Intel® Simics® Simulator Internals Training

Lab 01-02

Objects, Attributes, Interfaces, Classes

1 Introduction

In this lab, you will look at a running Intel® Simics® simulation to investigate the simulator object model and learn how to inspect a the system. It will look at how Simics memory maps work, and how to create objects and connect them to other objects.

2 Preliminary Note: Windows vs Linux Host

The instructions in this set of labs are all written using examples from a Windows host. The labs work just as well and in the same way on a Linux host. However, for brevity, only instructions for Windows are shown (except in cases where there is a significant difference).

Here is how you translate from Windows instructions to Linux instructions for the cases covered in this lab module:

2.1 Directory separator / \

The labs always use **/** when writing paths inside of Simics itself, since Simics automatically translates **/** to the appropriate type of separator for the host. Thus, Simics commands are always host-independent if written correctly.

Note that tab-completing a path on Windows from Simics CLI will produce Windows-style directory separators.

When referring to files to open in Eclipse or other editors, the labs use **/** consistently since those paths will typically be accessed using the Eclipse Project view or a file picker. Thus copy-paste of file paths is not expected to be used.

When the labs involve working from a **shell**, command examples are written using **** in order to work when copy-pasted from the lab document to the shell. On Linux, you need to replace **** with **/** in all file paths.

2.2 Starting Simics from shell

Assuming you have a shell in the project.

On Windows, there is a **`simics.bat`** script at the top level that is used to start Simics:

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

On Linux, there is a **`simics`** script at the project top level.

```
$ ./simics
```

3 Inspecting the System Structure

This lab covers how to find classes and objects and it also illustrates the relationships between classes, objects, attributes, ports and interfaces.

3.1 Start Simics

1. Start Simics with the Quick Start Platform with Clear Linux from within your project.

   On Linux:

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

   On Windows:

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

3.2 List classes and objects

In some cases, you search for "something of interest". For example, you might suspect that the i2c communication is somewhat broken and you want to trace and log what goes on at the corresponding devices. You might not know the exact device names, nor what classes they are instantiated from.

2. List all classes that are currently loaded in the Simics configuration.

   ```
   simics> list-classes -l
   ```

   You will see lots of output, like below.

   ```
   The following classes have been registered:
   ┌──────────────────────────────────────────────────────────────────────────────┬──────────────────────────────────────────────────────────────────────────────┐
   │                                    Class                                     │                                Short description                             │
   ├──────────────────────────────────────────────────────────────────────────────┼──────────────────────────────────────────────────────────────────────────────┤
   │ ISA                                                                          │ model of standard PC bus                                                     │
   │ NS16450                                                                      │ model of NS 16450 UART                                                       │
   │ NS16450.HRESET                                                               │ simulates resetting by power loss                                            │
   […]
   │ rgb_led_i2c                                                                  │ rgb led on I2C bus                                                           │
   […]
   │ x86_broadcast                                                                │ model of CPU broadcast device                                                │
   │ x86_reset_signal_conv                                                        │ converter of reset signals                                                   │
   └──────────────────────────────────────────────────────────────────────────────┴──────────────────────────────────────────────────────────────────────────────┘
   ```

3. Sticking with the "we are interested in i2c devices" scenario, a class with i2c in its name strikes us as interesting, so let's see if there are more.

   ```
   simics> list-classes -l substr = i2c
   ```

   Which gives us all classes with i2c in their names.

   ```
   The following classes have been registered:
   ┌──────────────────────────────────────────────────────────────────────────────┬──────────────────────────────────────────────────────────────────────────────┐
   │                                    Class                                     │                                Short description                             │
   ├──────────────────────────────────────────────────────────────────────────────┼──────────────────────────────────────────────────────────────────────────────┤
   […]
   │ rgb_led_i2c                                                                  │ rgb led on I2C bus                                                           │
   │ toggle_i2c                                                                   │ toggle on I2C bus                                                            │
   └──────────────────────────────────────────────────────────────────────────────┴──────────────────────────────────────────────────────────────────────────────┘
   ```

   There are a few of them.

