The default behavior is automatic; thus, you do not have to specify this option or know about slots. The automatic behavior covers most cases with OpenVMS. Tru64, however, is very sensitive to changes in the hardware configuration. When you copy your disk images from the real system, it can be required to specify the slots in such a way to reflect the configuration of the real. The number of slots depends on the actual emulated system. The mapping of the slots to PCI hoses and IDSELS also depends on the system.
- **TraceRX** enables tracing of the packets received by the NIC.
- **TraceTX** enables tracing of the packets transmitted by the NIC.

The emulator shares the same Windows network interface with other Windows programs. However, the emulator maintains a different Ethernet address from Windows. This is necessary that the address is different, so that packets meant for the emulator are not mixed with packets meant for Windows.

For performance reasons you may wish to use a dedicated network interface for the emulator. To achieve this, disable all Windows protocols in Windows NIC settings. In this case Windows will not interfere with the activity of the emulator. You may also wish to use the same Ethernet address as the real address of the Windows NIC.

2.10.1 Communication between the host and AlphaVM

When both the host and the emulator use the same network interface, there is a problem of communication between the host and the guest. This options works only for communication with a remote system. However often it is desired to communicate between the host and the guest. For instance, you may wish an X-server running on the host to connect to the guest. This section describes how to configure network to allow for such communication.

The simplest solution is to use a dedicated host network interface for the emulator. Thus, you should have two network interfaces in your host: one used by the host and one by the emulator. They should both be connected to the same network. In this way packets sent between the host and the emulator go through the real network. It works just like it normally works with a remote machine.

Another solution involves a virtual network within your system to communicate between the host and the emulator. It can be achieved by means of Microsoft Network Bridge. The given solution is tested on Windows 7. All you have to do is to create the bridge and to add your host NIC (e.g. “Local Area Connection”) as a single NIC to it. You should still use your real NIC in the emulator (not the bridge).

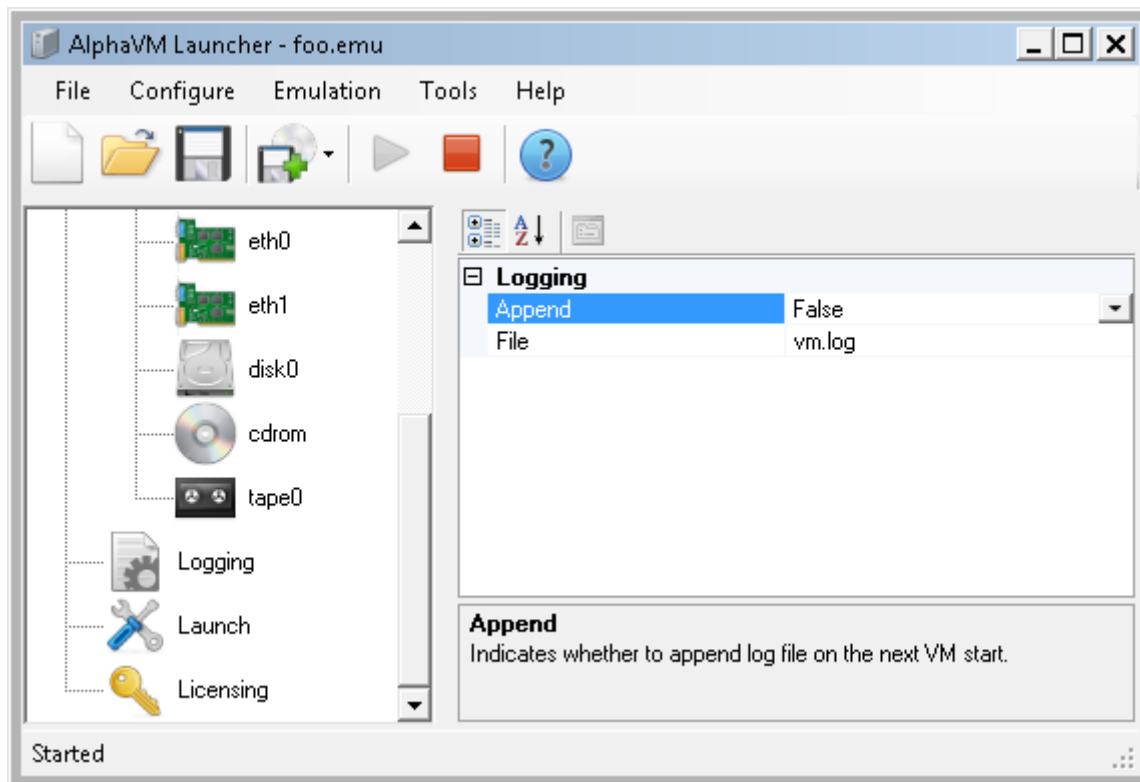
Create the bridge as follows:

