



# **Intel® Simics® Simulator New User Training**

## **Lab 001 – Simulator basics**



# Lab 001 – Intel® Simics® Simulator basics

In this lab, you will get familiar how to create projects and how to run Intel® Simics® simulation sessions.

## A. Conventions

The following conventions are used in the lab instruction:

Actions from the shell on the host (the machine that runs Simics) are indicated by **\$** or **C:\...>**. The commands to type are in **bold**:

```
$ ls
```

Note that the Intel Simics simulator core allows the use of Linux-style paths on Windows hosts. Thus, in the cases where you refer to file paths when starting a new simulator session, you can use Linux-style script names that you copy from the instructions even on Windows host. For example, this is a valid invocation of the simulator:

```
C:\...> simics.bat targets/vacuum/vacuum.simics
```

Tab completion in Windows CMD will produce \-separated paths, and these also work. Tab-completing paths from the simulator command line will also produce native Windows paths.

Commands entered on the Intel Simics simulator command line are indicated by **simics>**:

```
simics> help
```

When the simulator is running, the prompt changes to **running>**. Most command-line commands can be used while the simulator is running:

```
running> list-processors
```

Actions from the shell on the target (the simulated machine) are also indicated by **\$** or **#**. The instructions will indicate that this is to be entered on the target system. The target system prompt is only used while the simulation is running, so it should be fairly clear when the target console is used and when the host shell is used.

```
# lspci
```

Output from commands (in any environment) are shown as a box with slightly smaller text, and in regular font. Typically, the command used to generate the output is not shown in the box (but it is sometimes included for clarity). For example:

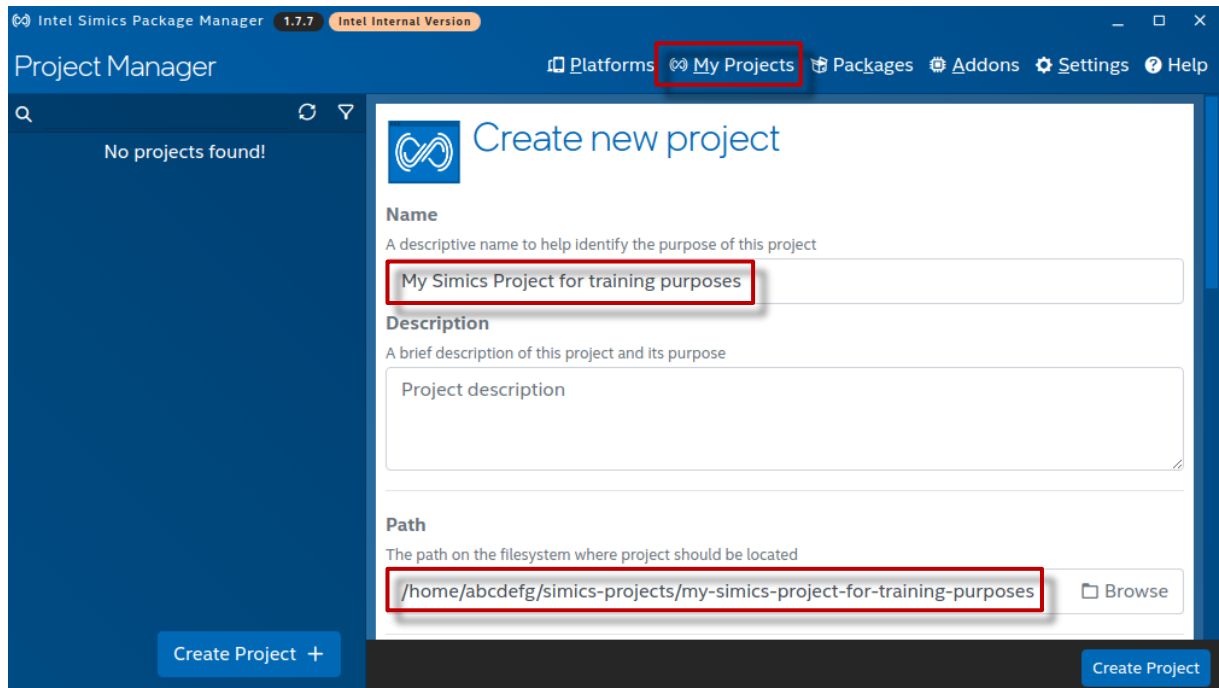
```
Status of sim [class sim]
=====

Environment:
  Hide Console Windows : No

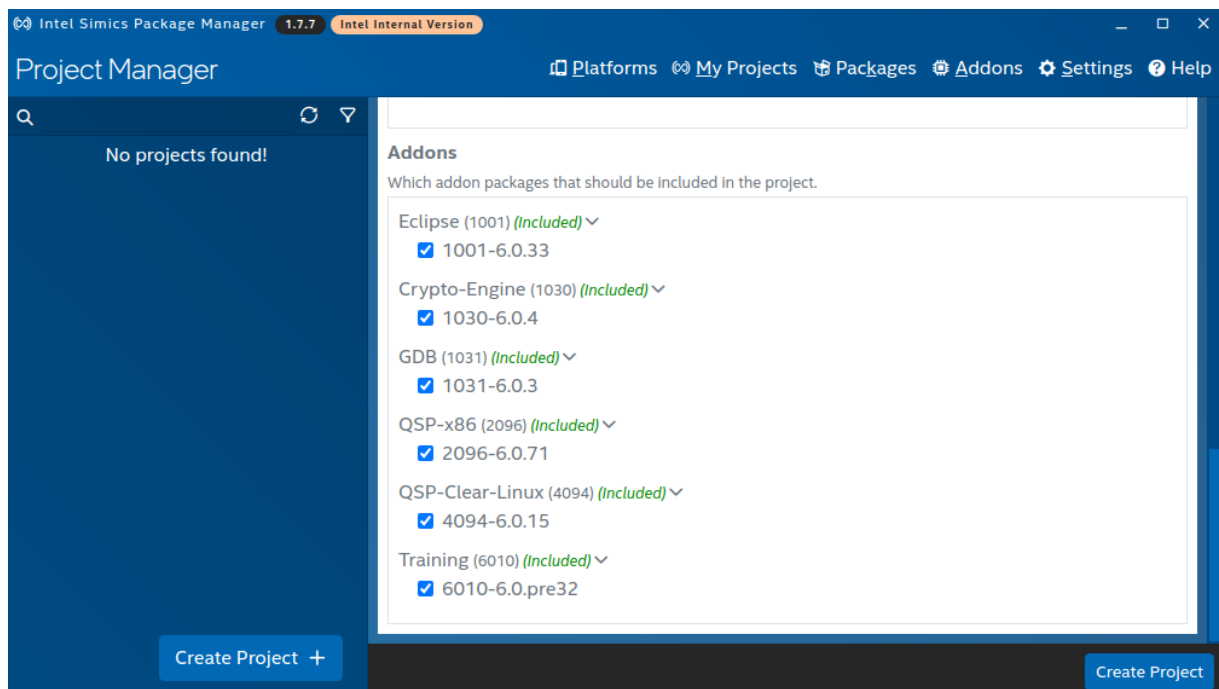
Simulation Engine:
  Page Sharing : Disabled
  Multithreading enabled : Enabled
  Thread limit : Unlimited
  Worker threads limit : 1
  Simulation threads limit : 0
  CPU module load mode : normal
  Image memory usage : Limited to 22.24 GB
  Image memory limit hit : 0 times
...
```

## B. Set up the project

1. We assume you followed the Installation Guide and hence know how to start the Intel® Simics® Package Manager (ISPM). Start ISPM again, select **My Projects** and in the shown **Create new project** dialog, set the project description, and select a directory in which the project shall be created.



