Emulating RH850

for fun and vulnerability research

Damien Cauquil

Quarkslab

Agenda

- Introduction
  - Who are we ?
  - Emulation may be our savior
  - State of the Art
- Creating a RH850 basic board in QEMU
  - Buying a reference RH850 devboard
  - Adding Renesas RH850 CPU into QEMU
  - Adding RH850 core peripherals
  - Adding a UART console
  - Adding an Ethernet controller
- It opens a world of possibilities
  - Debugging RH850 with GDB
  - Adding support for RH850 in Avatar2
  - Adding support for RH850 in Unicorn
- Conclusion

Introduction

Agenda

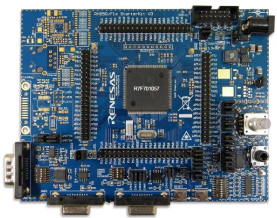
- Introduction
  - Who are we ?
  - Emulation may be our savior
  - State of the Art
- Creating a RH850 basic board in QEMU
  - Buying a reference RH850 devboard
  - Adding Renesas RH850 CPU into QEMU
  - Adding RH850 core peripherals
  - Adding a UART console
  - Adding an Ethernet controller
- It opens a world of possibilities
  - Debugging RH850 with GDB
  - Adding support for RH850 in Avatar2
  - Adding support for RH850 in Unicorn
- Conclusion

Who are we ?

- **French cyber-security company** based in Paris, France
- Specialized in **cryptography** and **embedded systems** security analysis
- We perform a lot of security tests on **automotive embedded systems**

Emulation may be our savior

- Customer provided us with a device with **no debugging capabilities**
- **RH850 architecture** is widely used but **badly supported by classic tools**
- We performed a **static analysis** and searched for **vulnerabilities**
- Would have been **faster with a dynamic analysis** !



Emulation may be our savior

Debugging

- Requires a **proprietary interface** (E1/E2 debugger)
- **Speed limited** by hardware (CPU)
- Difficult to **set the system in a specific state**

Emulation

- **No proprietary debugging interface** required
- Speed limited by the **host system**
- **Great control** over the system state
- **CAN bus** and **network interface exposed on host**

State of the Art

- **Marko Klopcic**'s RH850 support for **QEMU** [KLOPCIC] (iSYSTEM Labs, 2019)
- **MetaEmu**: An Architecture Agnostic Rehosting Framework for Automotive Firmware [METAEMU] (Zitai Chen, Sam L. Thomas, Flavio D. Garcia, 2022)
- Renesas **high-speed simulator for software development**, based on **Qemu** [RENESAS] (2023)

State of the Art

QEMU

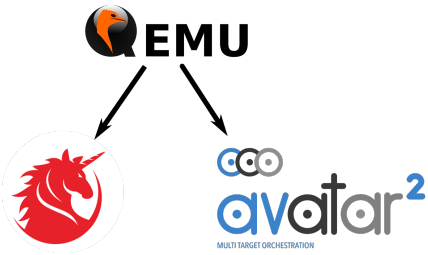
- CPU and board **emulation**
- **CAN bus** and **network interface** emulation

Unicorn

- Makes **fuzzing** possible (*afl-unicorn*)
- Allows **code instrumentation**

Avatar2

- **Hardware peripheral** implementation in **Python**
- **QEMU/GDB orchestration**



Let’s add support for RH850 into QEMU

- We initiated this project in **2021** with a former colleague, **Anthony Rullier**
- It took us **2 years** to have something working quite fine
- Once QEMU supports RH850, it will be **easy to port it into Unicorn and Avatar2** !