- Open Control Panel/Network and Sharing Center/Network Connections
- Select two NICs: your NIC (e.g. Local Area Connection) and any other NIC.

- Right click on selection and choose “Bridge Connections”.
- The bridge is created. You can throw the second NIC out of the bridge in the bridge properties available via the right click.

2.11 VM logging configuration

Virtual machine produces log, which is saved to log file. The logging pane enables configuration of where and how the log file is written. To view the log file use the Tools/VM Log menu item. When reporting a problem with the product, please send us this log file.

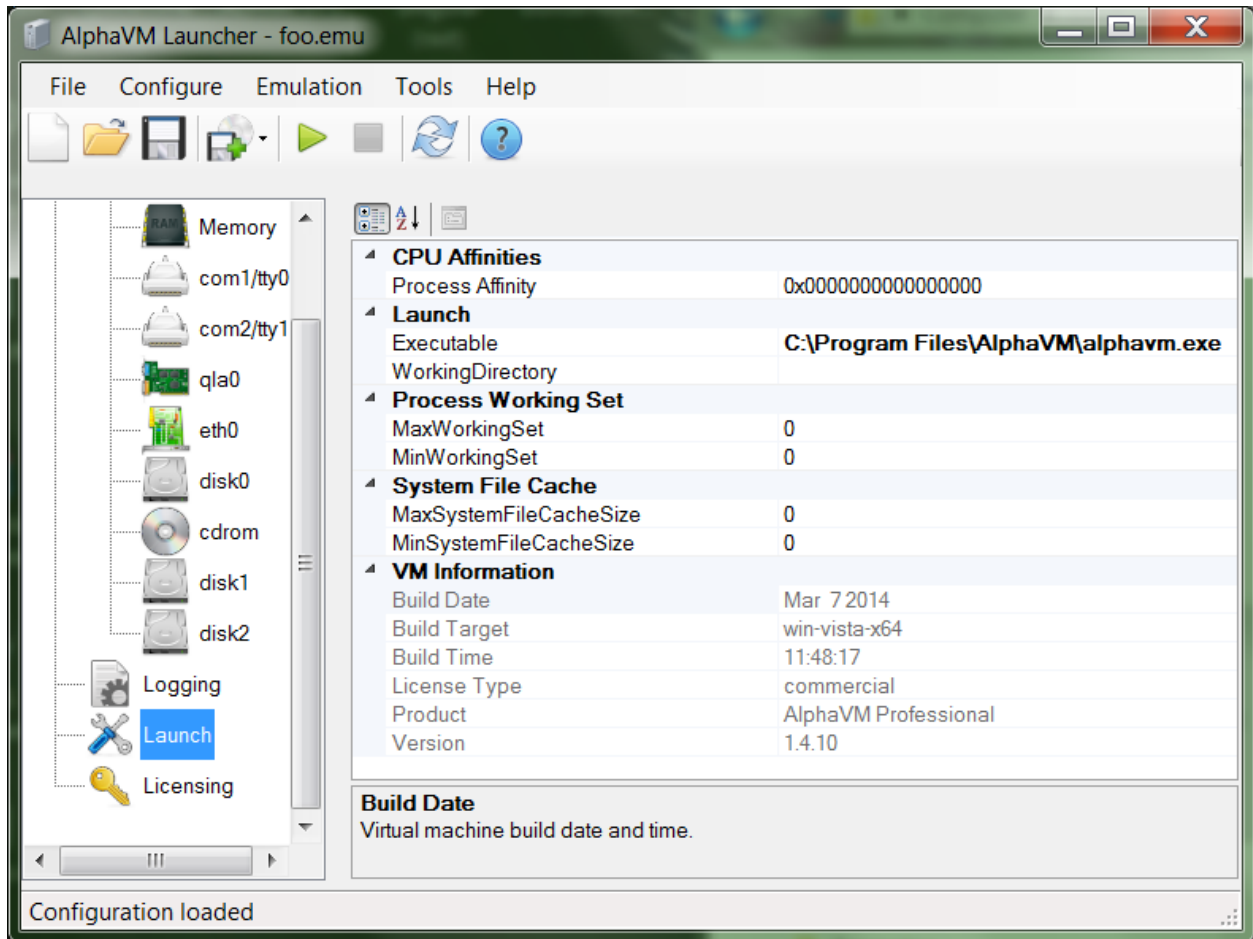


Configuration properties:

- **File** specifies the file where the VM log is written. The default value is “vm.log”. By default, if the path is not provided, the system uses the path of the current configuration file.
- **Append** specifies whether the log file is appended or truncated on every run. The default value is false (truncate every time). Note that in append mode the file can become huge over time. Note also that when you get a problem with the emulator, you should save the log file before restarting of the emulation process, otherwise the log of the erroneous run will be lost.

2.12 VM launching configuration

The Launch configuration pane enables specification of the virtual machine to launch and of its properties.



Configuration properties:

- **Executable** specifies the VM executable. You can change the executable in order to provide an alternative virtual machine, which can be useful when experimenting with different versions. The default value is the main product virtual machine executable, for instance, “C:\Program Files\AlphaVM\alphavm.exe”. For AlphaVM the following virtual machines are provided:
 - The standard AlphaVM virtual machine - alphavm.exe. It is a full featured virtual machine. It requires a dongle for its operation. The functionality is limited by the license you purchased.
 - The free virtual machine – alphavm_free.exe. This virtual machine has the limited functionality and performance. It does not require a dongle for its operation.
- **WorkingDirectory** specifies the working directory for the virtual machine. The working directory determines where files with relative filenames are located. If this property is empty, the path of the configuration file will be used as a default working directory. The default value is empty.
- **Process Affinity** specifies a CPU affinity mask to be used by the VM process. Each CPU in the mask specifies whether the VM can run on the corresponding host CPU. This feature allows limiting the amount of the CPU resources used by the VM. The default value is zero, which means that the VM can run on any available CPU.

