Projektarbeit CXL

Julius Armbrüster und Moritz Riefer

Emulation mit QEMU

CXL kann mit dem Programm `qemu-system-x86_64` emuliert werden. Hierzu muss man die CXL Struktur definieren dies geschieht per Kommandozeilenargumente. Eine Beispielstrukture mit einem CXL Type 3 Device und Persistent Memory sieht wie folgt aus:

```
qemu-system-x86_64 \
  -machine q35,cxl=on \
  -m 2048 \
  -smp 2 \
  -cpu max \
  -drive file=debian-x86.img,format=qcow2 \
  -boot d \
  -net nic -net user,hostfwd=tcp::10022-:22 \
  -object memory-backend-file,id=cxl-mem0,share=on,mem-path=cxl-nvdimm0,size=256M \
  -object memory-backend-file,id=cxl-lsa0,share=on,mem-path=cxl-lsa0,size=256M \
  -device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.0 \
  -device cxl-rp,port=0,bus=cxl.0,id=cxl-rp0,chassis=0,slot=2 \
  -device cxl-type3,bus=cxl-rp0,persistent-memdev=cxl-mem0,lsa=cxl-lsa0,id=cxl-pmem0 \
  -M cxl-fmw.0.targets.0=cxl.0,cxl-fmw.0.size=256M
```

D3OS

In Rust haben wir schon verschiedene Memory-Mapped-Register ausgelesen, die sich auf CXL beziehen. (TODO: Hier Code einfügen und/oder Repo referenzieren)

Probleme unter Linux

Unter Linux taucht in der `cxl`-Utility die korrekte Struktur auf.

```
$ cxl list
[
  {
    "memdev":"mem0",
    "pmem_size":268435456,
    "serial":0,
    "host":"0000:0d:00.0",
    "firmware_version":"BWFW VERSION 00"
  }
]
```

Leider lässt sich mit QEMU 10.1.0 keine CXL Region erstellen. Als Host haben wir sowohl Apple-Silicon mit macOS als auch x86_64 Debian 13 und Windows 11 ausprobiert. Das Guest-OS war Debian 13 mit Kernel version 6.15.4-1 verwendet.

```
$ sudo cxl create-region -d decoder0.0 -s 256M -m mem0
cxl region: create_region: region0: failed to commit decode: No such device or address
cxl region: cmd_create_region: created 0 regions
```

So läuft CXL unter QEMU in Linux

Um QEMU unter Linux ans Laufen zu bekommen haben wir uns an folgender Anleitung orientiert: https://github.com/moking/moking.github.io/wiki/Basic:-CXL-Test-with-CXL-emulation-in-QEMU

Mit dieser Anleitung hat das Emulieren eines CXL Type 3 Geräts in QEMU erfolreich funktioniert. Hier für wird sowohl gepachtes QEMU als auch ein gepachter Kernel kompiliert.

QEMU Fork von moking klonen und die Version dcd-v6 kompilieren

```
$ git clone https://github.com/moking/qemu
$ git switch dcd-v6
$ ./configure --target-list=x86\_64-softmmu --enable-debug
$ make -j 16
```

Linux Kernel kernel mit dcd support von weiny2 klonen und für das Kompilieren konfigurieren.

```
$ git clone https://github.com/weiny2/linux-kernel
$ git switch dcd-2024-03-24
$ make menuconfig
```

Folgende config Einstellungen müssen aktiviert sein:

```
CONFIG_ARCH_WANT_OPTIMIZE_DAX_VMEMMAP=y
CONFIG_CXL_BUS=m
CONFIG_CXL_PCI=m
CONFIG_CXL_MEM_RAW_COMMANDS=y
CONFIG_CXL_ACPI=m
CONFIG_CXL_PMEM=m
CONFIG_CXL_MEM=m
CONFIG_CXL_PORT=m
CONFIG_CXL_SUSPEND=y
CONFIG_CXL_REGION=y
CONFIG_CXL_REGION_INVALIDATION_TEST=y
CONFIG_CXL_PMU=m
CONFIG_ND_CLAIM=y
CONFIG_ND_BTT=m
CONFIG_ND_PFN=m
CONFIG_NVDIMM_DAX=y
```

```
CONFIG_DAX=m
CONFIG_DEV_DAX=m
CONFIG_DEV_DAX_PMEM=m
CONFIG_DEV_DAX_HMEM=m
CONFIG_DEV_DAX_CXL=m
CONFIG_DEV_DAX_HMEM_DEVICES=y
CONFIG_DEV_DAX_KMEM=m
```

Kernel kompilieren und installieren. Das kernel image liegt anschließend am Pfad: `arch/x86/boot/bzImage`

```
$ make -j 16
$ sudo make modules\_install
```

Mit `$QEMUDIR` und `$LINUXDIR` meinen wir im folgenden immer den Repository Ordner für QEMU bzw. Linux.