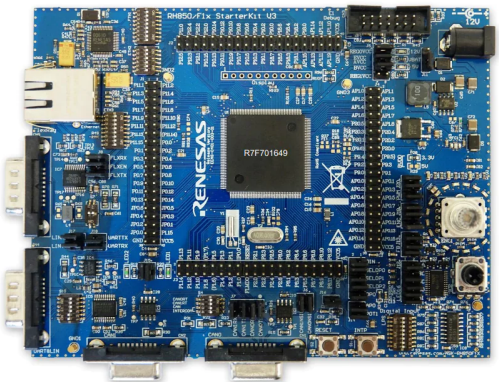
Creating a RH850 basic board in QEMU

Quarkslab

Agenda

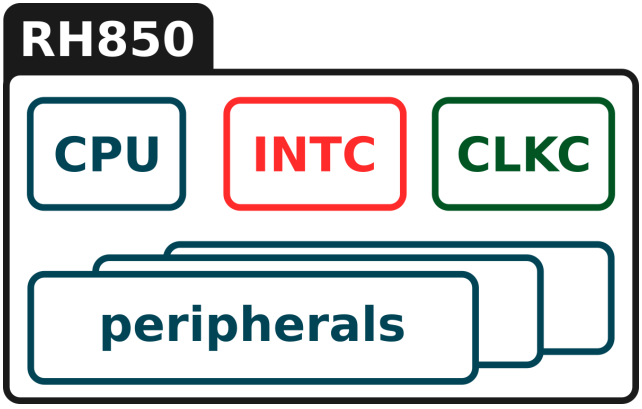
- Introduction
  - Who are we ?
  - Emulation may be our savior
  - State of the Art
- Creating a RH850 basic board in QEMU
  - Buying a reference RH850 devboard
  - Adding Renesas RH850 CPU into QEMU
  - Adding RH850 core peripherals
  - Adding a UART console
  - Adding an Ethernet controller
- It opens a world of possibilities
  - Debugging RH850 with GDB
  - Adding support for RH850 in Avatar2
  - Adding support for RH850 in Unicorn
- Conclusion

Buying a reference RH850 devboard



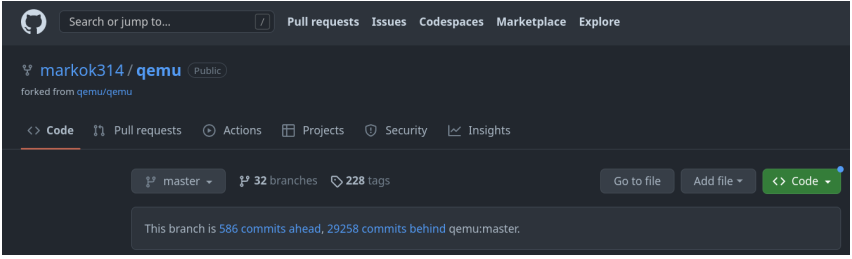
- RH850/F1KM-S4 (US$ 480)
- **Debugger** and **IDE** included
- Real-world device with **full control** !

What’s inside a RH850 MCU ?



Adding Renesas RH850 CPU into QEMU

- **Renesas RH850/F1KM** is based on a **NEC V850E3 core**
- **CPU specifications** are available on the Internet
- **Marko Klopcic's QEMU fork** is a good starting point !



QEMU Tiny Code Generator (TCG)

- QEMU **translates guest code** into native host code for **performance purpose**
- It uses **its own Intermediate Representation** (RISC-like operations)
- Splits guest code into **chained Translated Blocks** (IR code)
- **Optimizes** generated IR to speed up execution

Drawback

- Makes translated code **difficult to debug**

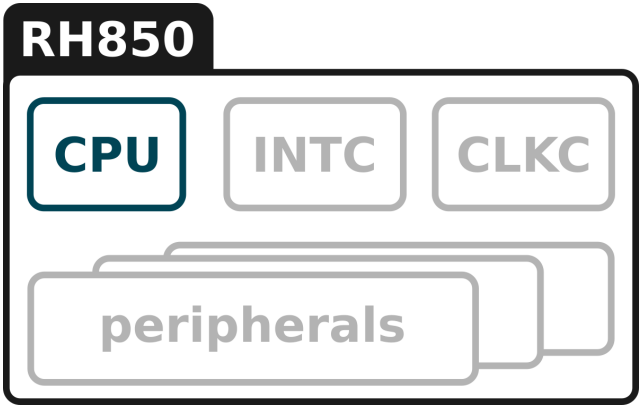
QEMU Tiny Code Generator (TCG)

```
case OPC_RH850_JARL_reg1_reg3:
{
    /* Get reg1 content into dest_addr. */
    gen_get_gpr(dest_addr, rs1);

    /* Get reg3 index, and store PC+4 in it. */
    rs3 = extract32(ctx->opcode, 27, 5);
    tcg_gen_movi_i32(link_addr, ctx->pc);
    tcg_gen_addi_i32(link_addr, link_addr, 0x4);
    gen_set_gpr(rs3, link_addr);

    /* Update pc */
    tcg_gen_andi_i32(dest_addr, dest_addr, 0xfffffffe);
    tcg_gen_mov_i32(cpu_pc, dest_addr);

    /* Goto corresponding TB (indirect jump). */
    ctx->base.is_jmp = DISAS_INDIRECT_JUMP;
}
break;
```

Improving Marko Klopcic's implementation

- We **added support for FPU** instructions
- We also **fixed several bugs** in this implementation
- We completely **reworked the interrupt/exception internals**
- We made this CPU compliant with the **QEMU Object Model** (QOM)

We now have a working CPU !