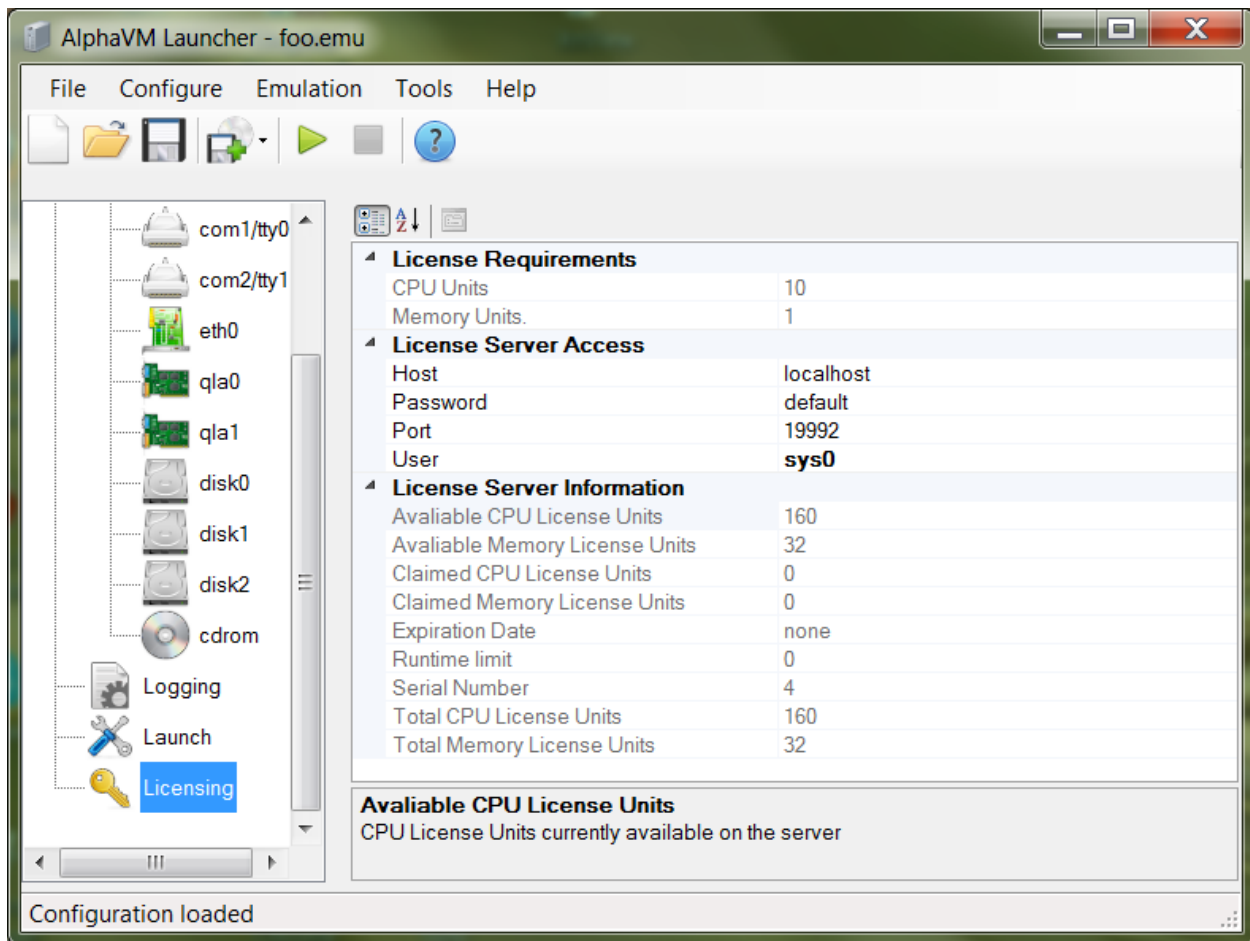
- **MinWorkingSet** and **MaxWorkingSet** specify the minimal and maximal working set limits. These are advanced settings. Do not change them unless you are sure what you are doing. Wrong settings can badly impact the emulator and the system performance. The default value is zero, which means that the virtual machine sets the limits automatically. Working set limits can be changed to tune the VM performance in case the system defaults do not work well. Working set is the amount of physical memory used by the process, in our case the VM. Too low working set limit can cause VM page faults on the emulated memory access, which can disturb timing of the emulated CPU. Too high working set limits can lead to lack of resources for the host system, which degrades the whole system performance including the VM.
- **MinSystemFileCacheSize** and **MaxSystemFileCacheSize** specify the limits for Windows file cache. These settings are system-wide settings. The values are in megabytes. The emulator can adjust them at startup. Note that the settings do not survive the host OS reboot. This is an advanced setting. The default values are zero, which means that the system settings are not changed. The recommended size for the file cache can be around 20% of the host RAM.
- **VM Information** fields show the currently selected virtual machine properties.

