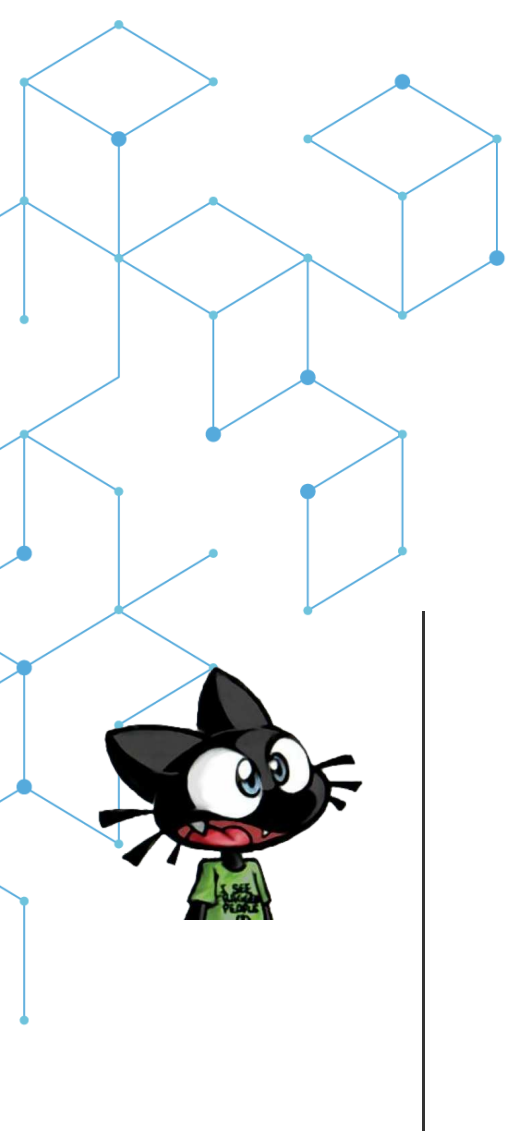




# Tame the (Q)emu !

**(Domptez l'émeu !)**

Damien Cauquil |  @virtualabs@mamot.fr

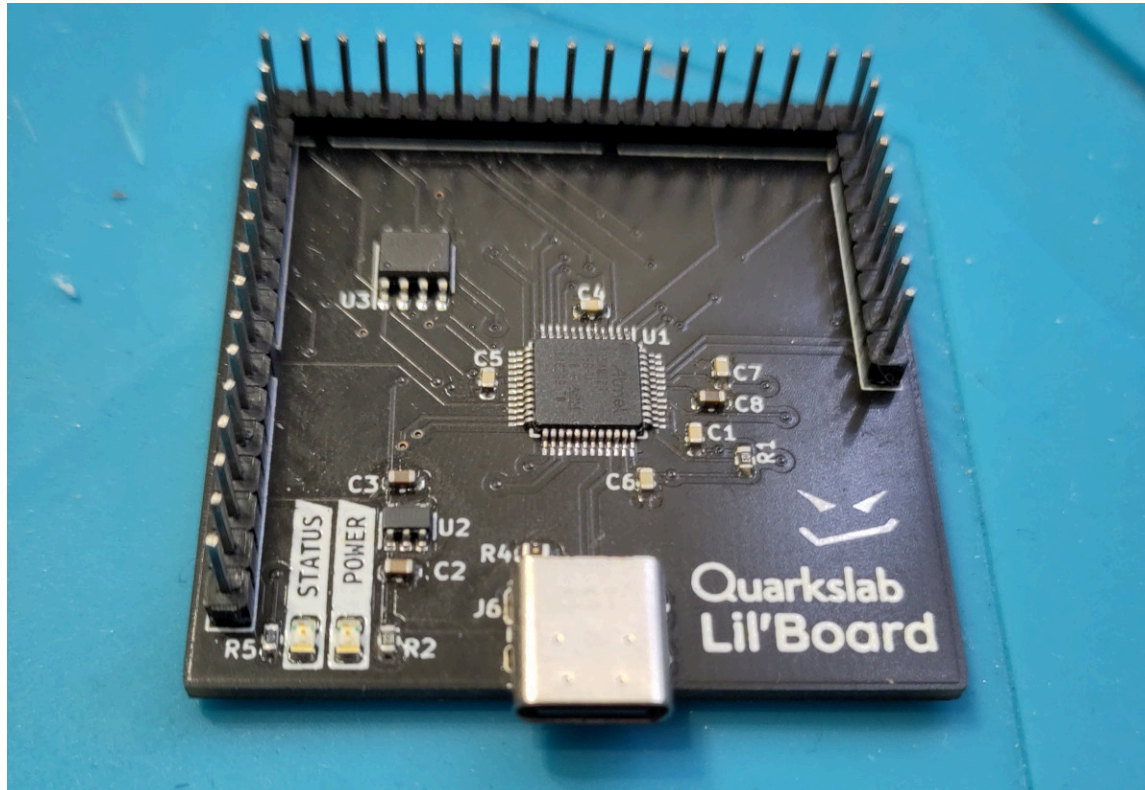


# Who am I ?

- R&D Engineer @ Quarkslab
- Hardware & software RE
- I can (and will) speak fast 😄

# Introduction

# It all started with this board



Badge Reverse ESIEA

# Quarkslab Lil'Board

- Based on an ~~ATMEL~~ Microchip **SAMD21 MCU**
- Embeds an **SPI Flash** chip
- Exposes a **UART** interface

# Firmware emulation

- First implementation with **Unicorn**:
  - no UART interface available
  - quite slow
  - No GDB debugging capability
- We need something **more robust** !

# Off-the-shelf tools

# Off-the-shelf tools

- **QEMU**: well-known++ machine emulator



# Off-the-shelf tools

- **QEMU**: well-known++ machine emulator
- **Renode**: QEMU-based emulation framework

# Off-the-shelf tools

- **QEMU**: well-known++ machine emulator
- **Renode**: QEMU-based emulation framework
- **Qiling**: Unicorn-based emulation framework

# Off-the-shelf tools

- **QEMU**: well-known++ machine emulator
- **Renode**: QEMU-based emulation framework
- **Qiling**: Unicorn-based emulation framework
- **Avatar2**: QEMU-based firmware emulation framework

# Off-the-shelf tools

- **QEMU**: well-known++ machine emulator
- **Renode**: QEMU-based emulation framework
- **Qiling**: Unicorn-based emulation framework
- **Avatar2**: QEMU-based firmware emulation framework
- **Firmadyne**: QEMU-based firmware emulation tools

# QEMU

- Off-the-shelf **CPUs**, including ARM Cortex-M0
- Off-the-shelf **SPI Flash** emulation
- Integrated **GDB server**
- **UART interface** can be exposed through **QEMU console**