4. Let's focus on the I2C connected LED now and narrow down the list to only it and its port classes.

```
simics> list-classes -l substr = rgb_led_i2c -show-port-classes
```

The output shows us that the class defines two port classes, so we would expect only two types of port objects in the device.

```
The following classes have been registered:
┌──────────────────────┬──────────────────────────────────────────────────┐
│        Class         │                Short description                 │
├──────────────────────┼──────────────────────────────────────────────────┤
│rgb_led_i2c           │rgb led on I2C bus                                │
│rgb_led_i2c.i2c_in    │Inbound I2C traffic                               │
│rgb_led_i2c.i2cregs   │Register bank holding I2C state (not software addr│
└──────────────────────┴──────────────────────────────────────────────────┘
```

5. Now, let's check which objects have been created from the rgb_led_i2c class.

```
simics> list-objects class = rgb_led_i2c
```

We see that 64 objects are created from that class.

```
┌──────────────────────────────────┬──────────────┐
│              Object              │    Class     │
├──────────────────────────────────┼──────────────┤
│machine.training_card.rgb_led[0]  │<rgb_led_i2c> │
│machine.training_card.rgb_led[1]  │<rgb_led_i2c> │
│machine.training_card.rgb_led[2]  │<rgb_led_i2c> │
│machine.training_card.rgb_led[3]  │<rgb_led_i2c> │
[…]
│machine.training_card.rgb_led[63]│<rgb_led_i2c> │
└──────────────────────────────────┴──────────────┘
```

3.3 Identifying interfaces, ports and attributes

6. To see what interface are offered by such a device either directly or through its ports, use the help command.

```
simics> help rgb_led_i2c
```

You will see what interfaces are offered by the device (like **conf_object** or **log_object**), and you can also see the port objects and the interfaces they offer. You can quickly spot the **i2c_in** port implements the **i2c_slave_v2** interface , while **bank.i2cregs** is a register bank.

Note: You could also invoke help on an object to get the help text for its class.

7. Small excursus: If you had known that you are looking for devices implementing the **i2c_slave_v2** interface, you can list them directly:

```
simics> list-objects iface = i2c_slave_v2 -show-port-objects
```

You can see that there are more classes than just the rgb_led_i2c class that implement the **i2c_slave_v2** interface.

```
┌──────────────────────────────────────────┬──────────────────────────────────────┐
│                  Object                  │                Class                 │
├──────────────────────────────────────────┼──────────────────────────────────────┤
│machine.mb.sb.smbus                       │<ich10_smbus_i2c_v2>                  │
│machine.mb.sb.smbus.port.slave_side       │<ich10_smbus_i2c_v2.slave_side>       │
│machine.mb.smbus.ep2ac11899d3eb2caf       │<i2c-link-endpoint>                   │
│machine.mb.smbus.ep72e3d343bd50cdb2       │<i2c-link-endpoint>                   │
│machine.mb.smbus.epbf51bed6c9c6b49f       │<i2c-link-endpoint>                   │
│machine.training_card.button_a            │<button_i2c>                          │
│machine.training_card.button_a.i2c_ep     │<i2c-link-endpoint>                   │
│machine.training_card.button_a.port.i2c_in│<button_i2c.i2c_in>                   │
│machine.training_card.button_b            │<button_i2c>                          │
│machine.training_card.button_b.i2c_ep     │<i2c-link-endpoint>                   │
│machine.training_card.button_b.port.i2c_in│<button_i2c.i2c_in>                   │
│machine.training_card.master_bb           │<led_system_controller_bb>            │
│machine.training_card.master_bb.i2c_ep    │<i2c-link-endpoint>                   │
│machine.training_card.master_bb.port.i2c_in│<led_system_controller_bb.i2c_in>     │
│machine.training_card.rgb_led[0]          │<rgb_led_i2c>                         │
[…]
│machine.training_card.toggle_u            │<toggle_i2c>                          │
│machine.training_card.toggle_u.i2c_ep     │<i2c-link-endpoint>                   │
│machine.training_card.toggle_u.port.i2c_in│<toggle_i2c.i2c_in>                   │
└──────────────────────────────────────────┴──────────────────────────────────────┘
```