2.13 Licensing information

The licensing configuration node contains information related to licensing of the product.

License configuration is required for AlphaVM-Pro. AlphaVM-Free does not require any license configuration. The license configuration section of the configuration file is ignored by AlphaVM-Free.

However, AlphaVM-Free fails with license error when the AlphaVM-Free limits are exceeded. For instance, if you specify more than one CPU or more than 512MB RAM, AlphaVM-Free fails with the license error.



Properties:

- **Host** is the IP address of the system running the EmuVM licensing service. When using a USB dongle, this is normally *localhost*. For evaluation set the evaluation server IP provided by EmuVM.
- **Port** specifies a port number used to connect to the licensing service. Use 19991 with the evaluation license server. Use 19992 with a USB dongle server.
- **User** is a username used to connect to the licensing service. For evaluation, use the user name provided by EmuVM. When using a USB dongle it is usually *sys0* unless another name is provided by EmuVM.
- **Password** is a password used to authenticate the user at the licensing service. When using a USB dongle, please use *default*. Otherwise use the evaluation password provided by EmuVM.
- License requirement properties are read-only. They show the amount of units required by the current configuration of the emulator.
- License Server Information group of properties are read-only. They show the information about the currently connected licensing server. These properties are updated when you switch to the Licensing configuration node in the left panel. If you change the licensing information, please select another node (for instance logging) and then back to Licensing.

2.13.1 Configuring for evaluation

The AlphaVM evaluation can be done using a remote EmuVM server. You will receive the server IP address, port number, username and password to be used. The EmuVM evaluation server uses the port 19991.

Please make sure the outgoing port is open at your firewall and anti-virus software. Please first use ping to check the availability of the server.

2.13.2 Configuring with USB dongle on the local machine

The dongle service is called `keylok_service.exe`. It is available in the `\Program Files\AlphaVM` directory. The AlphaVM installer installs and starts the service. The service appears as `EmuVMLicense` in the Windows Service manager. You do not have to do anything special to start the service. Configure your emulator as follows:

- Host=localhost,
- Port=19992,
- User=sys0,
- Password=default.

2.13.3 Configuring with USB dongle on a remote machine

If you wish to run the service on a remote machine, you can do it in two ways. The simplest is just to install AlphaVM on that machine. The installer will install and start the service. The emulator on that machine will not be unused; it will not use any license units.

Alternatively, you can copy just the service executable `keylok_service.exe` to the machine where you wish to have the dongle plugged. You will have to install and start the service as follows

```
keylok_service --install
```

```
keylok_service --start
```

Service c can be uninstalled as follows:

```
keylok_service --stop
```

```
keylok_service --uninstall
```

In either case configure the Host address of the remote license server to refer to the machine where you have the license server and the dongle.

2.14 Configuration of multiple instances

It is possible to run several instances of the emulator on a single host system.

The following steps are needed to configure several instances

- Make a separate directory for each system configuration.
- Place the configuration file for each system in the corresponding directory.
- Place the private disk and tape images in the corresponding directory.
- Set unique MAC addresses for each network card in each configuration.
- Make sure the serial line configuration use unique ports.
- Set CPU affinities in such a way that different instances use different host CPUs. CPU affinity is a bit mask where each bit represents one host CPU core or hyper-thread. When a bit is 1, the corresponding core is used to run the corresponding instance of AlphaVM. The table below contains the affinity setting examples.
- For AlphaVM-Pro: set the licensing information for each instance. The information about license settings will be provided when you purchase the product.