2. Scroll to the bottom of the **Create new project dialog** and select the packages as shown (exact versions can differ) then click on **Create Project**.



- When asked if the project directory shall be created, select Yes.

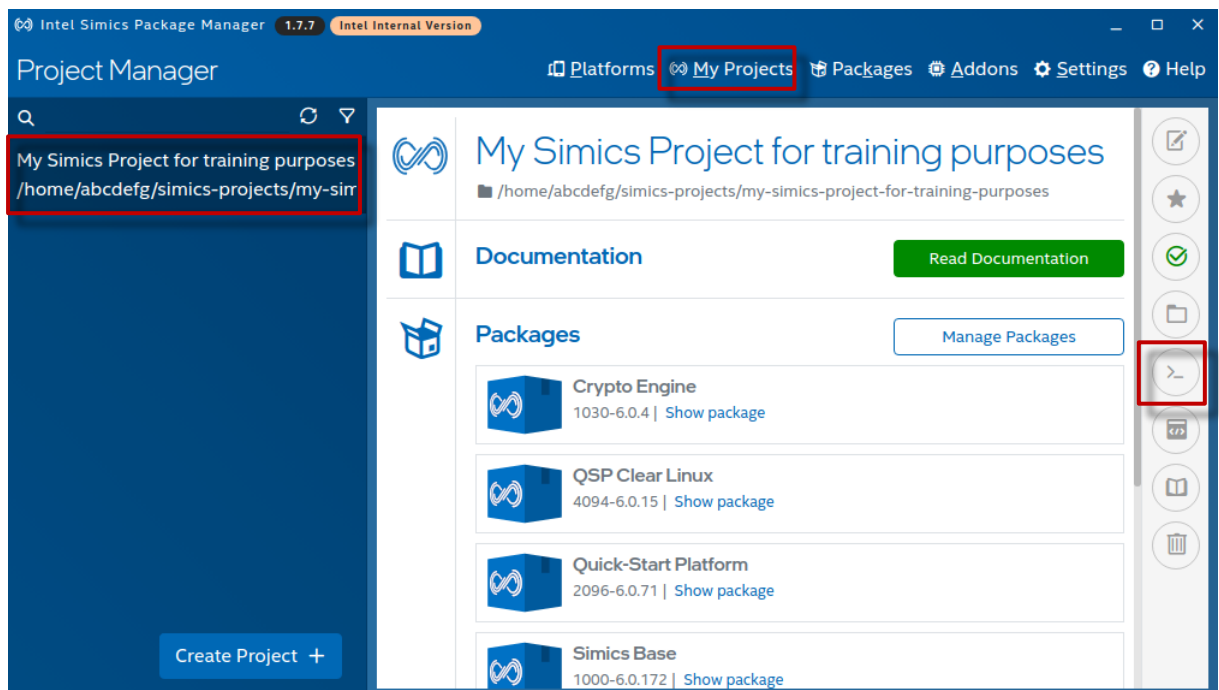
### Create Project

Project will be created at "/home/abcdefg/simics-projects/my-simics-project-for-training-purposes" which does not exist. Do you want to create it?



### Open a terminal in the project directory

- Start the Intel Simics Package Manager again (in case you closed it earlier).
- Select **My Projects**, then select your training project, and then click on the "Open in terminal" button.



NOTE: You can also manually open a terminal application and navigate to the project, but going through ISPM works the same way on all supported hosts.

### Download the Linux Image

For the training to work smoothly, we recommend having the required Linux image in your project. If you have downloaded the image previously, you can just copy it into your project. If not, below are the steps you need to perform to allow the Simics scripts to find the image.

- Create the directory **targets/qsp-x86/images** in your project.

- Download <https://downloads.yoctoproject.org/releases/yocto/yocto-5.1/machines/genericx86-64/core-image-sato-sdk-genericx86-64.rootfs-20240917113829.wic> into the directory you created.

Afterwards, you should have the file

**<project>/targets/qsp-x86/images/core-image-sato-sdk-genericx86-64.rootfs-20240917113829.wic.**

## Check the simulator version and available targets

- In the terminal, launch a new empty simulation session, as shown below. This will be used to check that the simulation is correctly set up for running the training.

In a Windows\* CMD environment:

```
C:\...> simics.bat
```

In a Windows PowerShell environment

```
PS C:\...> .\simics.bat
```

Note that in the rest of the labs, if you are using PowerShell instead of CMD, anytime you see “**simics.bat**”, please translate that to “**.\simics.bat**”.

In a Linux\* environment:

```
$ ./simics
```

The Intel Simics Simulator will start, and the terminal will switch to the simulator command line interface (CLI), which you can tell by seeing the **simics>** prompt.

- Check that all the required packages are activated in the project:

```
simics> version
```

For the labs to function, you should at least see the following packages:

...			
Installed Packages:			
Pkg	Name	Version	Build ID
1000	Simics-Base	...	...
1030	Crypto-Engine	...	...
1031	GDB	...	...
2096	QSP-x86	...	...
4094	QSP-Clear-Linux	...	...
6010	Training	...	...
...			

Note that the command also indicates the type of host the Intel Simics Simulator is running on, as well as whether the VMP acceleration technology for X86-on-X86 is installed.

- There are two ways to launch a target in the Intel Simics Simulator.

You can launch targets that were registered by packages and found in the project based on their file names (files ending in **.target.yml**).



Alternatively, you can run scripts that you find in the **targets/** directory of your Intel Simics project, named like **targets/targetname/targetvariant.simics**. This is the traditional way of representing target systems, going back to the very start of the Intel Simics Simulator. Using registered targets in the new way. Currently, you might see both ways still being used, with the old way being phased out over time.

List the targets that are found in the Intel Simics Packages associated with your current project:

```
simics> list-targets
```

This should show a list containing the Intel Simics Quick-Start Platform and the training package. There might be more targets from other packages.

Target	Package
qsp-x86/clear-linux	Quick-Start Platform
qsp-x86/clear-linux-2c	Quick-Start Platform
qsp-x86/clear-linux-multi	Quick-Start Platform
qsp-x86/uefi-shell	Quick-Start Platform
qsp-x86/user-provided-linux	Quick-Start Platform
simics-user-training/001-qsp-training	Simics Training
simics-user-training/007-multimachine-network	Simics Training

- For more details on the targets, including a description of what they contain and where they are located on disk, use **-verbose**:

```
simics> list-targets -verbose
```

- For more details on the packages you use the **list-packages** command:

```
simics> list-packages
```

- Quit this Simics session to get back out to the host terminal.

```
simics> quit
```

## C. Start a new simulation session

### Start the training setup virtual platform

1. In the terminal, launch a simulation session as shown below. By giving a Simics script, target name, or checkpoint as an argument to the **simics** program, you ask the simulator to use that as the initial setup. Here, we use a target from the list you produced in the previous step.