8. Now that we know what instances of **rgb_led_i2c** exist, we can inspect the attributes of them. Let's pick the first of them as an example.

```
simics> list-attributes machine.training_card.rgb_led[0]
```

Note that you can see the data types (Column "**Type**"), the configuration types (Column "**Attr**") and the actual values. You can see that the device has only one "required" attribute, which means that in order to instantiate such a device, a value must be provided for that attribute. All others are "optional" (have a safe default or can be assigned later) or "pseudo" (just "gateways" into the device to, for example, set multiple attributes at once with a single assignment).

```
┌──────────────────────────┬──────┬──────────┬─────┬──────────────────────────────────────┐
│Attribute                 │Type  │Attr      │Flags│Value                                 │
├──────────────────────────┼──────┼──────────┼─────┼──────────────────────────────────────┤
│address                   │i     │Required  │     │16                                    │
│current_color_value_driven│i     │Optional  │     │0                                     │
│drive_value               │b     │Pseudo    │     │                                      │
│i2c_link                  │o|[os]|n│Optional│     │machine.training_card.rgb_led[0].i2c_ep│
│panel_led_out             │o|[os]|n│Optional│     │machine.training_card.panel.led_0_0   │
│written_value             │i     │Optional  │     │0                                     │
└──────────────────────────┴──────┴──────────┴─────┴──────────────────────────────────────┘
```

There are two attributes of type "**o|[os]|n**". These are attributes that can point to (port) objects ("**o**"), legacy "named ports" ("**[os]**"), or nothing/None/Null ("**n**"). They are the "outgoing" parts of an object->interface->(port)object connection. On device will call interface function on these attributes. Their current value shows which (port) object will receive these calls. More information on this is in section 3.4.

9. To find more information about an attribute, do **help** on it. The **local_memory** attribute of the **master_bb** object is a typical required configuration attribute:

```
simics> help machine.training_card.rgb_led[0]->address
```

There you can learn what the attribute is used for:

```
Attribute <rgb_led_i2c>.address

   Required attribute; read/write access; type: integer.

   7-bit address on the i2c bus. It can be any valid i2c address in range of [0, 127].
```

10. You can also see internal attributes.

```
simics> list-attributes machine.training_card.rgb_led[0] -i
```

You get a much longer list, containing attributes that are either generated during build are used internally by Simics or that have explicitly been defined as internal by the model creator. Such attributes are rarely interacted with by users, but in some cases, they help with debugging.

```
┌──────────────┬──────┬──────────────────┬─────┬──────────────────────────────────┐
│Attribute     │Type  │Attr              │Flags│Value                             │
├──────────────┼──────┼──────────────────┼─────┼──────────────────────────────────┤
│access_count  │i     │Pseudo Internal   │     │0                                 │
│address       │i     │Required          │     │16                                │
[...]
│name          │s     │Pseudo Internal   │     │"machine.training_card.rgb_led[0]"│
│object_id     │s     │Optional Internal │     │"obj_000004eb4626b4e1"            │
│panel_led_out │o|[os]|n│Optional          │     │machine.training_card.panel.led_0_0│
│ports         │[s*]  │Pseudo Internal   │     │["i2c_in", "i2cregs"]             │
│queue         │o|n   │Optional Internal │     │machine.mb.cpu0.core[0][0]        │
│written_value │i     │Optional          │     │0                                 │
└──────────────┴──────┴──────────────────┴─────┴──────────────────────────────────┘
```

3.4 Object references

Simics objects refer to other objects that they communicate with via object references in attributes. Follow such a chain from the **master_bb**. You will look more in depth into this in the next lab section.