# QEMU



# QEMU ?



# Let's dive into QEMU !

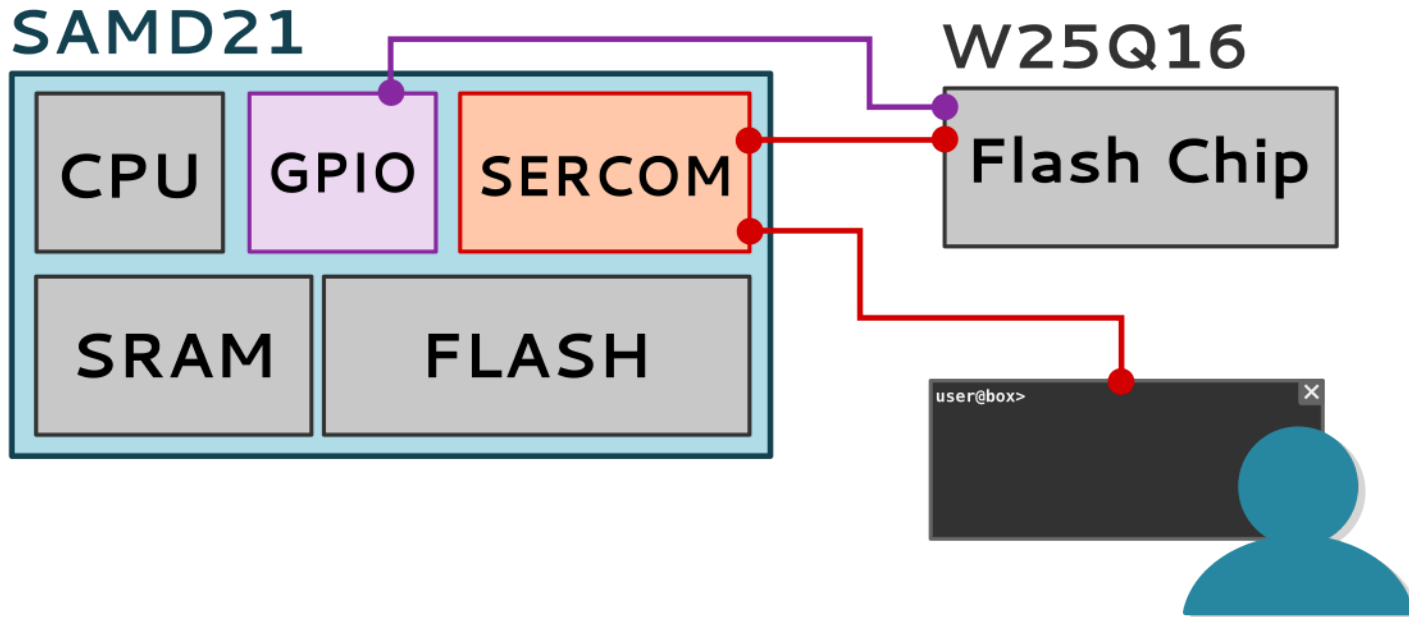




# Emulating our custom board with QEMU



# Board hardware overview



# SAMD21 specifications

## 10.2 Physical Memory Map

The High-Speed bus is implemented as a bus matrix. All High-Speed bus addresses are fixed, and they are never remapped in any way, even during boot. The 32-bit physical address space is mapped as follow:

**Table 10-1. SAM D21 Physical Memory Map<sup>(1)</sup>**

Memory	Start address	Size					
		SAMD21x18	SAMD21x17	SAMD21x16	SAMD21x15	SAMD21x16L	SAMD21x15L
Internal Flash	0x00000000	256 Kbytes	128 Kbytes	64 Kbytes	32 Kbytes	64 Kbytes	32 Kbytes
Internal RWW section <sup>(2)</sup>	0x00400000	-	4 Kbytes	2 Kbytes	1 Kbytes	2 Kbytes	1 Kbytes
Internal SRAM	0x20000000	32 Kbytes	16 Kbytes	8 Kbytes	4 Kbytes	8 Kbytes	4 Kbytes

## 11.1 Cortex M0+ Processor

The SAM D21 implements the **ARM® Cortex®-M0+** processor, based on the **ARMv6** Architecture and Thumb®-2 ISA. The Cortex M0+ is 100% instruction set compatible with its predecessor, the Cortex-M0 core, and upward compatible to Cortex-M3 and M4 cores. The ARM Cortex-M0+ implemented is revision r0p1. For more information refer to <http://www.arm.com>.