In a Windows CMD environment:

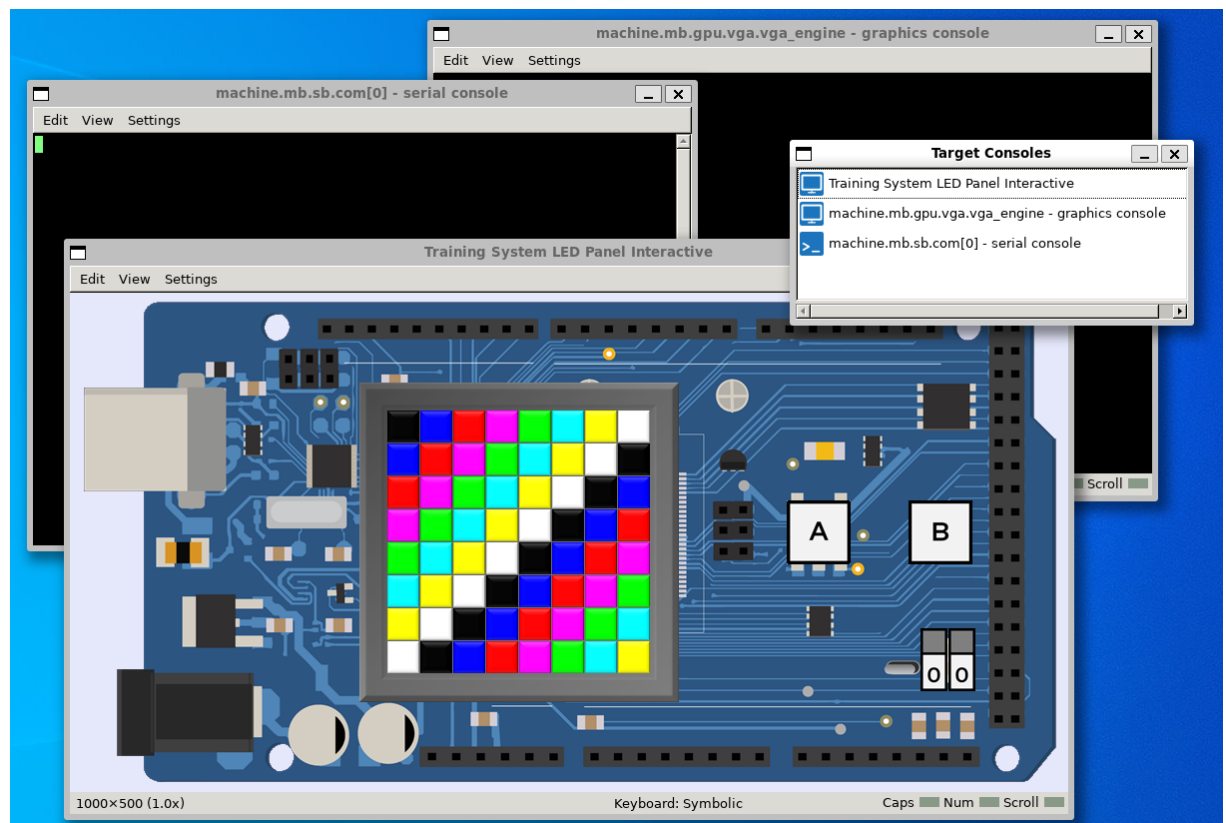
```
C:\...> simics.bat simics-user-training/001-qsp-training
```

In a Linux environment:

```
$ ./simics simics-user-training/001-qsp-training
```

2. Note that you can ignore any warnings about VMP not being enabled; it is not important for the New User Training labs. It does speed up execution though, so it is worth enabling.
3. Four new windows will pop up. One for the serial console, one for the graphics console and one for the LED Panel of the target system, as well as a target console control window that lets you show and hide individual consoles. The serial and graphical consoles are blank since we have not yet started the simulation. The LED Panel displays a stylized representation of the simulated board with its LED matrix, two dip switches and two buttons A and B.

The set of windows will look something like this:



4. Run the simulation, using the command line:

```
simics> r
```

The state of the session will be shown as “Running”, as you can see in the prompt in the command line changing to **running>**.

5. The graphics console will change size when the UEFI takes over, and soon you should see the UEFI splash screen for the Intel Simics Quick Start Platform (QSP):



6. Depending on your host computer, you may have to wait a while for the boot to finish and the Linux command line to come up on the serial console.
7. While it is booting, check out the target information to get an overview of the system that is booting:

```
running> print-target-info
```

8. List the processors in the system:

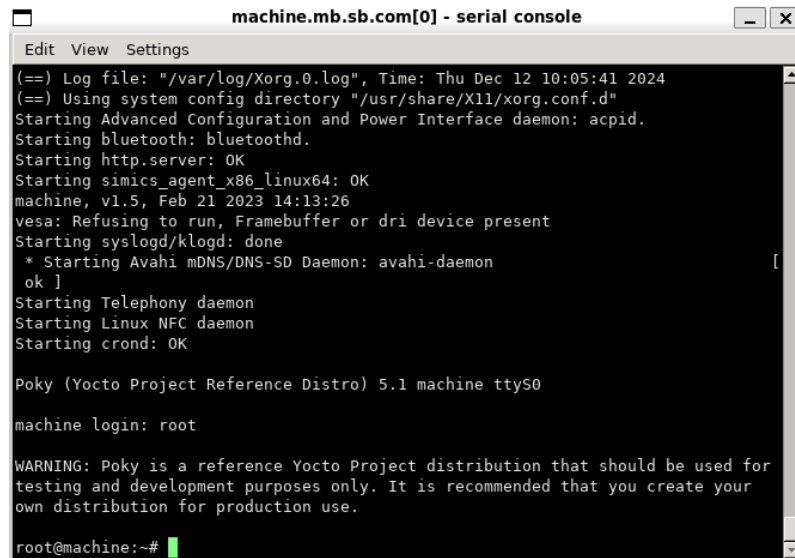
```
running> list-processors -time -disassemble
```

9. If the boot takes a long time on your machine, you can issue the **list-processors** command a few times to see how time is progressing. The boot should be done after 20 target seconds at most.

10. After the target system has booted, there should see a message on the Intel Simics Simulator command line stating:

```
Autologin as "root" was done on "machine.mb.sb.com[0] - serial console".
```

This is a note from an automation script that makes sure that you are logged into the system in the **Serial Console** window:



```
machine.mb.sb.com[0] - serial console
Edit View Settings
(==) Log file: "/var/log/Xorg.0.log", Time: Thu Dec 12 10:05:41 2024
(==) Using system config directory "/usr/share/X11/xorg.conf.d"
Starting Advanced Configuration and Power Interface daemon: acpid.
Starting bluetooth: bluetoothd.
Starting http.server: OK
Starting simics_agent_x86_linux64: OK
machine, v1.5, Feb 21 2023 14:13:26
vesa: Refusing to run, Framebuffer or dri device present
Starting syslogd/klogd: done
* Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon
ok ]
Starting Telephony daemon
Starting Linux NFC daemon
Starting crond: OK

Poky (Yocto Project Reference Distro) 5.1 machine ttyS0

machine login: root

WARNING: Poky is a reference Yocto Project distribution that should be used for
testing and development purposes only. It is recommended that you create your
own distribution for production use.

root@machine:~#
```

The reason the script logs you in as root is that this simplifies the Linux command-line operations dealing with the device driver for the system panel. It is admittedly not the best practice, but for a simulated system that is entirely self-contained in a simulator like this, it is fine.