Adding RH850 core peripherals

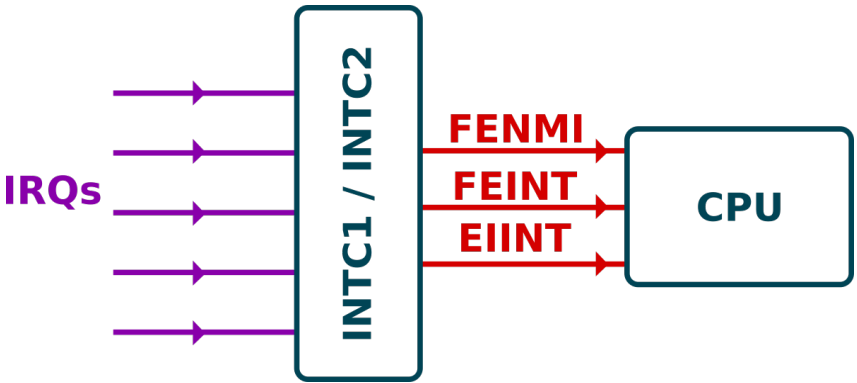
Interrupt Controllers (INTC1 & INTC2)

- Manage **Interrupt Requests (IRQs)** and how they are dispatched
- Also handle *non-maskable* interrupts (**NMI**)

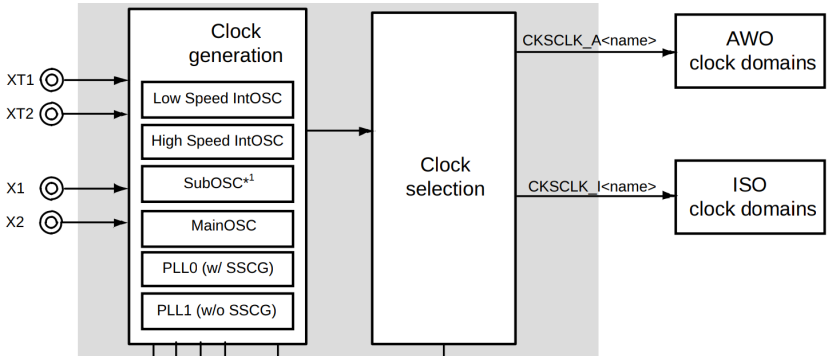
Clock controller

- clock configuration (PLL, external clock, etc.)
- **Clocks configuration** is one of the first things done at runtime

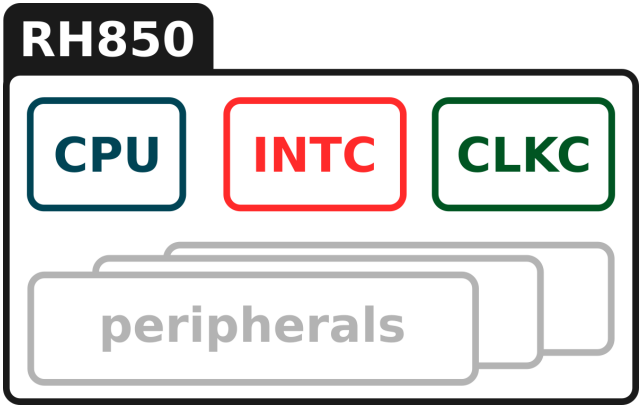
Interrupt Controllers



Clock controller



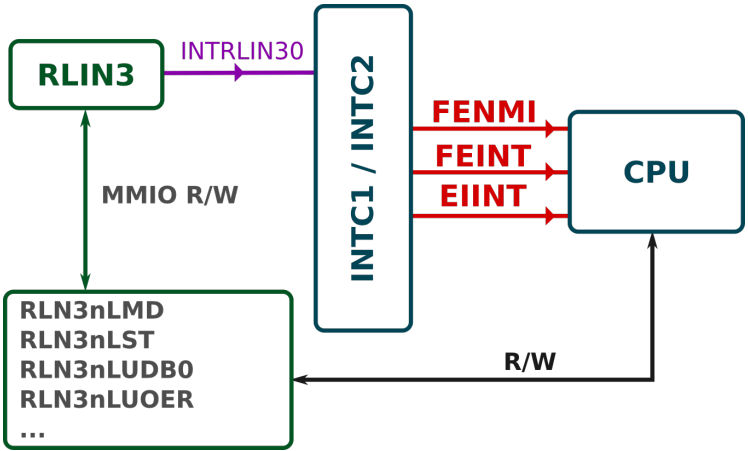
We have all the core components !