# Defining our SAMD21 MCU

```
static const TypeInfo samd21_info = {
    .name          = TYPE_SAMD21_MCU,
    .parent        = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(SAMD21State),
    .instance_init = samd21_init,
    .class_init    = samd21_class_init,
};
```

**QEMU Object Model (QOM) rulz !**

# Adding our Cortex-M0 CPU

```
/* Create our CPU. */  
object_initialize_child(OBJECT(s), "armv6m",  
    &s->cpu, TYPE_ARMV7M);  
qdev_prop_set_string(DEVICE(&s->cpu), "cpu-type",  
    ARM_CPU_TYPE_NAME("cortex-m0"));  
qdev_prop_set_uint32(DEVICE(&s->cpu), "num-irq", 32);
```

# ... and some Flash memory ...

```
/* Create the Flash memory ROM region. */
memory_region_init_rom(&s->flash, OBJECT(s), "samd21.flash",
                      SAMD21_X18_FLASH_SIZE, &err);
if (err) {
    error_propagate(errp, err);
    return;
}

/* Map the Flash memory. */
memory_region_add_subregion_overlap(
    &s->container,
    SAMD21_FLASH_BASE,
    &s->flash,
    -1
);
```

# ... and some RAM

```
/* Initialize SRAM memory region. */
memory_region_init_ram(&s->sram, OBJECT(s), "samd21.sram",
                      s->sram_size, &err);

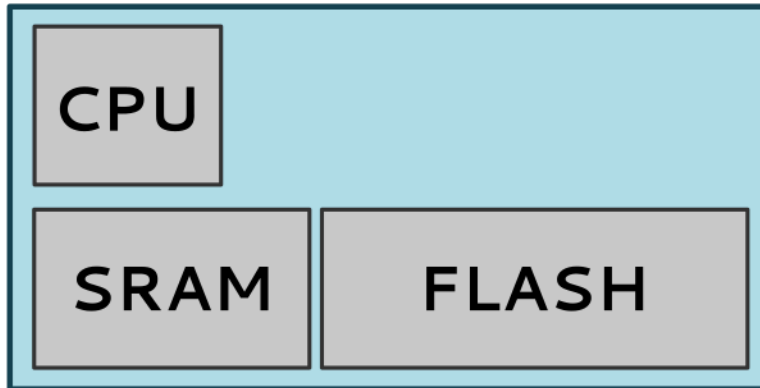
if (err) {
    error_propagate(errp, err);
    return;
}

/* Map the SRAM memory region. */
memory_region_add_subregion(
    &s->container,
    SAMD21_SRAM_BASE,
    &s->sram
);
```



# Habemus MCU !

SAMD21



# Adding peripherals

- Hardware peripherals expose **MMIO registers**
- We must handle **read/write** operations
- **Peripheral state** is updated accordingly