You can also log out of the system and then again in to the target system on the by using the user name "**root**", or, for a standard user, use user name "**simics**". Note that a password should not be needed.

11. Try some commands in the Linux environment in a target console where you are logged in. For example:

```
# cat /proc/cpuinfo
# ifconfig
# ls
```

12. Once you have explored the target a bit, quit the simulation by giving the "**quit**" command on the command-line.

```
running> quit
```

## D. Set target parameters and save a checkpoint

In this part, we will look at how to configure a new simulation using target parameters and how to save a checkpoint to use in later exercises.

1. To change the parameters for the system setup, you pass them as command-line arguments, after specifying the target you are launching. To see the available parameters for a particular target, run the below command from shell. Note that the parameters can also be listed from inside a running simulator, with more inspection power.

In a Windows CMD environment:

```
C:\...> simics.bat simics-user-training/001-qsp-training --help
```

In a Linux environment:

```
$ ./simics simics-user-training/001-qsp-training --help
```

You will see many parameters that can be changed. Look for **memory\_megs** parameter as shown below:

Name	Type	Description	Default
[...]			
platform:machine: hardware:memory_megs	int	Amount of RAM in the machine, in MiB. The highest supported value is 65536 (64 GiB). More than 64 GiB is not supported since the x86QSP1 processor is limited to a 36-bit physical address space.	8192
[...]			

If the output on the host shell is hard to read, you can also try starting an empty simulation session and run the command **params.help target="simics-user-training/001-qsp-training"**.

Note that by default, only the most important parameters are shown. The developer of a target can assign a certain “advanced” level to a parameter in order to only show the most common parameters by default. To see all parameters, you need to use the **params.help** command inside a simulation run.

- For purpose of illustration, we want to change the **memory\_megs** parameter from **8Ki** to **16Ki**, hence launch the session as shown below. The “Ki” is a shortcut available for changing parameters from the command line.

In a Windows CMD environment:

```
C:\...> simics.bat simics-user-training/001-qsp-training  
platform:machine:hardware:memory_megs=16Ki
```

In a Linux environment:

```
$ ./simics simics-user-training/001-qsp-training  
platform:machine:hardware:memory_megs=16Ki
```

- When the new session has launched, check the target information in the control window to see that the target system is now configured with 16GiB of RAM instead of 8GiB.

```
simics> print-target-info
```

The output should look like this:

QSP x86 with Linux - Simics Training Setup

System	a QSP x86 chassis
Processors	2 QSP X86-64, 2000.0 MHz
Memory	16 GiB
Ethernet	1 of 1 connected
Storage	2 disks (208 GiB)

- Run the simulation to boot the target, just like before.

```
simics> r
```

## Save a checkpoint

- Wait for the target system to boot and automatically log in, like you did before.
- Stop the simulation once the target system has booted and been automatically logged in on the **Serial Console**. The simulation has to be stopped in order to save a checkpoint:

```
running> stop
```

- To save a checkpoint, use the **write-configuration** command. Provide a name for the checkpoint and a comment describing what it contains:

```
simics> write-configuration MyFirstCheckpoint.ckpt "Linux booted to  
prompt"
```

The checkpoint save might take a short while, since it will save the current state of target memory and any changes to the disk.

- Check the current state and time of the processors, to have something to compare to when opening the checkpoint:

```
simics> list-processors -time -cycles -disassemble
```

Take a note of what the state looks like.

9. Quit this simulation session:

```
simics> quit
```

## Open the checkpoint

10. To start from a checkpoint, simply pass it as the sole argument when launching a session.

In a Windows CMD environment:

```
C:\...> simics.bat MyFirstCheckpoint.ckpt
```

In a Linux environment:

```
$ ./simics MyFirstCheckpoint.ckpt
```

11. Check the current state and time of the processors, and compare to the output from the session where the checkpoint was saved:

```
simics> list-processors -time -cycles -disassemble
```

The shell output might look like this. Note how the current state for the processors is the same after opening the checkpoint:

```
## End of previous simulator session
simics> write-configuration MyFirstCheckpoint.ckpt "Linux booted to prompt"
simics> list-processors -time -cycles -disassemble
```

CPU Name	CPU Class	Freq	Cycles	Time (s)	Disassembly
machine.mb.cpu0.core[0][0]	* x86QSP1	2.00 GHz	29_343_888_717	14.67	cs:0xfffffffff81002077 p:0x001002077 test bl,0x8
machine.mb.cpu0.core[1][0]	x86QSP1	2.00 GHz	29_343_800_000	14.67	cs:0xfffffffff81d7194a p:0x001d7194a mov rax,qword ptr gs:[0x14dc0] # MWait state

```
* = selected CPU

simics> quit

## Starting a new session from the checkpoint:
PS C:\Users\...> .\simics.bat .\MyFirstCheckpoint.ckpt\
Intel Simics 6 (build 6243 win64) © 2023 Intel Corporation

Use of this software is subject to appropriate license.
Type 'copyright' for details on copyright and 'help' for on-line documentation.

simics> list-processors -time -cycles -disassemble
```

CPU Name	CPU Class	Freq	Cycles	Time (s)	Disassembly
machine.mb.cpu0.core[0][0]	* x86QSP1	2.00 GHz	29_343_888_717	14.67	cs:0xfffffffff81002077 p:0x001002077 test bl,0x8
machine.mb.cpu0.core[1][0]	x86QSP1	2.00 GHz	29_343_800_000	14.67	cs:0xfffffffff81d7194a p:0x001d7194a mov rax,qword ptr gs:[0x14dc0] # MWait state

