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.

Emulator/ Simulator	Machine composing		Device development automation		Debug Tools
	GUI	API	GUI	API	
Simics	+	C++, Python	.	DML $\rightarrow$ C++	.
AMD SimNow	+	C++	.	.	.
gem5	.	C++, Python	.	.	.
OVPSim	.	C, TCL $\rightarrow$ C	.	TCL $\rightarrow$ C	+
QEMU	+	C, CLI	.	.	.

The goal

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

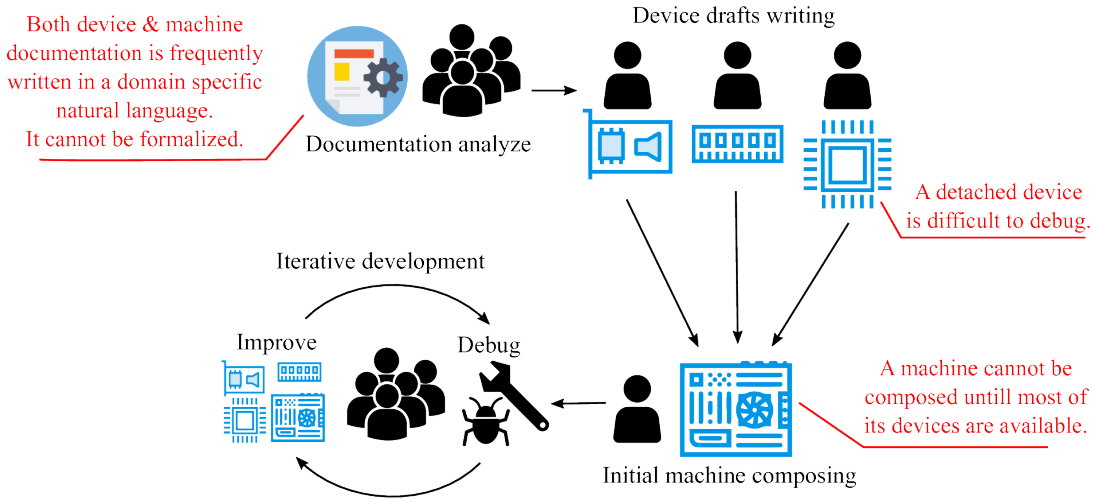
Objectives

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

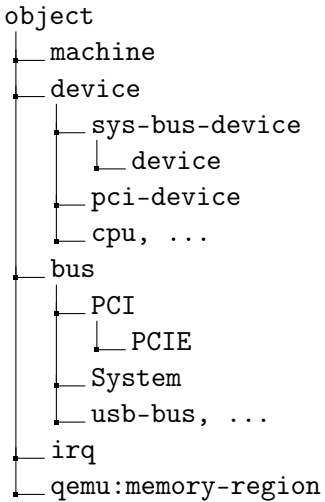
Automation concepts

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

Classic development workflow

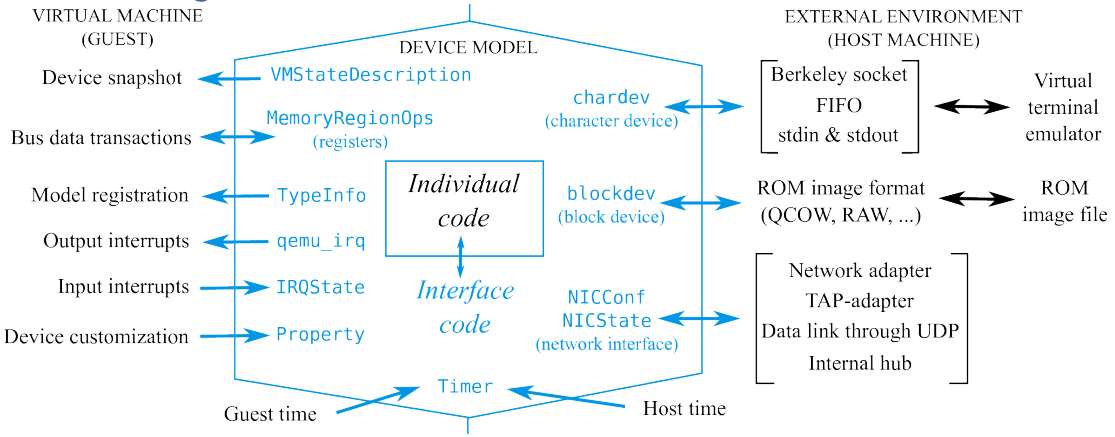


QEMU Object Model (QOM)