Examples of setting CPU affinities:

Host cores or hyper-threads	Example of the host system	Number of AlphaVM instances	Max Alpha CPUs	Affinities per instance
4	I5 with hyper-threading on, or I7 with hyper-threading off	2	1	0x0000000000000003 0x000000000000000C
8	I7 with hyper-threading on	2	2	0x000000000000000F 0x00000000000000F0
8	I7 with hyper-threading on	4	1	0x0000000000000003 0x000000000000000C 0x0000000000000030 0x00000000000000C0
6	I7-3970K with hyper-threading off	2	2	0x0000000000000007 0x0000000000000038
6	I7-3970K with hyper-threading off	3	1	0x0000000000000003 0x000000000000000C 0x0000000000000030
12	I7-3970K with hyper-threading on	3	2	0x000000000000000F 0x00000000000000F0 0x00000000000000F0
12	I7-3970K with hyper-threading on	2	3	0x000000000000003F 0x00000000000000FC
12	I7-3970K with hyper-threading on	6	1	0x0000000000000003 0x000000000000000C 0x0000000000000030 0x00000000000000C0 0x0000000000000300 0x0000000000000C00

3 Emulator operation

3.1 Running a VM from the GUI launcher.

3.1.1 Starting the emulation

When the configuration is done, you can start the emulator by means of the Emulation menu or by the toolbar buttons.

3.1.2 Stopping the emulation

Please do not stop the emulator by means of the user interface stop button unless it is really necessary. This corresponds to an abnormal system power failure and can cause troubles with the guest operating system or other guest software currently running in the emulator. Instead, shutdown the guest system and use SRM **power** command to "power down" the emulated system.

3.2 Running a VM from command line

Sometimes it is more convenient to run the virtual machine from the command line than from the EmuLaunch user interface. On Linux there is no launcher yet, so the only option is to run from the command line.

The launcher not only starts the VM itself, but also the terminal emulators, if configured. When running the VM from command line, the user has to connect the terminal emulators manually.

The VM is started as follows:

```
alphavm.exe <config-file>
```

Here it is assumed that alphavm.exe is either in PATH or you cd to the AlphaVM directory. Here is a full example of the command line:

```
"C:\Program Files\AlphaVM\alphavm.exe" c:\Test\foo.emu > foo.log 2>&1
```

Normally the configuration file is created by the launcher. When running VM directly, the user has to write the configuration file. Alternatively, one can modify a file created by the launcher.

To see an example of a configuration file use Tools menu/ View configuration as text.

3.3 Running a VM as a service

AlphaVM-Pro can be run in a context of a Windows service. There is a special service – EmuVMSrv. It is installed by the installer to manage the VMs. Multiple instances can be started as a service.

There is a special EmuVMSrv configuration file “\Program Files\AlphaVM\emuvmsrv.cfg”. Each line in this file specifies the full path to a VM configuration to be run. Usually it is a file that just contains one line. For example:

```
C:\Test\foo.emu
```

A line can be empty. A line starting with the # character is ignored (i.e. it is a comment).

A VM configuration can be created as usually by the GUI.

Initially emuvmsrv.cfg is empty. If it is empty, the EmuVMSrv service stops. When the emuvmsrv.cfg is changed, restart the service to take the effect. Currently the service does not automatically rescan the configuration file.

The service log is stored in “\Windows\system32\EmuVMSrv.log”.

When you are running multiple VMs on a single Windows host, you have to consider that the instances may have a resource conflict, if you do not take care of it. See [Configuration of multiple instances](#) for the details.

Note that all instances running in a service and all instances running interactively have to be considered when thinking multiple instances. In particular note that an instance, that runs automatically, should not be run interactively at the same time.

The VM configuration Launch section contains a parameter called **Restart**. The EmuVMSrv service uses it to check what to do if the service stops.

EmuVMSrv does not start the terminal emulators that are normally started by the GUI launcher. You have to connect the terminal emulators manually.

4 Usage example

4.1 OS installation on a new disk

This is a general sequence of installing of an OpenVMS, Tru64 or Linux on the emulated system.

Step 1: Create an empty disk by means of Tools/Make disk

Step 2: Configure the *disk0* to use the just created empty disk image.