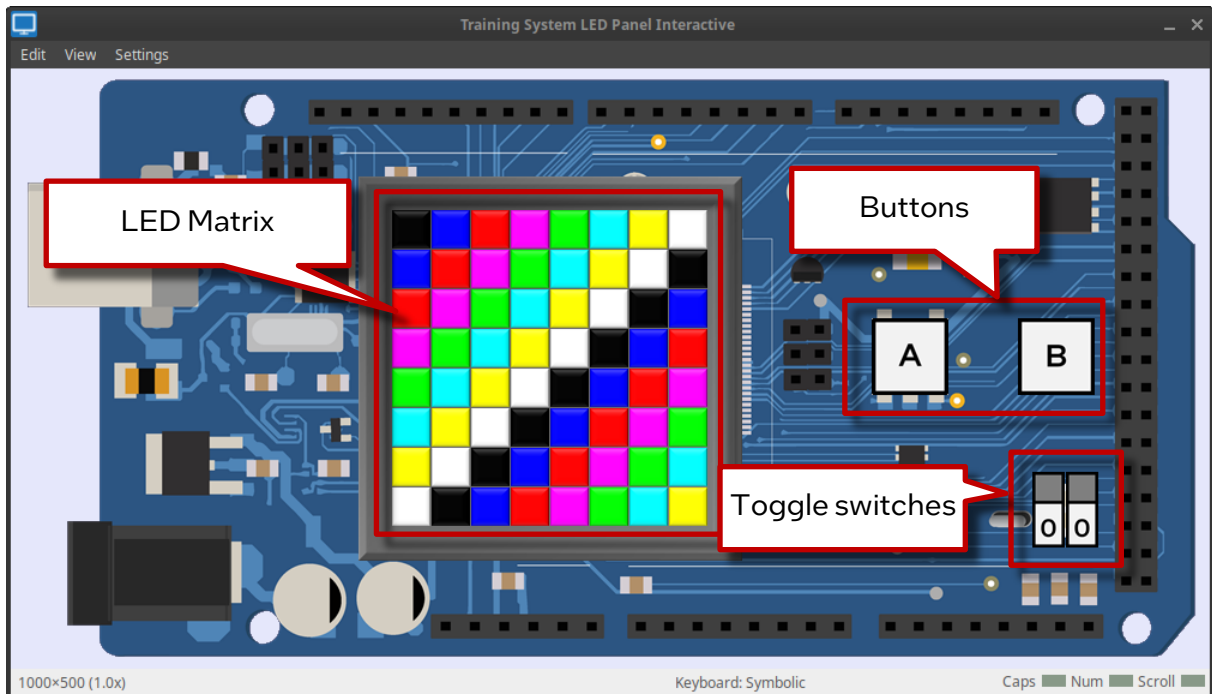
```
* = selected CPU

simics>
```

12. Note that all the windows are opened, and that the serial console shows the same logged-in state as when the checkpoint was saved.
13. Do not quit the simulator! We will continue working in this Simics session in the next lab section.

## E. Activate the software for the training panel

The training system uses an LED panel device for input and output during several labs. The panel is a front end connected to a set of hardware models that run in the virtual platform – every LED, button, and toggle in the panel has a device model behind it.



The device models for the input and output devices are connected to a controller device over I2C. The controller card is in turn connected over PCIe to the main Quick-Start Platform system. From a Linux perspective, the training card is a PCIe device, and Linux needs a custom driver in order to do anything with the card. Thus, to activate the device, you have to load the appropriate driver into the Linux kernel on the target system.

14. Run the simulation again so that the target becomes interactive.

```
simics> r
```

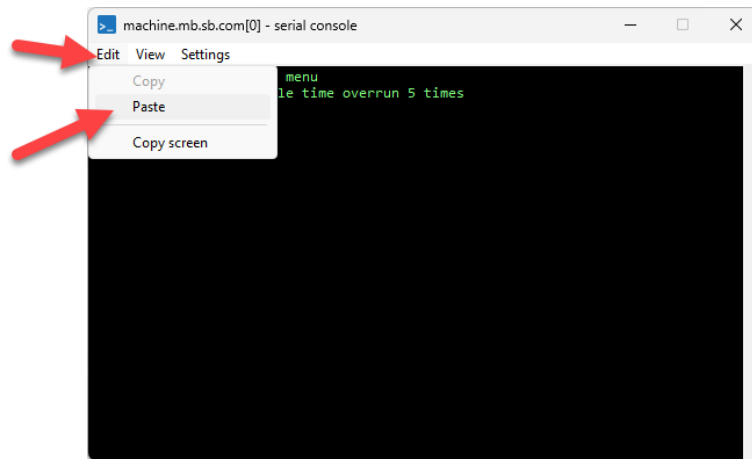
15. Go to the **Serial Console** window and enter the following command on the Linux command-line console (you can tab-complete after writing the first few letters of the name of the driver):

```
# insmod simics-training-pcie-driver.ko
```

Note that we show “#” to indicate the start of the Linux prompt for root in order to make it clear where the command is to be entered.

If you copy the above string, you can paste it into the serial console using the **Edit > Paste** command:





16. To check the result of the **insmod** operation, you need to check the kernel debug messages log:

```
# dmesg | tail -7
```

The result should look something like this:

```
[ 23.410137] simics_training_pcie_driver: Access to BAR0 ready, mapped at
0x000000004e063765.
[ 23.410153] simics_training_pcie_driver: Access to BAR3 ready, mapped at
0x00000000d33bd788.
[ 23.410466] simics_training_pcie_driver: MSI-X interrupts enabled (pci_enable_msix,
for 2 interrupts).
[ 23.410486] simics_training_pcie_driver: request_irq successful for interrupt 0,
vector=34
[ 23.410499] simics_training_pcie_driver: PCIe probe function returned successfully!
[ 23.410553] simics_training_pcie_driver: PCI device driver registration successful
[ 23.410566] simics_training_pcie_driver: Device driver initialization successful
```

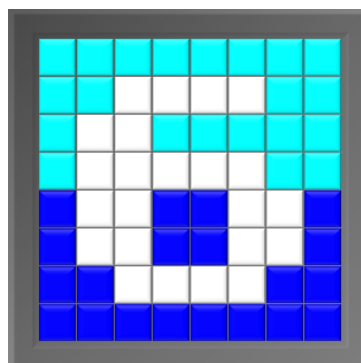
17. The driver should have established a **"/dev"** node in the Linux file system. Check the list of devices in the target system, and locate one called **simics\_training\_pcie\_driver**:

```
# ls /dev
```

18. To change the picture shown in the LED display, you can send a string of byte values to the device. There are a few ready-made patterns already loaded in text files on the training disk image. For example:

```
# cat six.txt > /dev/simics_training_pcie_driver
```

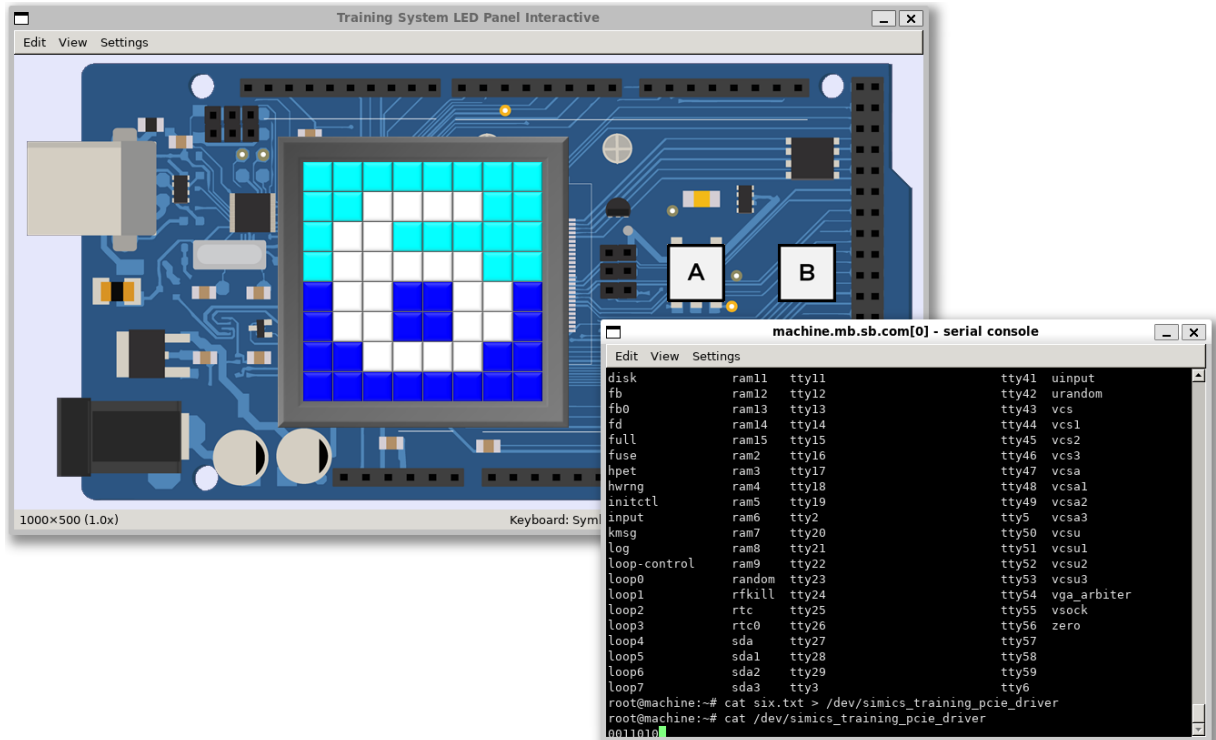
Which should result in a display like this:



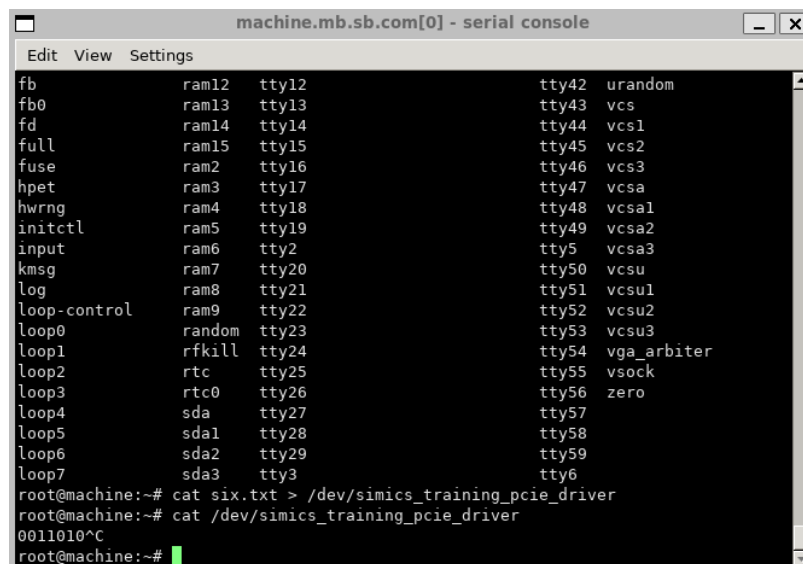
19. You can also observe button presses in the System Panel from the Linux shell, by reading from the device node:

```
# cat /dev/simics_training_pcie_driver
```

20. Click the buttons (labelled **A** and **B**) in the panel, and Linux should print a string of **0** and **1** to show each button press.



21. Use **Control-C** in the target serial console to break out of the **cat** operation and get back to prompt.



## Save another checkpoint

22. Stop the simulation:

```
simics> stop
```

23. Note that the graphics console showing the panel might be greyed out. This is the Intel Simics Simulator default behavior for graphics consoles, to indicate that input is not possible when the simulator is stopped. If you want to get the colors back, use the **dimming** command. Note that this dimming setting is not saved in a checkpoint:

```
simics> machine.training_card.panel.con.dimming FALSE
```

24. Save a checkpoint called "AfterDriver.ckpt":

```
simics> write-configuration AfterDriver.ckpt "Driver loaded and tested"
```

25. In the command line, list the checkpoints you have created so far.

```
simics> list-checkpoints
```

The output should show the two checkpoints and their comments

```
AfterDriver.ckpt
  Driver loaded and tested
MyFirstCheckpoint.ckpt
  Linux booted to prompt
```

You will be using these two checkpoints as a starting point in later labs.

26. Quit the simulation session.

```
simics> quit
```

## F. Explore the Intel Simics Simulator documentation

The Intel Simics Simulator has an extensive help system comprising both structured manuals and online help for commands and interfaces. To access the documentation (usage manuals), use the “documentation” script in your project. In the next lab, you will use the online help system from the command line.

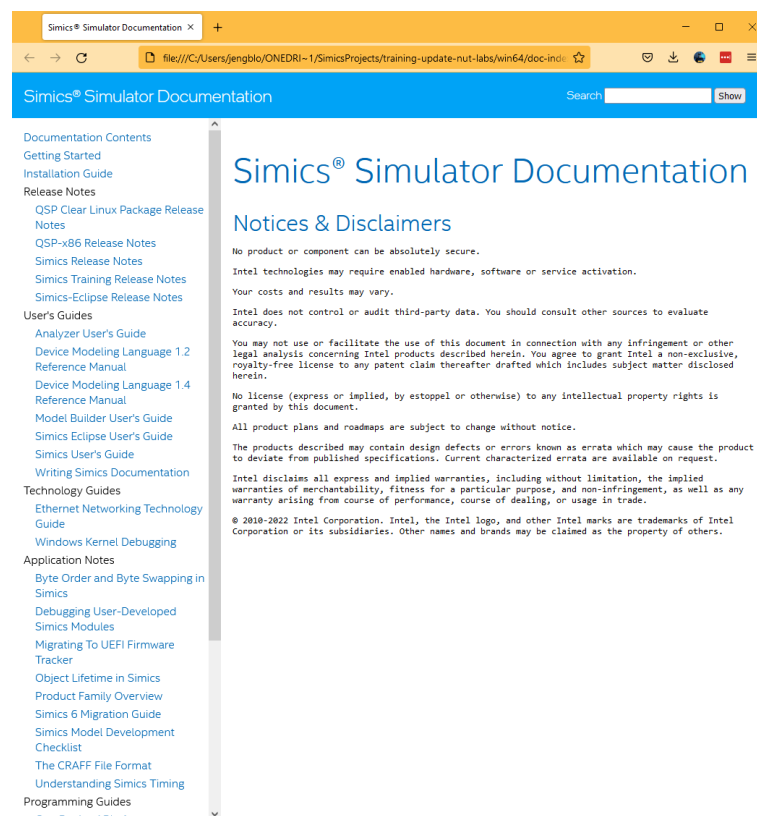
1. Open a new host terminal or CMD in your project, or use the terminal you already have open.
2. Run the documentation script.
3. In a Windows CMD environment:

```
C:\...> documentation.bat
```