Adding a UART console

- **UART** console is the **default communication mean** with an embedded system
- Having a working UART console **will prove that our CPU, interrupt and clock controllers are OK**
- It could also be useful to **generate execution traces** !

Adding a UART console



UART console is working fine !

```
$ qemu-system-rh850 -nographic -machine rh850-f1kms4 -kernel F1KM-S4_StarterKit.abs
UART Init OK
PORWUF: RESET
****************************************
*                RENESAS                *
*      RH850/F1x Starter Kit Demo!      *
****************************************

Begin with self test...
```

Adding an Ethernet controller

Ethernet AVB controller

- 10/100 Mbps **full-duplex** transfer
- **Credit-based shaping** (CBS) support
- **AVB Transport Protocol** (IEEE 1722) support
- Multiple FIFOs

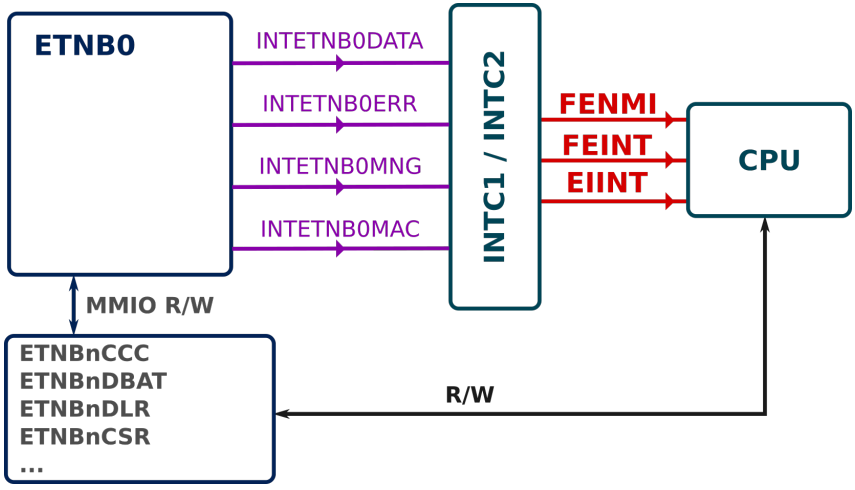
Adding an Ethernet controller

Ethernet AVB controller

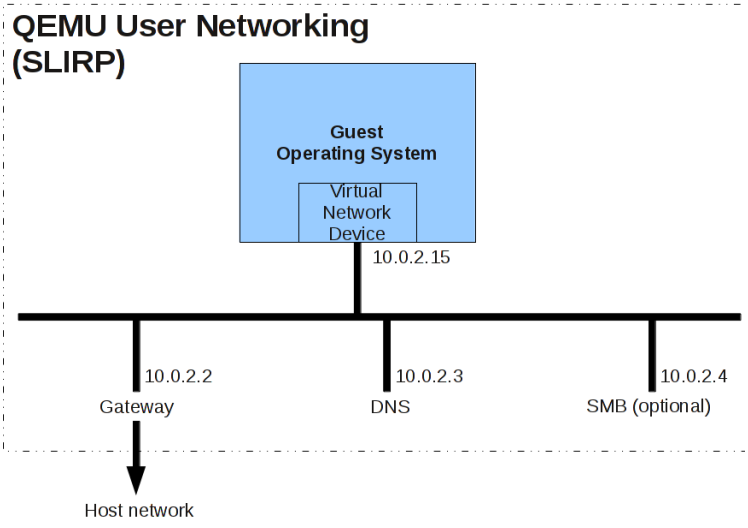
- 10/100 Mbps **full-duplex** transfer
- ~~**Credit-based shaping** (CBS) support~~
- ~~**AVB Transport Protocol** (IEEE 1722) support~~
- ~~Multiple FIFOs~~ **Single FIFO**

We decided to implement the minimum viable ethernet controller.