Um den Kernel zu starten verwenden wir den eingebauten Bootloader von QEMU. Hierzu erstellen wir ein Image `cxl-img` mit Debian und mounten es in den Ordner `cxl-dir`.

```
$ $QEMUDIR/build/qemu-img create cxl-img 16G
$ sudo mkfs.ext4 cxl-img
$ sudo mount -o loop cxl-img cxl-dir
$ sudo debootstrap --arch amd64 stable cxl-dir
```

Nun kann man in `cxl-dir` chrooten und weitere Einstellungen vornehmen, wie einen User mit root-Rechten erstellen und einige Packages installieren.

```
$ sudo chroot cxl-dir /bin/bash
$ useradd -m $USER
$ usermod -aG sudo $USER
$ apt install kmod pciutils cxl ndctl sudo daxctl
```

Mit STRG-D kann man aus der chroot-Shell zurückkehren und `cxl-dir` unmounten.

```
$ sudo umout \$DIR
```

Der folgende Command startet QEMU mit dem kompilierten Linux Kernel und einer CXL Struktur mit einem Memory-Device. Außerdem wird dyndbg für einige CXL module aktiviert.

```
$ $QEMUDIR/build/qemu-system-x86_64 -s
    -kernel $LINUXDIR/arch/x86/boot/bzImage
    -append "root=/dev/sda rw console=ttyS0,115200 ignore_loglevel nokaslr \
    cxl_acpi.dyndbg=+fplm cxl_pci.dyndbg=+fplm cxl_core.dyndbg=+fplm \
    cxl_mem.dyndbg=+fplm cxl_pmem.dyndbg=+fplm cxl_port.dyndbg=+fplm \
    cxl_region.dyndbg=+fplm cxl_test.dyndbg=+fplm cxl_mock.dyndbg=+fplm \
    cxl_mock_mem.dyndbg=+fplm dax.dyndbg=+fplm \
    dax_cxl.dyndbg=+fplm device_dax.dyndbg=+fplm" \
```

```
-smp 4 -serial mon:stdio -nographic -qmp tcp:localhost:4444,server,wait=off \
-netdev user,id=network0,hostfwd=tcp::2024-:22 -device e1000,netdev=network0 \
-monitor telnet:127.0.0.1:12345,server,nowait \
-drive file=cxl-dir,index=0,media=disk,format=raw \
-machine q35,cxl=on -m 4G \
-virtfs local,path=./lib/modules,mount_tag=modshare,security_model=mapped \
-virtfs local,path=/home/$USER,mount_tag=homeshare,security_model=mapped \
-object memory-backend-file,id=cxl-mem1,share=on,mem-path=/tmp/cxltest.raw,size=512M \
-object memory-backend-file,id=cxl-lsa1,share=on,mem-path=/tmp/lsa.raw,size=512M \
-device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 \
-device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 \
-device cxl-type3,bus=root_port13,memdev=cxl-mem1,lsa=cxl-lsa1,id=cxl-pmem0 \
-M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=4G,cxl-fmw.0.interleave-granularity=8k
```

Nun kann man sich mit dem erstellten User einloggen. Nach jedem Start müssen die cxl Module geladen werden:

```
$ sudo modprobe -a cxl\_acpi cxl\_core cxl\_pci cxl\_port cxl\_mem cxl\_pmem
```

Um den Speicher auf dem CXL Memory-Device nutzen zu können muss man zu nächste eine Region und einen Namespace für diese Region erstellen. Anschließend kann man den Namespace dem Ram hinzufügen. Der Speicher sollte dann in `lsmem` zu sehen sein.

```
$ sudo cxl create-region -m -d decoder0.0 -w 1 mem0 -s 512M
$ sudo ndctl create-namespace -m dax -r region0
$ sudo daxctl reconfigure-device --mode=system-ram --no-on
```