- 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)

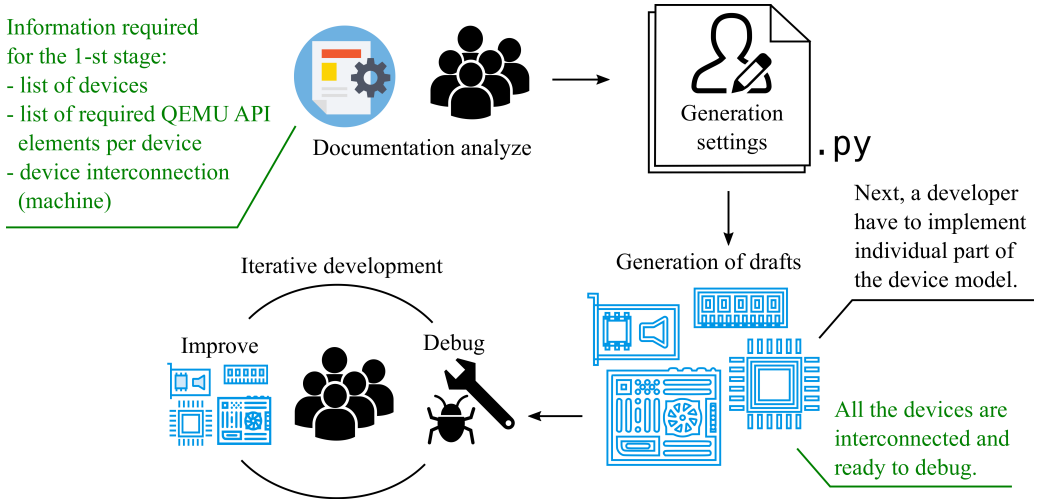
```
/* device instance creation */
dev = qdev_create(parent_bus, QOM_TYPE_NAME);

/* 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 composition API specifics

- 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.

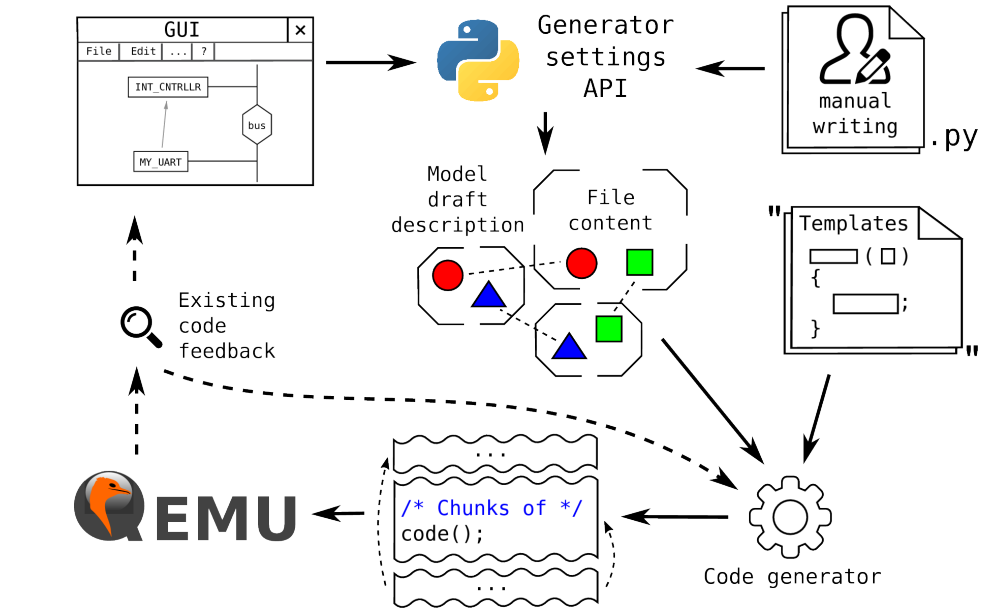
Therefore, the graphical editor was implemented. It represent a machine in a schematic form. The editor generates a code for the machine draft.

Proposed workflow



Developed toolset

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