Ethernet AVB Controller



User-mode networking in QEMU



User-mode networking in QEMU

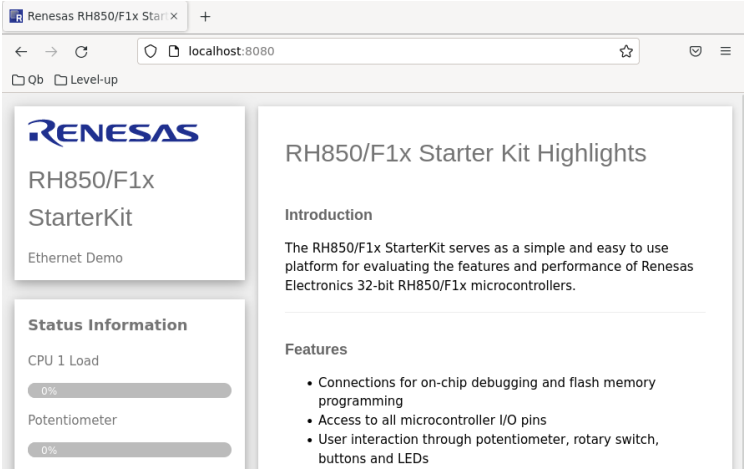
Pros

- **Default networking backend** in QEMU
- **Easy** to use, configure and implement
- **Port forwarding** between host and guest
- No root privileges required

Cons

- **No deep control** of network interfaces
- **NAT** network

Minimal ethernet controller is OK



Still a work in progress ...

- **CAN controller** not yet supported
- Ethernet controller is **minimal**
- **F1KM-S4** board is the **only version supported** for now

It opens a world of possibilities

Agenda

- Introduction
  - Who are we ?
  - Emulation may be our savior
  - State of the Art
- Creating a RH850 basic board in QEMU
  - Buying a reference RH850 devboard
  - Adding Renesas RH850 CPU into QEMU
  - Adding RH850 core peripherals
  - Adding a UART console
  - Adding an Ethernet controller
- It opens a world of possibilities
  - Debugging RH850 with GDB
  - Adding support for RH850 in Avatar2
  - Adding support for RH850 in Unicorn
- Conclusion

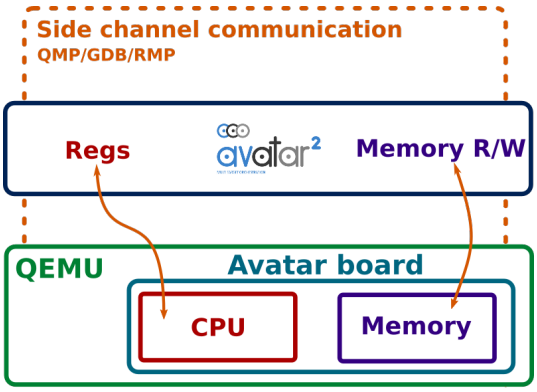
Emulation helps a lot

- QEMU can expose a **GDB server** that allows **remote debugging**
- **Avatar2** provides **hardware peripheral emulation** in Python
- **Unicorn** allows **code instrumentation** in Python (and other languages)

Debugging RH850 with GDB

- **Debugging** real hardware is sometimes **impossible** (debugging interface disabled)
- Emulation offers a way to **inspect registers and memory at any time**
- No need of real hardware when emulation works fine (**more free space** on bench) !

Adding support of RH850 in Avatar2



- **Avatar2** instruments a custom QEMU (using the `avatar` board)
- Since it's QEMU, we made this `avatar` board compatible with our RH850 CPU
- Provides a **convenient way to instrument** a RH850 VM in Python

Avatar2

```python
def test_add_nocarry_format1(self):
    '''Test Add instruction (format I)'''

    # Pick two numbers below 4096 and add them
    r1 = randint(0, 4096)
    r2 = randint(0, 4096)
    self.vm.wreg('r1', r1)
    self.vm.wreg('r2', r2)

    # execute add instruction
    self.vm.execute('add r1, r2')

    # assert result
    self.assertEqual(self.vm.rreg('r2'), r1+r2)
```

- We implemented and tested our RH850 CPU in Avatar2's QEMU fork
- Allowed us to implement **a set of 100+ unit tests** to check instructions implementation

Adding support for RH850 in Unicorn

- **Unicorn** is also based on a **custom version of QEMU**
- It provides a **CPU emulator with bindings** in various languages (Python, Ruby, Java, .NET, etc.)
- Used by **afl-unicorn** to fuzz any piece of binary !
- We ported our CPU into **Unicorn** and it **works like a charm** !