Zusätzliche Informationen:

- Namespace kann auch als fsdax erstellt werden, diesen kann man formattieren und mounten, danach ist es als Dateisystem verfügbar
- Veränderungen sind in der /tmp/cxltest.raw und /tmp/lsa.raw zu sehen
- diese Veränderungen in den beiden Dateien bleiben bis zum Löschen der Dateien bestehen
- beim Neustarten wird der namespace nicht mehr erkannt und das Erstellen eines neuen Namespaces schlägt fehl ()

Server

CXL Gerät wird am PCI Bus erkannt (lspci)

```
lspci
c1:00.0 CXL: SMART Modular Technologies Device c241
```

In den DVSEC (Designated Vendor-Specific) Capabilities ist als Capability Mem+ angegeben, heißt es kann als Memory Device verwendet werden, aber es ist in den Control Registern nicht aktiviert Mem-

```
sudo lspci -vvv
c1:00.0 CXL: SMART Modular Technologies Device c241 (prog-if 10 [CXL Memory Device (CXL 2.x)
Capabilities: [224 v1] Designated Vendor-Specific: Vendor=1e98 ID=0000 Rev=2 Len=60: CXL
    CXLCap: Cache- IO+ Mem+ Mem HW Init+ HDMCount 1 Viral+
    CXLCtl: Cache- IO+ Mem- Cache SF Cov 0 Cache SF Gran 0 Cache Clean- Viral-
```

CXL Gerät wird aber nicht als CXL Gerät erkannt (cxl list)

```
cxl list
Warning: no matching devices found


[
]
```

In den Kernel Meldungen wird ein Fehler wegen nicht gefundenen Registern gefunden (Yussuf Khalil vom KIT meinte, dass das wahrscheinlich aber auch erst mit CXL 2.0 Karten unterstützt wird, da dann erst der Kernel beim Initialisieren aktiv was macht)

```
sudo dmesg | grep cxl
cxl_pci 0000:c1:00.0: enabling device (0000 -> 0002)
cxl_pci 0000:c1:00.0: Mapped CXL Memory Device resource 0x0000030020f10000
cxl_pci 0000:c1:00.0: registers not found: status mbox memdev
```

BIOS Einstellungen:

Wir haben alles bis auf die CXL Encryption aktiviert. Auf Hinweis von Yussuf Khalil haben wir CXL SMP deaktiviert, das hat aber zu keiner Veränderung geführt

CXL Register manuell auslesen

Wir haben eine kleines C Programm geschrieben, dass eine PCI BAR mit `mmap` in den Speicher mapped und einen Hex-Dump erstellt machen. Die Base Address und den Offset haben wir mit `lspci` ausgelesen.

```
$ sudo lspci -vvv
c1:00.0 CXL: SMART Modular Technologies Device c241 (prog-if 10 [CXL Memory Device (CXL 2.x)
Region 0: Memory at 30020f00000 (64-bit, prefetchable) [size=256K]
Capabilities: [274 v1] Designated Vendor-Specific: Vendor=1e98 ID=0008 Rev=0 Len=28: CXL
    Block1: BIR: bar0, ID: component registers, offset: 0000000000000000
    Block2: BIR: bar0, ID: CXL device registers, offset: 0000000000010000
```

Die Struktur der *component register* stimmt mit den in der Spezifikation (CXL 2.0 Sektion 8.2.4) über ein. Jedoch passt der Dump der *CXL device register* nicht zur Spezifikation (CXL 2.0 Sektion 8.2.8).

Weitere Maßnahmen

Das BIOS wurde auf die Version R29_F43 geupdated. Firmware wurde geupdated (TODO: Welche Firmware?)

Durch die Hilfe von Yussuf Khalil ist uns aufgefallen, dass die CXL Karte im falschen PCI-Slot gesteckt hat. Der Server besitzt nämlich sowohl PCI 4.0 als auch 5.0 Slot. Die Karte steckte in einem der 4.0 Slot. Nach dem Umstecken der Karte lässt sich der Server jedoch nicht mehr Boot. Wenn man den ihn startet landet man in einem Boot-Loop und gelagt auch nicht ins BIOS.

Future Work

- Prüfen ob die CXL Karte wirklich der Grund für den Boot-Loop ist.