# Hardware peripheral MMIO registers

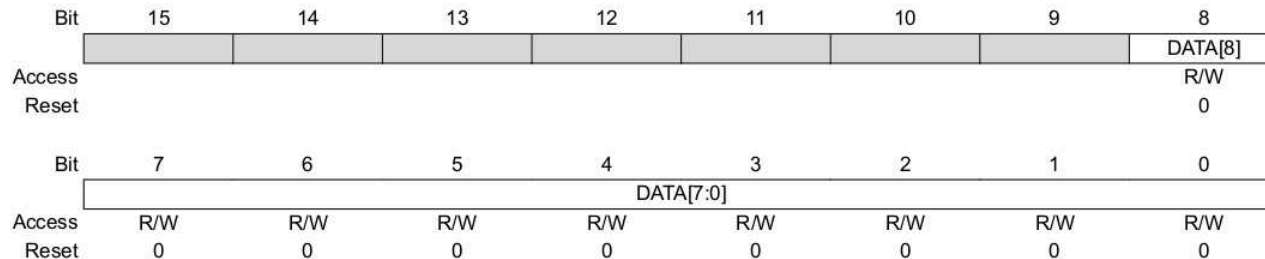
## 26.7 Register Summary

Offset	Name	Bit Pos.								
0x00	CTRLA	7:0	RUNSTDBY				MODE[2:0]		ENABLE	SWRST
		15:8	SAMPR[2:0]							IBON
		23:16	SAMPB[1:0]		RXPO[1:0]				TXPO[1:0]	
		31:24		DORD	CPOL	CMODE		FORM[3:0]		
0x04	CTRLB	7:0		SBMODE				CHSIZE[2:0]		
		15:8			PMODE			ENC	SFDE	COLDEN
		23:16							RXEN	TXEN
		31:24								
0x08 ... 0x0B	Reserved									
0x0C	BAUD	7:0	BAUD[7:0]							
		15:8	BAUD[15:8]							
0x0E	RXPL	7:0	RXPL[7:0]							
0x0F ... 0x13	Reserved									
0x14	INTENCLR	7:0	ERROR		RXBRK	CTSIC	RXS	RXC	TXC	DRE
0x15	Reserved									
0x16	INTENSET	7:0	ERROR		RXBRK	CTSIC	RXS	RXC	TXC	DRE
0x17	Reserved									
0x18	INTFLAG	7:0	ERROR		RXBRK	CTSIC	RXS	RXC	TXC	DRE
0x19	Reserved									
0x1A	STATUS	7:0		TXE	COLL	ISF	CTS	BUFOVF	FERR	PERR
		15:8								

# Hardware peripheral MMIO registers

## 26.8.10 Data

**Name:** DATA  
**Offset:** 0x28  
**Reset:** 0x0000  
**Property:** -



### Bits 8:0 – DATA[8:0] Data

Reading these bits will return the contents of the Receive Data register. The register should be read only when the Receive Complete Interrupt Flag bit in the Interrupt Flag Status and Clear register (INTFLAG.RXC) is set. The status bits in STATUS should be read before reading the DATA value in order to get any corresponding error.

Writing these bits will write the Transmit Data register. This register should be written only when the Data Register Empty Interrupt Flag bit in the Interrupt Flag Status and Clear register (INTFLAG.DRE) is set.

# SERCOM support

```
static const TypeInfo samd21_sercom_info = {  
    .name = TYPE_SAMD21_SERCOM,  
    .parent = TYPE_SYS_BUS_DEVICE,  
    .instance_size = sizeof(SAMD21SERCOMState),  
    .instance_init = samd21_sercom_init,  
    .class_init = samd21_sercom_class_init  
};
```

# SERCOM MMIO R/W

```
static const MemoryRegionOps sercom_ops = {
    .read = sercom_read,
    .write = sercom_write,
    .endianness = DEVICE_LITTLE_ENDIAN,
    .impl.min_access_size = 4,
    .impl.max_access_size = 4
};
```