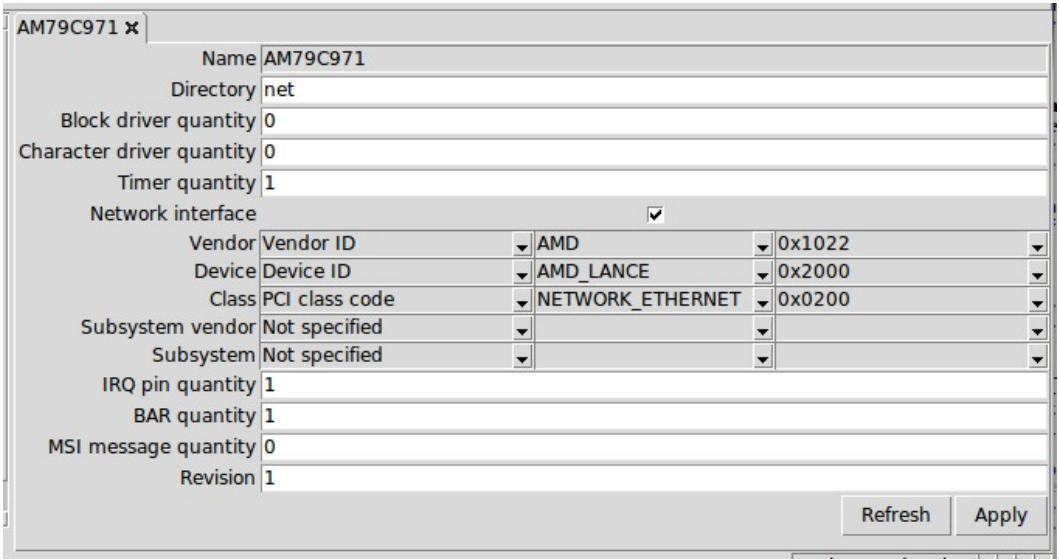

GUI
File
Edit
...
?
INT_CNTRLLR
bus
MY_UART
Generator
settings
API
manual
writing
.py
Model
draft
description
File
content
Templates
Existing
code
feedback
QEMU
/* Chunks of */
code();
Code generator

Device draft generation capabilities

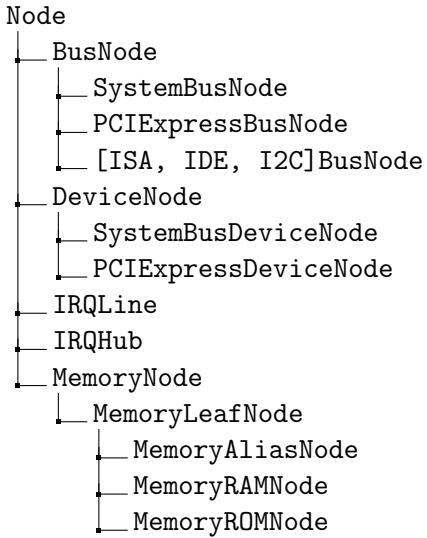
Device class	Capabilities
Any	QOM registration
	VM state and property declaration
	timers
	character and block devices
	network interface
System	MMIO
bus	PMIO
device	in/out IRQ
PCI(E)	BAR
device	out IRQ (INTx)
function	MSI(X)
	identification information

Fast Ethernet adapter draft generation settings example

