Automation of device and machine development for QEMU*

Vasiliy Efimov <real@ispras.ru> (corresponding)
Aleksandr Bezzubikov <abezzubikov@ispras.ru>
Danila Bogomolov <bda@ispras.ru>
Oleg Goremykin <goremykin@ispras.ru>
Vartan Padaryan <vartan@ispras.ru>

Ivannikov Institute for System Programming of the RAS

2017 Ivannikov ISPRAS Open Conference, Moscow, 1st December

*This work is supported by RFBR, grant No 16-29-09632
The problem

Development of either device or machine model for QEMU is a time-consuming task. Therefore, an automation is required.

<table>
<thead>
<tr>
<th>Emulator/Simulator</th>
<th>Machine composing</th>
<th>Device development automation</th>
<th>Debug Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GUI</td>
<td>API</td>
<td></td>
</tr>
<tr>
<td>Simics</td>
<td>+</td>
<td>C++, Python</td>
<td></td>
</tr>
<tr>
<td>AMD SimNow</td>
<td>+</td>
<td>C++</td>
<td></td>
</tr>
<tr>
<td>gem5</td>
<td>.</td>
<td>C++, Python</td>
<td></td>
</tr>
<tr>
<td>gem5</td>
<td>.</td>
<td>C, TCL → C</td>
<td>+</td>
</tr>
<tr>
<td>QEMU</td>
<td>+</td>
<td>C, CLI</td>
<td></td>
</tr>
</tbody>
</table>
The goal

The goal is to automate the development of both device and machine for QEMU emulator.

Objectives

- analyse QEMU internals
- search the workflow for stages to automate
- develop a toolset automation
- evaluate the toolset
Automation concepts

- Classic development workflow
- QEMU Object Model (QOM)
- Device modelling API
- Machine composition API
- Proposed workflow
Automation concepts

Classic development workflow

Both device & machine documentation is frequently written in a domain specific natural language. It cannot be formalized.

Device drafts writing

A detached device is difficult to debug.

Iterative development

A machine cannot be composed until most of its devices are available.

Documentation analyze

Initial machine composing

Improve

Debug
QEMU Object Model (QOM)

- OOP in C (like Gnome library's `gobject`)
- Hierarchy node is called a *type*
- type = *class* + *instance*
- class/instance
  = *structure* ([`struct`, `C`])
  + *constructor* (a callback, `C`)
- object is a type that supports *properties* to both class and instance.
- a property is an opaque value identified by a *string* and accessed through set and get function.
Device modelling API

*Interface code* is composed using finite set of API elements. Each the element has finite set of parameters. Given those parameters, a draft with interface code stubs can be generated for a device.
Machine composition API (example)

/* device instance creation */
dev = qdev_create(parent_bus, QOM_TYPE_NAME);

/* specification of properties */
object_property_set_TYPE(dev, PROP_VALUE, PROP_NAME, ...);

/* device instance "realization" */
qdev_init_nofail(dev);

/* mapping of registers */
sysbus_mmio_map(dev, REG_INDEX, REG_ADDRESS);

/* interrupt lines binding */
my_incoming_irq = qdev_get_gpio_in(dev, IN_IRQ_INDEX);
sysbus_connect_irq(dev, OUT_IRQ_INDEX, neighbour_incoming_irq);
Machine content is described in a declarative way.

- An object model is used for content description.
- A complicated device interconnection is difficult to sense in form of code.

Therefore, the graphical editor was implemented. It represents a machine in a schematic form. The editor generates a code for the machine draft.
Proposed workflow

Information required for the 1-st stage:
- list of devices
- list of required QEMU API elements per device
- device interconnection (machine)

Documentation analyze
Generation settings

Iterative development
Generation of drafts

All the devices are interconnected and ready to debug.

Next, a developer have to implement individual part of the device model.
Developed toolset

- Toolset infrastructure
- Settings format
- Generator capabilities
- Examples
- GUI
- Existing QEMU code feedback
Developed toolset

GUI

File  Edit  ...  ?

INT_CTRLRLR

MY_UART

bus

Generator

settings

API

manual
writing

.py

Existing
code
feedback

Model
draft
description

File
content

Templates

{  

(void)

;

}

Code
generator

/* Chunks of */
code();

...

...
## Device draft generation capabilities

<table>
<thead>
<tr>
<th>Device class</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>QOM registration</td>
</tr>
<tr>
<td></td>
<td>VM state and property declaration</td>
</tr>
<tr>
<td></td>
<td>timers</td>
</tr>
<tr>
<td></td>
<td>character and block devices</td>
</tr>
<tr>
<td></td>
<td>network interface</td>
</tr>
<tr>
<td>System bus device</td>
<td>MMIO</td>
</tr>
<tr>
<td></td>
<td>PMIO</td>
</tr>
<tr>
<td></td>
<td>in/out IRQ</td>
</tr>
<tr>
<td>PCI(E) device function</td>
<td>BAR</td>
</tr>
<tr>
<td></td>
<td>out IRQ (INTx)</td>
</tr>
<tr>
<td></td>
<td>MSI(X)</td>
</tr>
<tr>
<td></td>
<td>identification information</td>
</tr>
</tbody>
</table>
Fast Ethernet adapter draft generation settings example

```python
obj55 = PCIExpressDeviceDescription(
    name = "AM79C971" ,  # model name
    vendor = "0x1022" , device = "0x2000" , pci_class = "0x0200" ,
    revision = 0x1 ,
    # subsys = None , subsys_vendor = None ,
    directory = "net" ,  # directory name
    irq_num = 0x1 ,
    mem_bar_num = 0x1 ,
    nic_num = 0x1 ,
    timer_num = 0x1 ,
    # msi_messages_num = 0 ,
    # char_num = 0 ,
    # block_num = 0
)
```
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>AM79C971</td>
</tr>
<tr>
<td>Directory</td>
<td>net</td>
</tr>
<tr>
<td>Block driver quantity</td>
<td>0</td>
</tr>
<tr>
<td>Character driver quantity</td>
<td>0</td>
</tr>
<tr>
<td>Timer quantity</td>
<td>1</td>
</tr>
<tr>
<td>Network interface</td>
<td>✓</td>
</tr>
<tr>
<td>Vendor ID</td>
<td>AMD</td>
</tr>
<tr>
<td>Device ID</td>
<td>AMD_LANCE</td>
</tr>
<tr>
<td>PCI class code</td>
<td>NETWORK_ETHHERNET</td>
</tr>
<tr>
<td>IRQ pin quantity</td>
<td>1</td>
</tr>
<tr>
<td>BAR quantity</td>
<td>1</td>
</tr>
<tr>
<td>MSI message quantity</td>
<td>0</td>
</tr>
<tr>
<td>Revision</td>
<td>1</td>
</tr>
</tbody>
</table>
Machine content description