Step 3: Configure the *cdrom* to map to your ISO file. Alternatively, use [\\.\Cdrom](#) instead of the disk image to use the real CD .

Step 4: Save configuration to a file, say vms83.emu.

Step 5: boot from *cdrom*: **boot dka400**

Step 6: Follow the OS installer sequence as usually. The target disk for the installation is DKA0.

Step 7: When the installation is completed, you can boot from the new system: **boot dka0**.

5 Migration

A real system can be replaced by the emulator software. Firstly, the emulator should be configured to reflect the real system as close as possible. Secondly the software should be transferred to the emulator.

5.1 Migration by copying disks

The simplest way of migration is by copying the real system disks to disk images and then using these disk images to run the emulation. Thus, the whole OS, software and data are copied. The new system behaves in the same way as the old one.

Unfortunately, this method does not always work. At the moment we cannot emulate all kinds of Alpha systems and all kinds of peripheral devices. Some OSs and applications are flexible and can run on a different hardware configuration without changes. Others require more or less complicated reconfiguration.

5.1.1 Copying disks using a Live Linux CD

When a system is booted from a disk, this disk cannot be modified while being copied. To avoid such problems you can boot a Linux system from so called Live CD (for instance, Gentoo LiveCD). In this case you get a booted Linux system that does not use any of your OpenVMS or Tru64 disks. This Linux system can be used to copy your disks by means of the `dd` command. You will have to use a storage on the network to store the resulting disk images.

When copying disks please use the whole disk devices like `/dev/sda`, rather than partition devices (like `/dev/sda1`)

Please note that Tru64 can be very sensitive to configuration changes. When the copied disk image is booted in the emulator it may fail due to difference in the configuration. In this case you can boot the generic kernel and reconfigure/rebuild your kernel as usually.

See a detailed example on emuvms.com/migrate_vms.php.

5.1.2 Migration of OpenVMS using backup /image

The usual way to migrate OpenVMS is to use backup /image copies of the disks. The following command can be used to make such backups:

```
$ mount dka100: /over=id  
  
$ backup /image/verify dka100: dka500:[000000]foo.sav/sav
```

You can use the command to backup your system disk.

The following commands can be used to restore disk images:

```
mount dka100: /foreign  
  
backup /image/verify dka500:[000000]foo.sav/sav dka100:
```

The migration process can be outlined as follows:

1. Make backup /image backups of all disks in your system and make them available via the network.
2. Install AlphaVM and create N+1 empty disk images. Add the disk images to the emulated system. One disk will be used for a freshly installed copy of OpenVMS. The other disks will be used to restore your original disks. The AlphaVM disk images can be larger than the original disks.
3. Install a fresh copy of OpenVMS on AlphaVM and configure the network to access the disk backups.
4. Restore the disk backups from the files to the empty emulated disks.
5. Boot from the restored system disk.

It is advised that your fresh system disk is made large enough to contain any of the *.sav files (or all of them). In this case you can restore from a local copy rather than from a DECNET remote copy. It is possible to use TCPIP to ftp files and you do not need to configure DECNET.

5.1.3 Copying disks on OpenVMS

On OpenVMS a proper disk image can be created by means of backup /physical. Please note that the physical image has the same size as the original disk. Therefore you need to have enough storage to store the resulting file. Please note that you can use a network path for the destination file.

5.1.4 Copying disks on Tru64

On Tru64 a proper disk image can be created by means of the dd command. Please note that the physical image has the same size as the original disk. Therefore, you need to have enough storage to store the resulting file. Please note that you can use a storage available via the network as the result, for instance NFS.

5.2 Migration by reinstalling software from scratch

When it is impossible or inconvenient to copy disks, the software can be installed on the emulator in the usual way. These are the steps to be done:

- Install the OS and its layered products.
- Install and configure the application software.

You may need to copy data from the old system.