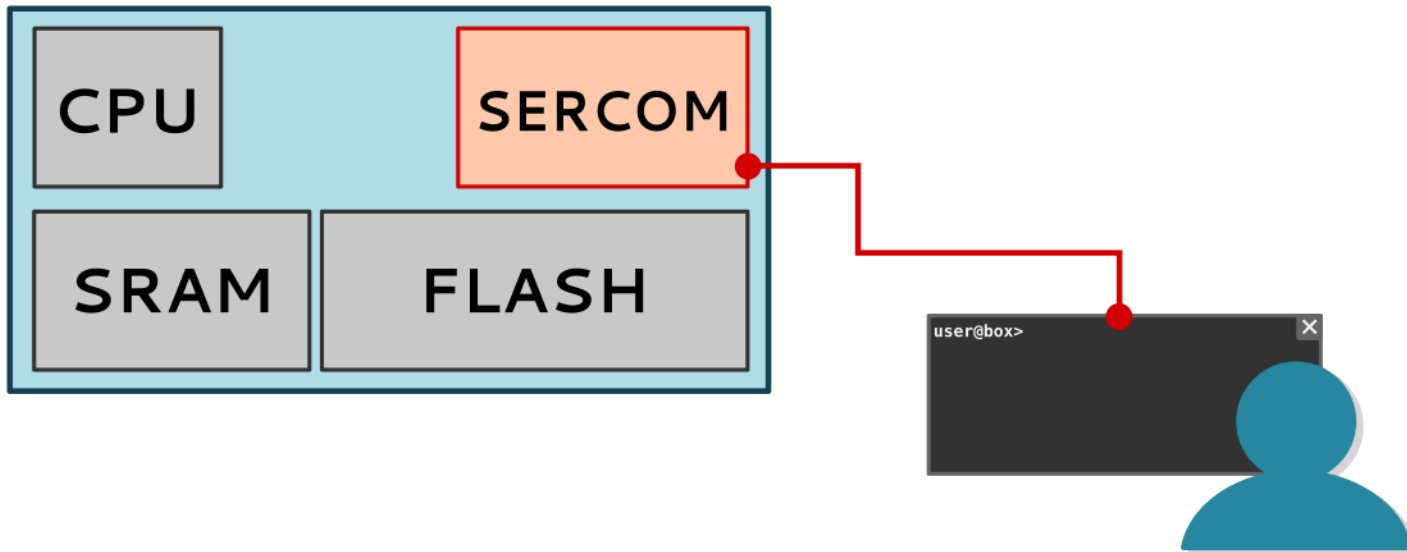
# Update state on R/W

```
case A_SERCOM_DATA:
{
    switch (s->current_mode)
    {
        case SERCOM_USART:
        {
            /* Write register. */
            s->reg16[R_SERCOM_DATA] = (value & 0xffff);

            /* Mark byte as sent. */
            s->pending_tx_byte = true;
            uart_transmit(NULL, G_IO_OUT, s);
        }
        break;
    }
}
```

# We now have USART and SPI

SAMD21

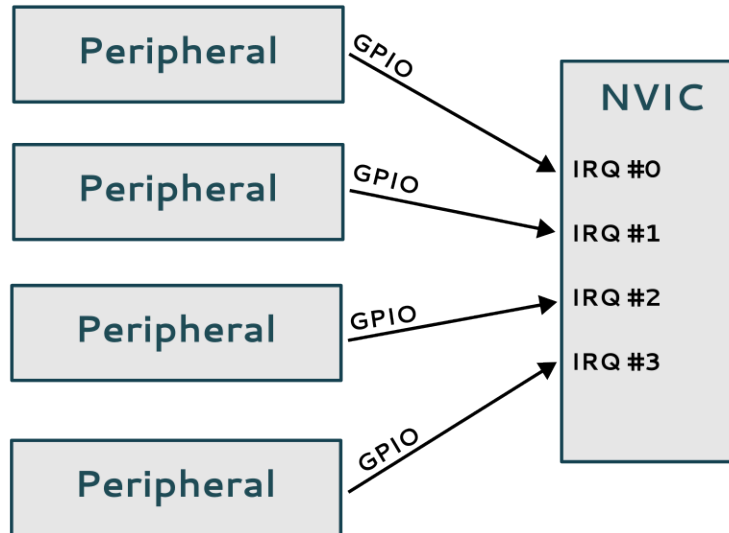




# Adding GPIOs

- Required for **external communication**
- Our Flash chip needs a GPIO for its Chip Select line