In a Linux environment:

```
$ ./documentation
```

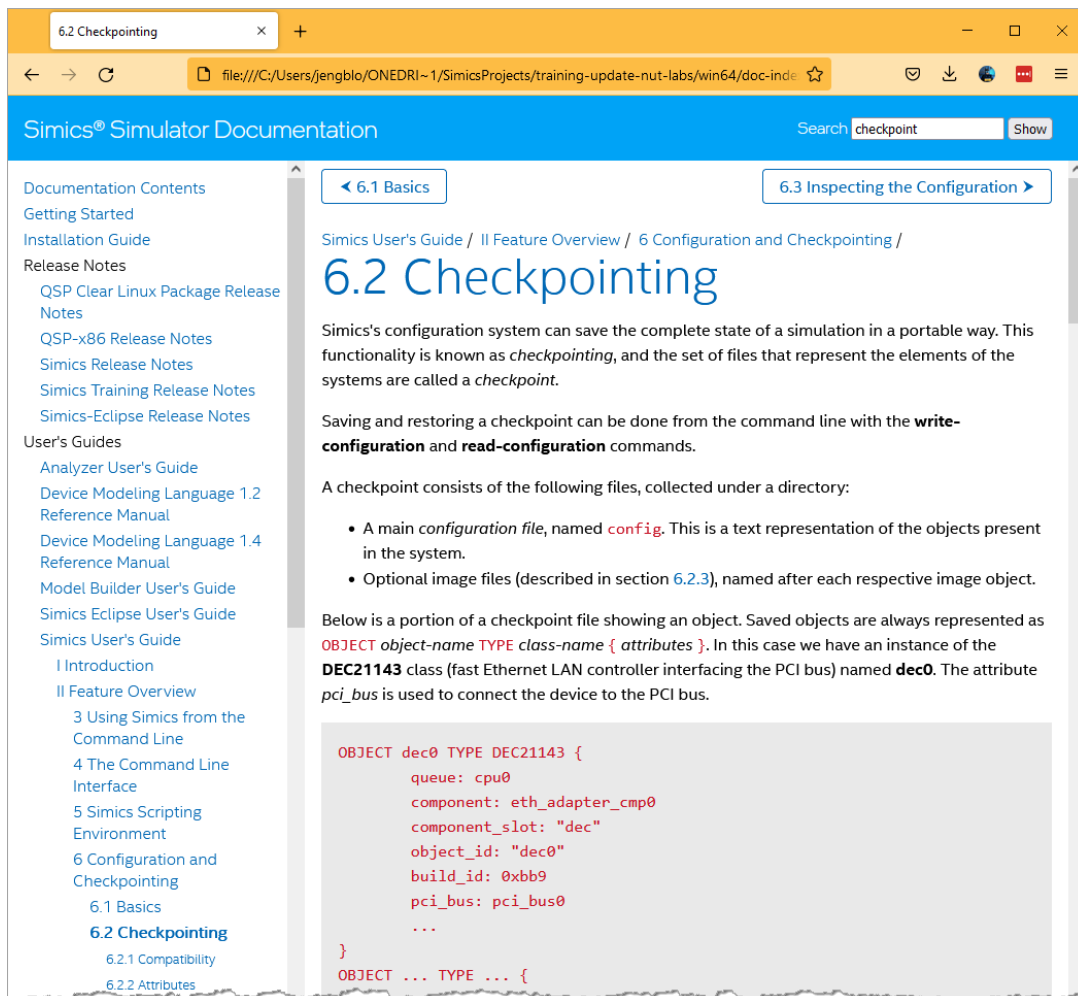
4. The documentation will build its index, and then open in a browser window:



- To search the documentation, type the string into the **Search** box. For example, search for “**checkpoint**”. The precise hits presented will vary between simulator releases, as the documentation is updated.



6. Click on a search result to jump to that part of the documentation. For example:



- To look for information in just a single document: Once you have a page from a certain document open, check the **Only Search in...** check box in the search pane:

**Simics® Simulator Documentation**

[Search](#)
[checkpoint](#)
[Hide](#)

---

- » Documentation Contents
- » Getting Started
- » Installation Guide
  
- RELEASE NOTES**
- » Crypto Engine Release Notes
- » QSP Clear Linux Package Release Notes
- » Simics GDB Platform Notes
- » Simics Release Notes
- » Simics-Eclipse Release Notes
  
- USER'S GUIDES**
- » Analyzer User's Guide
- » Creating Simics packages
- » Device Modeling Language 1.2 Reference Manual
- » Device Modeling Language 1.4 Reference Manual
- » Model Builder User's Guide
- » Python Coroutines in Simics
- » Simics Eclipse User's Guide
- » Simics User's Guide**
- » Introduction
- » II Feature Overview**
- 3 Using Simics from the Command Line
- » 4 The Command Line Interface
- » 5 Simics Scripting Environment
- » 6 Configuration and Checkpointing**
- » 6.1 Basics
- » 6.2 Checkpointing**
- 6.2.1 Compatibility
- 6.2.2 Attributes
- 6.2.3 Images
- Image Search Path
- 6.2.4 Saving and Restoring Persistent Data
- 6.2.5 Modifying Checkpoints
- 6.2.6 Merging Checkpoints
- 6.3 Inspecting the Configuration
- » 6.4 Components

6.1 Basics

## Simics User's Guide / II Feature Overview / 6 Configuration and Checkpointing / 6.2 Checkpointing

Simics's configuration system can save the complete state of the system as a **checkpoint**, and the saved state is called a **checkpoint**.

Saving and restoring a checkpoint can be done from the command line using the **red-config** commands.

A checkpoint consists of the following files, collected in a directory:

- A main *configuration file*, named **config**. This is a plain text file.
- Optional image files (described in section 6.2.3).

Below is a portion of a checkpoint file showing an object definition. In this example, the object is a PCI bus device connected to an Ethernet LAN controller interfacing the PCI bus) network interface card (NIC).

```
OBJECT dec0 TYPE DEC21143 {
    queue: cpu0
    component: eth_adapter_cmp0
    component_slot: "dec"
    object_id: "dec0"
    build_id: 0xbb9
    pci_bus: pci_bus0
    ...
}
OBJECT ... TYPE ... {
    ...
}
```

Objects are saved in the main checkpoint file in no particular order.

### 6.2.1 Compatibility

Simics maintains checkpoint compatibility with older versions of Simics. Checkpoints created in a previous version of Simics will load correctly in the current version.

Search checkpoint Hide

## G. Optional. Undo mistakes by using checkpoints

Checkpoints can be used to facilitate experimentation on the software state of a target system, by making it easy to go back to a prior known good state. We try this with a simple example.

1. Start a new session by opening the “AfterDriver.ckpt” checkpoint:

In a Windows CMD environment

```
C:\...> simics.bat AfterDriver.ckpt
```

In a Linux environment

```
$ ./simics AfterDriver.ckpt
```

2. Run the simulation:

```
simics> r
```

### Destroy the target state

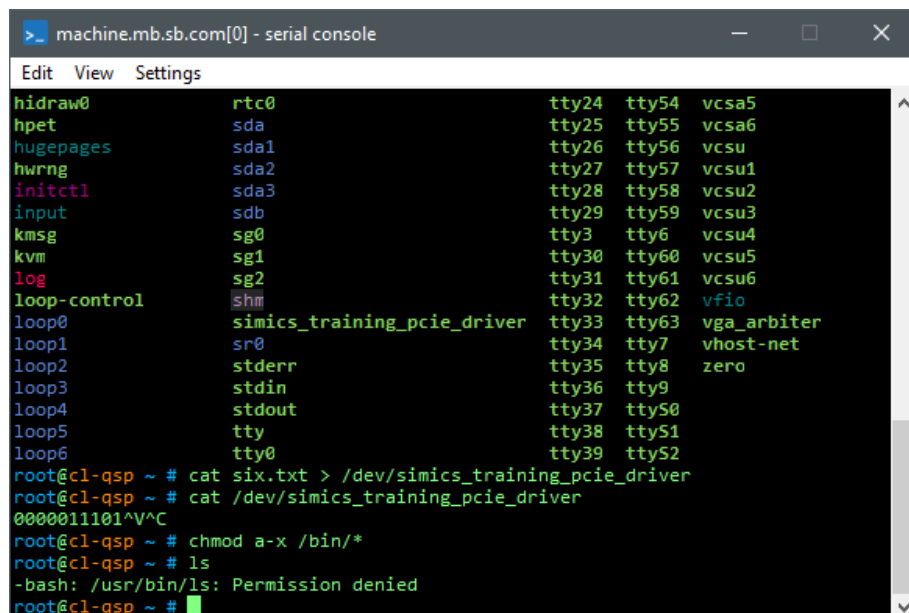
3. Go to the target serial console and enter a command like:

```
# chmod a-x /bin/*
```