11. On the Simics command line, look at the **i2c_link** attribute of the **rgb_led[0]** object:

```
simics> help machine.training_card.rgb_led[0]->i2c_link
```

Resulting in:

```
Attribute <rgb_led_i2c>.i2c_link

   Optional attribute; read/write access; type: [os], object, or nil.

   I2C link outbond connection

   Required interfaces: i2c_master_v2.
```

The output tells us that the object pointed to by this attribute must implement the master interfaces for i2c, and that it can be either an object or an **(object, portname)** pair.

12. Check the current value of the attribute:

```
simics> machine.training_card.rgb_led[0]->i2c_link
```

The result should be the i2c endpoint associated to the **rgb_led[0]** object:

```
"machine.training_card.rgb_led[0].i2c_ep"
```

This kind of indirect connection between a network-connected object and its network link is common to many Simics networks (including Ethernet). Each endpoint is private to each connected device, and provides a "stand in" for the link.

13. Investigate the endpoint object:

```
simics> help machine.training_card.rgb_led[0].i2c_ep
```

Resulting in:

```
Class i2c-link-endpoint

   Description
      Connects a link with a device

   Interfaces Implemented
      conf_object, i2c_master_v2, i2c_slave_v2, link_endpoint_v2, log_object

   Provided By
      i2c-link-v2
…
```

Note how it implements the required interface **`i2c_master_v2`** – otherwise, setting the object reference would have failed.

14. To check the current configuration of the endpoint, use the info command on it:

```
simics> machine.training_card.rgb_led[0].i2c_ep.info
```

The info command shows the actual link, as well as the pointer back to the **`master_bb`**:

```
Information about machine.training_card.rgb_led[0].i2c_ep [class i2c-link-endpoint]
================================================================================

                         Link : machine.training_card.i2c_bus (i2c-link-impl)
             Connected device : machine.training_card.rgb_led[0].port.i2c_in
    I2C addresses listened to : 0x20
                                0x21
```

The reference back to the **`rgb_led[0]`** object is a port object.

3.5 Inspecting the system structure top-down

If you want to familiarize yourself with a system, you may not search for something specific but just get a grasp on the system structure, the objects within and the classes used. This sections covers how to do that.

15. From the command line, you use the **`list-components`** command to show components and their connectors. Note that the command shows all the components inside the given namespace. Thus, to see the connectors on **`machine.training_card`**, you have to do **`list-components`** on the machine level:

```
simics> list-components machine -v
```

Which should show something like:

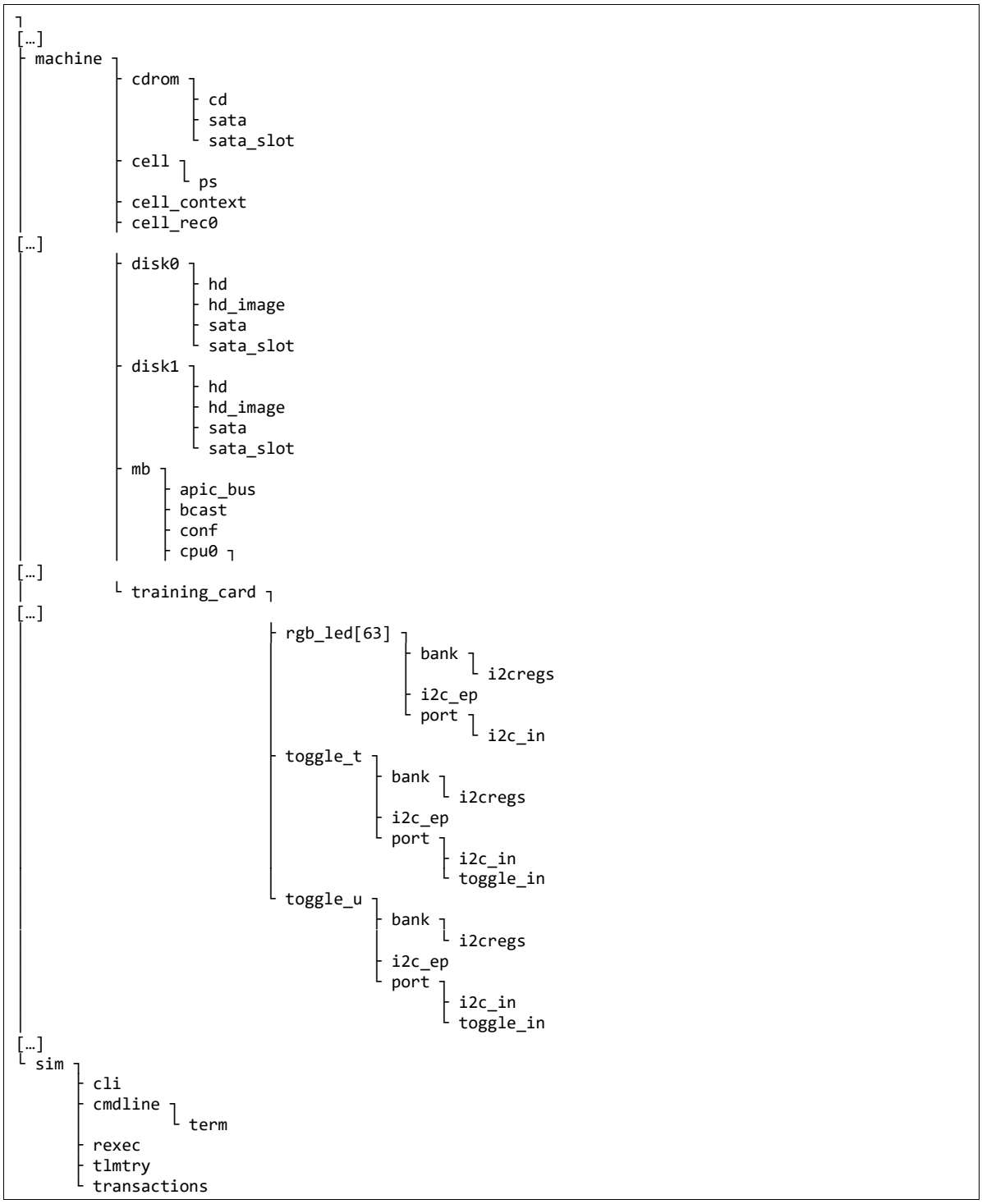
```
simics> list-components machine -v
```

Component	Class		
cdrom	sata_cdrom_comp		
sata_slot	sata-slot	up	machine.mb.sb:sata_slot[1]
console	gfx_console_comp		
abs_mouse	abs-mouse	up	machine.tablet:abs_mouse
device	graphics-console	up	machine.mb.gpu:console
keyboard	keyboard	up	machine.mb.sb:kbd_console
mouse	mouse	up	machine.mb.sb:mse_console
…			
tablet	usb_tablet_component		
abs_mouse	abs-mouse	down	machine.console:abs_mouse
usb_host	usb-port	up	machine.mb.sb:usb_port[0]
training_card	led_system_comp		
pci	pci-bus	up	machine.mb.nb:pcie_slot[1]

16. To see the complete hierarchy of the Simics configuration, use the **`list-objects`** command with the **`-tree`** argument.

```
simics> list-objects -tree
```

This prints all the objects in the hierarchy, which can be rather overwhelming. The last part would look something like below. There you can see all objects and then use previously introduced commands to learn more about the object's attributes, ports, and interfaces.



17. **`List-objects`** can be used to get a sense for the size of the configuration. You do this by sending the output from **`list-objects`** to **`list-length`**. With **`-all`**, **`list-objects`** lists all the objects in the Simics configuration:

```
simics> list-length (list-objects -all)
```

This rather simple target machine contains more than 800 objects (!).

18. How much of this is due to the rather object-rich **`training_card`** subsystem? Count all the objects inside the namespace **`machine.training_card`**, recursively:

```
simics> list-length (list-objects namespace=machine.training_card)
```

This number is about a third of the total number of objects.