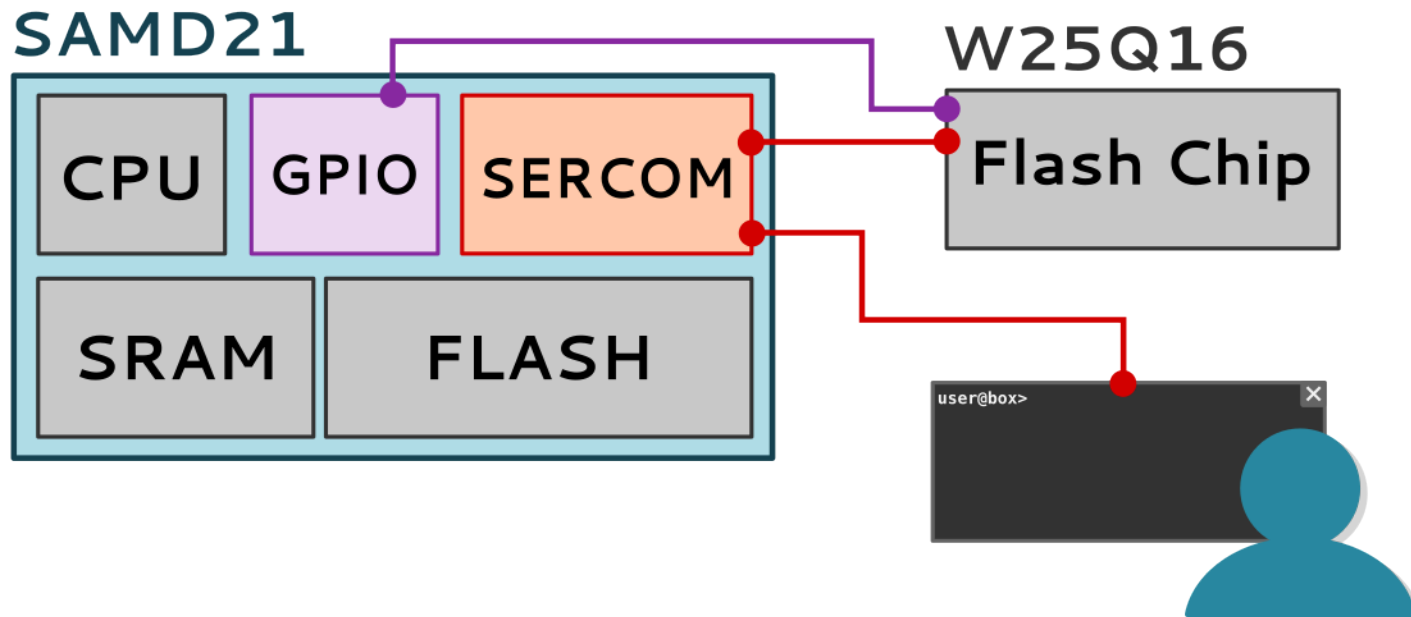
# QEMU, GPIOs and IRQs



# GPIO support

```
static const TypeInfo samd21_port_info = {  
    .name = TYPE_SAMD21_PORT,  
    .parent = TYPE_SYS_BUS_DEVICE,  
    .instance_size = sizeof(SAMD21GPIOState),  
    .instance_init = samd21_port_init,  
    .class_init = samd21_port_class_init  
};
```

# We have GPIOs !



# Demo: Emulation

Fichier Édition Affichage Rechercher Terminal Aide

```
virtualabs@virtubox:~/demo$
```

# Demo: Debug



Fichier Édition Affichage Rechercher Terminal Aide  
virtualabs@virtubox: ~/demo\$



Fichier Édition Affichage Rechercher Terminal Aide  
virtualabs@virtubox: ~\$

# Code & documentation



<https://github.com/quarkslab/sstic-tame-the-qemu>

# Conclusion



# Takeaways

- QEMU provides a lot of **basic building blocks**, like **Lego bricks**
- **Add missing bricks if needed**, and you can build your own custom board !
- Full-featured emulation: GDB debugging, UART interaction
- **QEMU design and performances are just awesome !**




# However ...

- You need to **know C** in order to add your own bricks to QEMU
- If not familiar with QEMU, **adding a custom board can require days of dev**
- The process to push your board upstream is complex

**Special thanks to Philippe Teuwen, Eloïse Brocas, Maxime Rossi Bellom, Ryad Benadjila for their advices and guidance 🙏**

# Thanks ! Questions ?

**Damien  
Cauquil**

 dcauquil@quarkslab.com  
 @virtualabs@mamot.fr  
 quarkslab