This type hierarchy is based on QOM.

IRQHub allows to deliver one IRQ to many devices.

Most part of memory address space is defined by devices internally. But several kinds of memory (like a RAM or a simple ROM) have to be defined explicitly. MemoryNode ancestors are used for it.
Developed toolset
Bus interconnection example in GUI
Developed toolset

IRQ line interconnection example in GUI

![Image of GUI for IRQ line settings]
Existing QEMU code feedback

- **Automatic header analysis**
  - Inclusion graph (used to generate header inclusions)
  - Preprocessor macros (used by both GUI and generator core)

- **Heuristic based support for different QEMU version.**
  - A new value is propagated towards future commits.
  - An old value is propagated:
    1. towards past commits,
    2. towards future commits.
  - During merging new values are chosen.
  - Given SHA1, the actual value can be obtained.
The toolset usage examples

- Intel Q35 chipset based PC
- CISCO 2600 series router (C2621XM)
Intel Q35 chipset based PC

- There is another implementation in QEMU already. It is one of most complicated machines in the emulator.
- The goal of this experiment is to prove the proposed workflow correctness.
- All required devices are already present in QEMU.
- Several old devices were updated using the toolset.
Q35 machine scheme
## Evaluation*

<table>
<thead>
<tr>
<th>Stage</th>
<th>Files touched</th>
<th>Lines inserted</th>
<th>Lines deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation**</td>
<td>4</td>
<td>42</td>
<td>31</td>
</tr>
<tr>
<td>Generation</td>
<td>8</td>
<td>599</td>
<td>0</td>
</tr>
<tr>
<td>Implementation</td>
<td>5</td>
<td>162</td>
<td>93***</td>
</tr>
<tr>
<td>Total</td>
<td>12</td>
<td>803</td>
<td>31</td>
</tr>
</tbody>
</table>

*The measurements were made using git diff.

**A refactoring mostly.

***Note that amount of deleted lines is a measure of piece of generated code to be adjusted.
C2600 series router (C2621XM)

- Based on Dynamips.
- CPU PowerPC MPC860 presents in QEMU except for full system emulation support.
- Both machine and devices were implemented using the toolset (except for CPU).
C2621XM router scheme
## Evaluation

<table>
<thead>
<tr>
<th>Stage</th>
<th>Files touched</th>
<th>Lines inserted</th>
<th>Lines deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation*</td>
<td>8</td>
<td>128</td>
<td>35</td>
</tr>
<tr>
<td>Generation</td>
<td>37</td>
<td>2186</td>
<td>0</td>
</tr>
<tr>
<td>Implementation</td>
<td>31</td>
<td>4747</td>
<td>419</td>
</tr>
<tr>
<td>Total</td>
<td>45</td>
<td>6642</td>
<td>35</td>
</tr>
</tbody>
</table>

*Memory management unit, CPU’s special registers and interrupt support, PCI identifiers.*
## Evaluation*

<table>
<thead>
<tr>
<th>Device</th>
<th>Configuration size</th>
<th>Draft size</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPC860_IC</td>
<td>6</td>
<td>125</td>
</tr>
<tr>
<td>C2600_PCI_HOST</td>
<td>6</td>
<td>133</td>
</tr>
<tr>
<td>C2600_PCI</td>
<td>7</td>
<td>82</td>
</tr>
<tr>
<td>NS16552</td>
<td>7</td>
<td>181</td>
</tr>
<tr>
<td>C2600_IO_FPGA</td>
<td>8</td>
<td>137</td>
</tr>
<tr>
<td>CISCO_REMOTE</td>
<td>7</td>
<td>152</td>
</tr>
<tr>
<td>AM79C971</td>
<td>12</td>
<td>175</td>
</tr>
</tbody>
</table>

*The size is measured in lines.*
Conclusion

- Results
- Future work
Results

- The first stage of device and machine model development was automated using the code draft generation toolset.
- A generation configuration is written in Python.
- The size of resulting device draft is 11-25 times bigger than size of corresponding configuration.
- The GUI was implemented including schematic machine editor.
- The toolset supports complex machines like Intel Q35.
- The piece of generated code is between 1/4 and 3/4 depending on amount of available device models.
- Existing QEMU code is accounted including QEMU version adaptation mechanism.
Future work

Runtime debug feedback form QEMU.
The End

Thank you for your attention!

Questions?