3.6 Filtering list-object output

It common to look for specific objects using **`list-objects`** with various filters.

19. To see where the **`master_bb`** device sits in the system hierarchy, combine **`-tree`** with a **`substr=`** filter:

```
simics> list-objects -tree substr = "rgb_led[0]" -show-port-objects
```

The output will show the hierarchy leading to master_bb, as well as all objects underneath it:

```
machine
    training_card
        rgb_led[0]
            bank
                i2cregs
            i2c_ep
            port
                i2c_in
```

20. To list the objects inside a component or hierarchical object from the command line, use the **`list-objects`** command with the **`namespace`** argument:

```
simics> list-objects namespace = machine.training_card
```

For a component like **`training_card`**, all objects contained within the components are shown.

3.7 Setting default table-driven output format

Many Simics **`list-`** commands use the Simics table system to provide output formatting that helps put a lot of information onto the screen at once. The default border of the CLI tables is to draw a “thin” line around the contents and the headers, but this can be controlled by user. The rest of the model builder labs have been created using no border at all, as that has less overhead and fits more information into a limited area.

21. Look at what the current border style looks like. Using a command such as **`list-attributes`** on an object:

```
simics> list-attributes machine.training_card.i2c_bus
```

With the default thin border, it looks something like this:

Attribute	Type	Attr	Flags	Value
class_desc	s|n	Pseudo		NIL
effective_latency	f	Pseudo		1e-08
endpoints	[o*]	Pseudo		[machine.training_card.rgb_led[34].i2c_ep, machine.training_card.rgb_led[13].i2c_ep, machine.trainin ...
global_id	s|n	Optional		"obj_000002c10b16b888"
goal_latency	f	Required		1e-08
immediate_delivery	b	Optional		FALSE
outside_cell	b	Optional		TRUE
stats	[iiii]	Pseudo		[0, 0, 0, 0]

22. Change the default setting to **`borderless`**:

```
simics> prefs->cli_table_border_style = borderless
```

23. Check what this looks like by repeating the previous command:

```
simics> list-attributes machine.training_card.i2c_bus
```

Which results in a more compact output, but that might also be harder to read as the contents is less well-separated:

```
Attribute           Type    Attr      Flags  Value
class_desc          s|n     Pseudo           NIL
effective_latency   f       Pseudo           1e-08
endpoints           [o*]    Pseudo           [machine.training_card.rgb_led[34].i2c_
                                             ep, machine.training_card.rgb_led[13].
                                             i2c_ep, machine.trainin ...
global_id           s|n     Optional         "obj_000002c10b16b888"
goal_latency        f       Required         1e-08
immediate_delivery  b       Optional         FALSE
outside_cell        b       Optional         TRUE
stats               [iiii]  Pseudo           [0, 0, 0, 0]
```

24. If you want to use this style going forward, save your settings so that they apply to all future sessions from this Simics project:

```
simics> save-preferences
```

3.8 Optional: Scripting with lists of objects

25. It is possible to retrieve a list of objects into a variable, and iterate over the elements of that list to perform operations on multiple objects. To retrieve a list of all **button_i2c** objects, assign the return value of the command to a list:

```
simics> $l = (list-objects class = button_i2c)
```

26. To check the list:

```
simics> $l
```

Which should return:

```
["machine.training_card.button_a", "machine.training_card.button_b"]
```

27. Iterate over all the found objects, and call the info command on each:

```
simics> foreach $b in $l { $b.info }
```

This command should result in output similar to this, printing information about the buttons:

```
Information about machine.training_card.button_a [class button_i2c]
=====================================================================

I2C configuration:
        I2C device address : 92
    I2C address to notify  : 100
                  I2C link : machine.training_card.button_a.i2c_ep
Information about machine.training_card.button_b [class button_i2c]
=====================================================================

I2C configuration:
        I2C device address : 93
    I2C address to notify  : 100
                  I2C link : machine.training_card.button_b.i2c_ep
```

28. **Quit** this Simics session.