Or the classic:

```
# rm -rf /*
```

4. Then, try to run a command like “ls”. Which will give an error since you were root and just destroyed the state of the system.

A screenshot of a serial console window titled "machine.mb.sb.com[0] - serial console". The window shows a list of system components on the left, including hidraw0, hpet, hugepages, hwrng, initctl, input, kmsg, kvm, log, loop-control, loop0, loop1, loop2, loop3, loop4, loop5, loop6, rtc0, sda, sda1, sda2, sda3, sdb, sg0, sg1, sg2, shm, simics\_training\_pcie\_driver, sr0, stderr, stdin, stdout, tty, tty0, tty24, tty25, tty26, tty27, tty28, tty29, tty3, tty30, tty31, tty32, tty33, tty34, tty35, tty36, tty37, tty38, tty39, tty54, tty55, tty56, tty57, tty58, tty59, tty6, tty60, tty61, tty62, tty63, tty7, tty8, tty9, ttyS0, ttyS1, ttyS2, vcsa5, vcsa6, vcsu, vcsu1, vcsu2, vcsu3, vcsu4, vcsu5, vcsu6, vfio, vga\_arbiter, vhost-net, and zero. The console output shows the user running several commands as root@cl-qsp: "cat six.txt > /dev/simics\_training\_pcie\_driver", "cat /dev/simics\_training\_pcie\_driver", "chmod a-x /bin/\*", and "ls". The "ls" command results in a "Permission denied" error. The prompt is root@cl-qsp ~ #.

5. Pause the simulation:

```
simics> stop
```

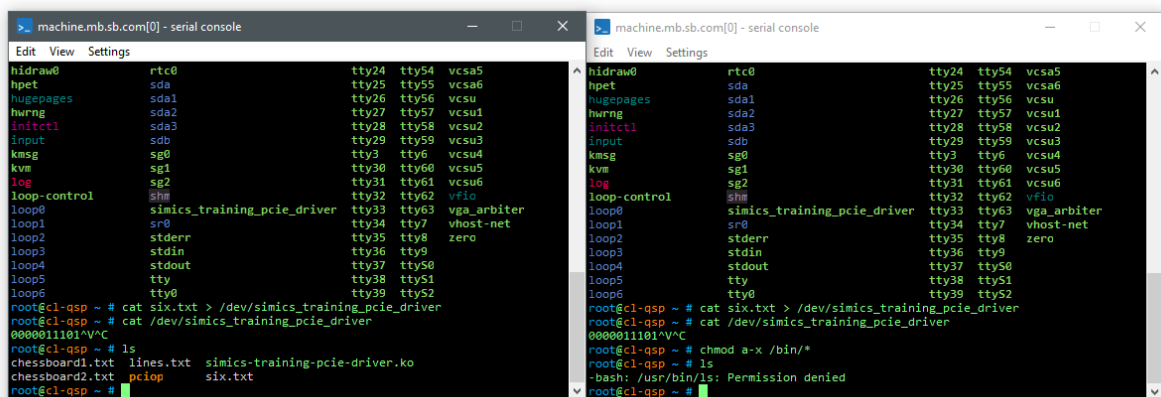


## Open a second session

- Open a new terminal, host shell, or CMD in your project.
- Start a new session by opening the “**AfterDriver.ckpt**” checkpoint , just like above.
- Run the simulation in the new session.
- Go to the serial console window for the new session (it will be at the state where you saved the checkpoint).
- Try running commands in the target Linux system.

It will work since the checkpoint was not affected by the destructive actions done in the previous simulation session.

Comparing the two sessions’ serial consoles:



```
machine.mb.sb.com[0] - serial console
Edit View Settings
hidraw0      rtc0          tty24        tty54        vcsa5
hpet         sda           tty25        tty55        vcsa6
hugepages   sda1          tty26        tty56        vcsu
hwrng        sda2          tty27        tty57        vcsu1
initctl     sda3          tty28        tty58        vcsu2
input        sdb           tty29        tty59        vcsu3
kmsg         sg0           tty3         tty6         vcsu4
kvm          sg1           tty30        tty60        vcsu5
log          sg2           tty31        tty61        vcsu6
loop-control shm           tty32        tty62        vfio
loop0        simics_training_pcie_driver tty33        tty63        vga_arbiter
loop1        sr0           tty34        tty7         vhost-net
loop2        stderr        tty35        tty8         zero
loop3        stdin         tty36        tty9
loop4        stdout        tty37        tty50
loop5        tty           tty38        tty51
loop6        tty0          tty39        tty52

root@cl-qsp ~ # cat six.txt > /dev/simics_training_pcie_driver
root@cl-qsp ~ # cat /dev/simics_training_pcie_driver
0000011101^V^C
root@cl-qsp ~ # ls
chessboard1.txt  lines.txt  simics-training-pcie-driver.ko
chessboard2.txt  pciop     six.txt
root@cl-qsp ~ #

machine.mb.sb.com[0] - serial console
Edit View Settings
hidraw0      rtc0          tty24        tty54        vcsa5
hpet         sda           tty25        tty55        vcsa6
hugepages   sda1          tty26        tty56        vcsu
hwrng        sda2          tty27        tty57        vcsu1
initctl     sda3          tty28        tty58        vcsu2
input        sdb           tty29        tty59        vcsu3
kmsg         sg0           tty3         tty6         vcsu4
kvm          sg1           tty30        tty60        vcsu5
log          sg2           tty31        tty61        vcsu6
loop-control shm           tty32        tty62        vfio
loop0        simics_training_pcie_driver tty33        tty63        vga_arbiter
loop1        sr0           tty34        tty7         vhost-net
loop2        stderr        tty35        tty8         zero
loop3        stdin         tty36        tty9
loop4        stdout        tty37        tty50
loop5        tty           tty38        tty51
loop6        tty0          tty39        tty52

root@cl-qsp ~ # cat six.txt > /dev/simics_training_pcie_driver
root@cl-qsp ~ # cat /dev/simics_training_pcie_driver
0000011101^V^C
root@cl-qsp ~ # chmod a-x /bin/*
root@cl-qsp ~ # ls
-bash: /usr/bin/ls: Permission denied
root@cl-qsp ~ #
```

- Quit both simulation sessions.