```
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
)
```


AM79C971
Name
AM79C971
Directory
net
Block driver quantity
0
Character driver quantity
0
Timer quantity
1
Network interface
Vendor
Vendor ID
AMD
0x1022
Device
Device ID
AMD_LANCE
0x2000
Class
PCI class code
NETWORK_ETHERNET
0x0200
Subsystem vendor
Not specified
Subsystem
Not specified
IRQ pin quantity
1
BAR quantity
1
MSI message quantity
0
Revision
1
Refresh
Apply

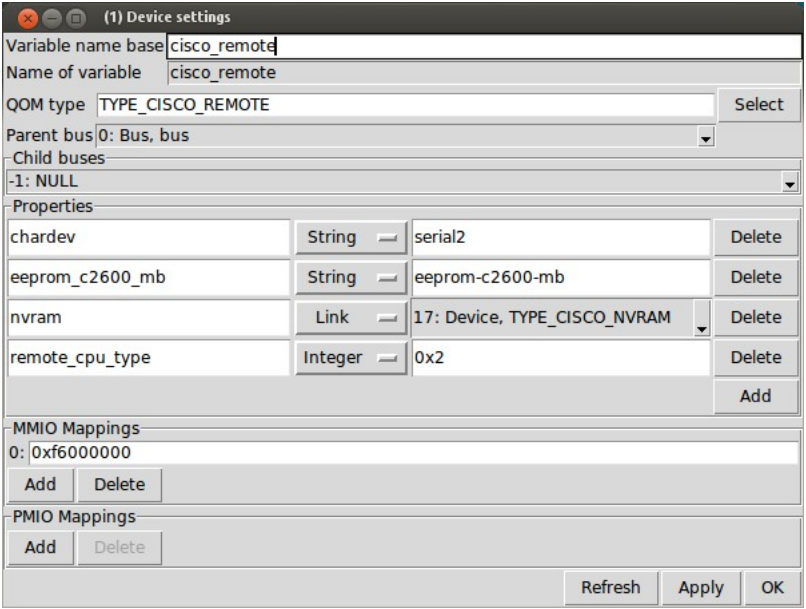
Machine content description



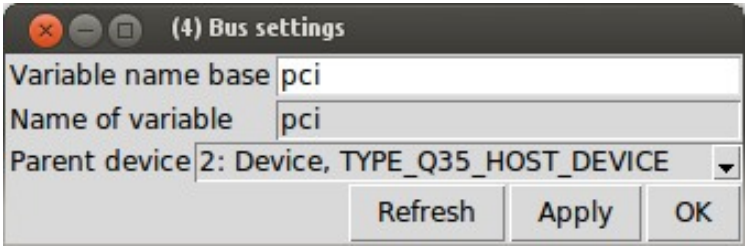
This type hierarchy is based on QOM.

`IRQHub` allows to deliver one IRQ to many devices.

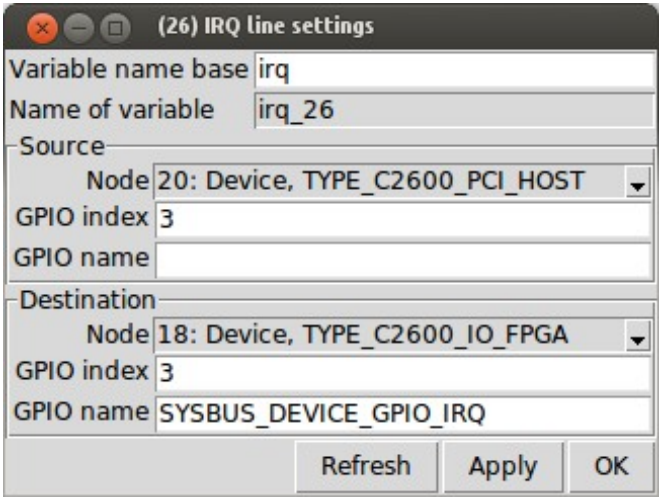
Most part of memory address space is defined by devices internally. But several kinds of memory (like a RAM or a simple ROM) have to be defined explicitly. `MemoryNode` ancestors are used for it.


(1) Device settings
Variable name base
cisco_remote
Name of variable
cisco_remote
QOM type
TYPE_CISCO_REMOTE
Select
Parent bus
0: Bus, bus
Child buses
-1: NULL
Properties
chardev
String
serial2
Delete
eeprom_c2600_mb
String
eeprom-c2600-mb
Delete
nvram
Link
17: Device, TYPE_CISCO_NVRAM
Delete
remote_cpu_type
Integer
0x2
Delete
Add
MMIO Mappings
0:
0xf6000000
Add
Delete
PMIO Mappings
Add
Delete
Refresh
Apply
OK

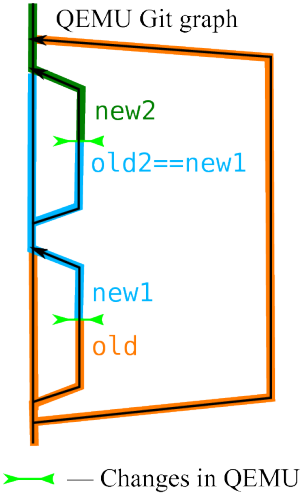