Adding support for RH850 in Unicorn

```
                        *************************************************************
                        *                          FUNCTION                         *
                        *************************************************************
                        size_t __stdcall _strlen(char * __s)
      size_t              r10:4          <RETURN>
      char *              r6:4           __s
                        _strlen                                    XREF[8]:     Entry Point(*),
                                                                                _strnstr.1:00018e72(c),
                                                                                _get_tag_insert.1:0001914e(c),
                                                                                _get_tag_insert.1:00019162(c),
                                                                                _get_tag_insert.1:000191a0(c),
                                                                                _dhcp_option_hostname.1:0001b8c4.
                                                                                _pbuf_strstr:0001cee2(c),
                                                                                __REL_stoa_r:000296ba(c)
0002876e 1f 52          mov         -0x1,r10

                        LAB_00028770                               XREF[1]:     0002877a(j)
00028770 41 52          add         0x1,r10
00028772 06 5f 00 00    ld.b        0x0[__s],r11
00028776 41 32          add         0x1,__s
00028778 60 5a          cmp         0x0,r11
0002877a ba fd          bne         LAB_00028770
0002877c 7f 00          jmp         [lp]
```

Adding support for RH850 in Unicorn

```c
int main(int argc, char **argv, char **envp)
{
    [...]
    /* Initialize emulator in little endian mode */
    uc_open(UC_ARCH_RH850, UC_MODE_LITTLE_ENDIAN, &uc);

    /* Map memory and write code and the provided string. */
    uc_mem_map(uc, CODE_ADDRESS, 1024 * 1024, UC_PROT_ALL);
    uc_mem_write(uc, CODE_ADDRESS, RH850_STRLEN_CODE, sizeof(RH850_STRLEN_CODE) - 1);
    uc_mem_write(uc, RAM_ADDRESS, argv[1], strlen(argv[1]));

    /* Initialize machine registers (r6 -> pointer to our text string). */
    uc_reg_write(uc, UC_RH850_REG_R6, &r6);

    /* Emulate machine code (strlen) and show result. */
    uc_emu_start(uc, CODE_ADDRESS, CODE_ADDRESS + sizeof(RH850_STRLEN_CODE) - 1, 0, 0);
    uc_reg_read(uc, UC_RH850_REG_R10, &r10);
    printf("Text length: %d\r\n", r10);

    [...]
}
```

Adding support for RH850 in Unicorn

```
virtualabs@virtubox:~$ ./rh850-strlen
[!] Usage: ./rh850-strlen <string>
virtualabs@virtubox:~$ ./rh850-strlen Test
Text length: 4
virtualabs@virtubox:~$ ./rh850-strlen "Moar text"
Text length: 9
```

Adding support for RH850 in Unicorn

- This implementation **needs more tests**
- `afl-unicorn` has **not been tested yet**
- **Helped us a lot** during one of our latest assessment !

Conclusion

Conclusion

- We implemented a **basic RH850 board in QEMU**
- **Unicorn** and **Avatar2** rely on a **modded version of QEMU**, so we **ported the CPU** into these projects
- We are able to **emulate a firmware with UART and network support** (CAN interface is a WIP, will be supported soon)
- We can also **instrument any RH850 code** in order to perform a **code coverage analysis** or implement a **naive fuzzer**
- **Tested recently** on a customer ECU firmware for code coverage and **specific functions emulation**

Code release

- QEMU RH850 board still **lacks some critical peripherals** and ethernet-related features
- We plan to **push our work into QEMU/Unicorn/Avatar2**, but it still needs **a lot of code rework/cleanup**
- **Stay tuned !**

Thank you

Contact information:

Email: dcauquil@quarkslab.com

Masto: @virtualabs@mamot.fr

Website: https://www.quarkslab.com

Quarkslab

References

KLOPCIC	https://github.com/markok314/qemu
METAEMU	https://arxiv.org/pdf/2208.03528.pdf
RENESAS	https://www.renesas.com/us/en/software-tool/high-speed-simulator-software-development
QEMU	https://www.qemu.org/
UNICORN	https://github.com/unicorn-engine/unicorn
AVATAR2	https://github.com/avatartwo/avatar2