Bus interconnection example in GUI



IRQ line interconnection example in GUI



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.

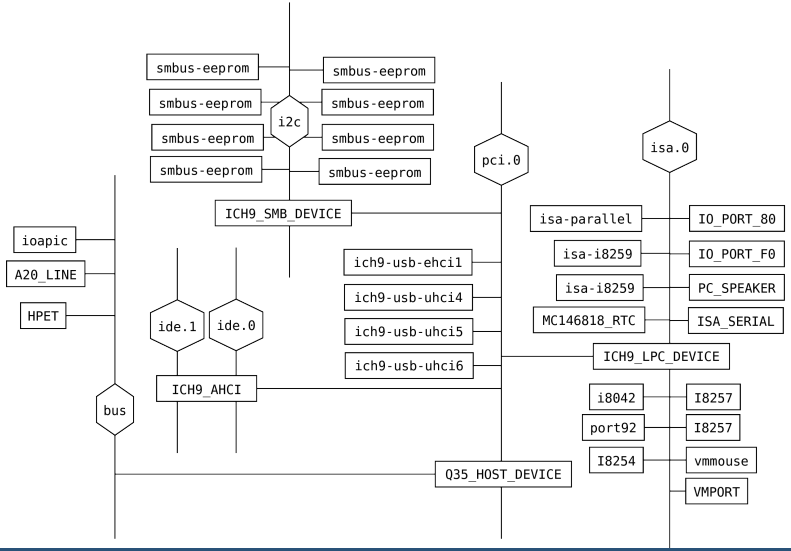
The toolset usage examples

- 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 requred devices are already present in QEMU.
- Several old devices were updated using the toolset.

Q35 machine scheme



Evaluation*

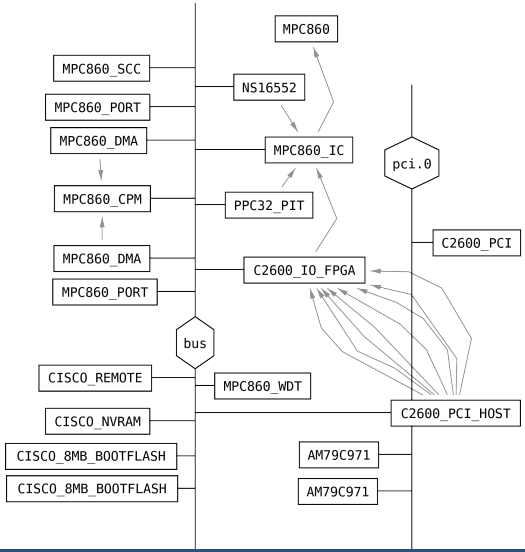
Stage	Files touched	Lines inserted	Lines deleted
Preparation**	4	42	31
Generation	8	599	0
Implementation	5	162	93***
Total	12	803	31

**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

Stage	Files touched	Lines inserted	Lines deleted
Preparation*	8	128	35
Generation	37	2186	0
Implementation	31	4747	419
Total	45	6642	35

**Memory management unit, CPU's special registers and interrupt support, PCI identifiers.*

Evaluation*

Device	Configuration size	Draft size
MPC860_IC	6	125
C2600_PCI_HOST	6	133
C2600_PCI	7	82
NS16552	7	181
C2600_IO_FPGA	8	137
CISCO_REMOTE	7	152
AM79C971	12	175

**The size is measured in lines.*

Conclusion

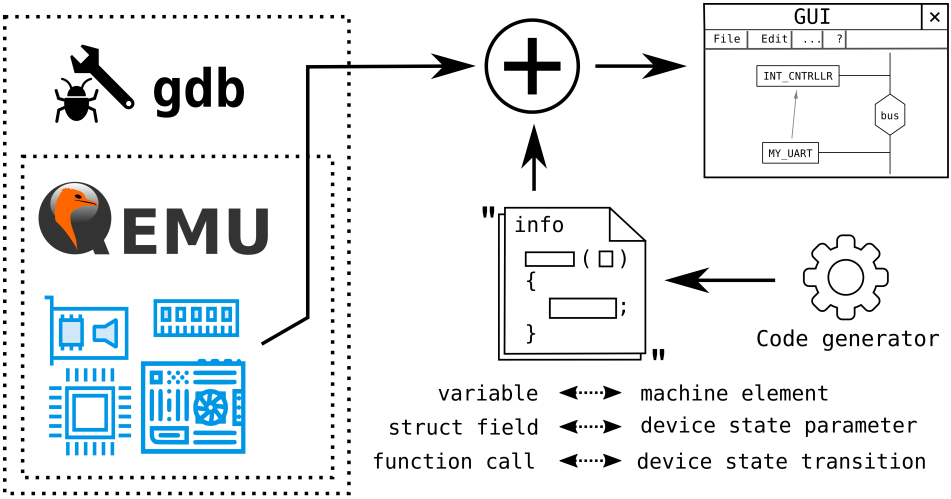
- Results
- Future work

Results

- The first stage of device and machine model development was automated using the code draft generation toolset.
- A generation configuration is wrtten 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?