gdb & lldb debugging custom RPI4 linux kernel

Agenda

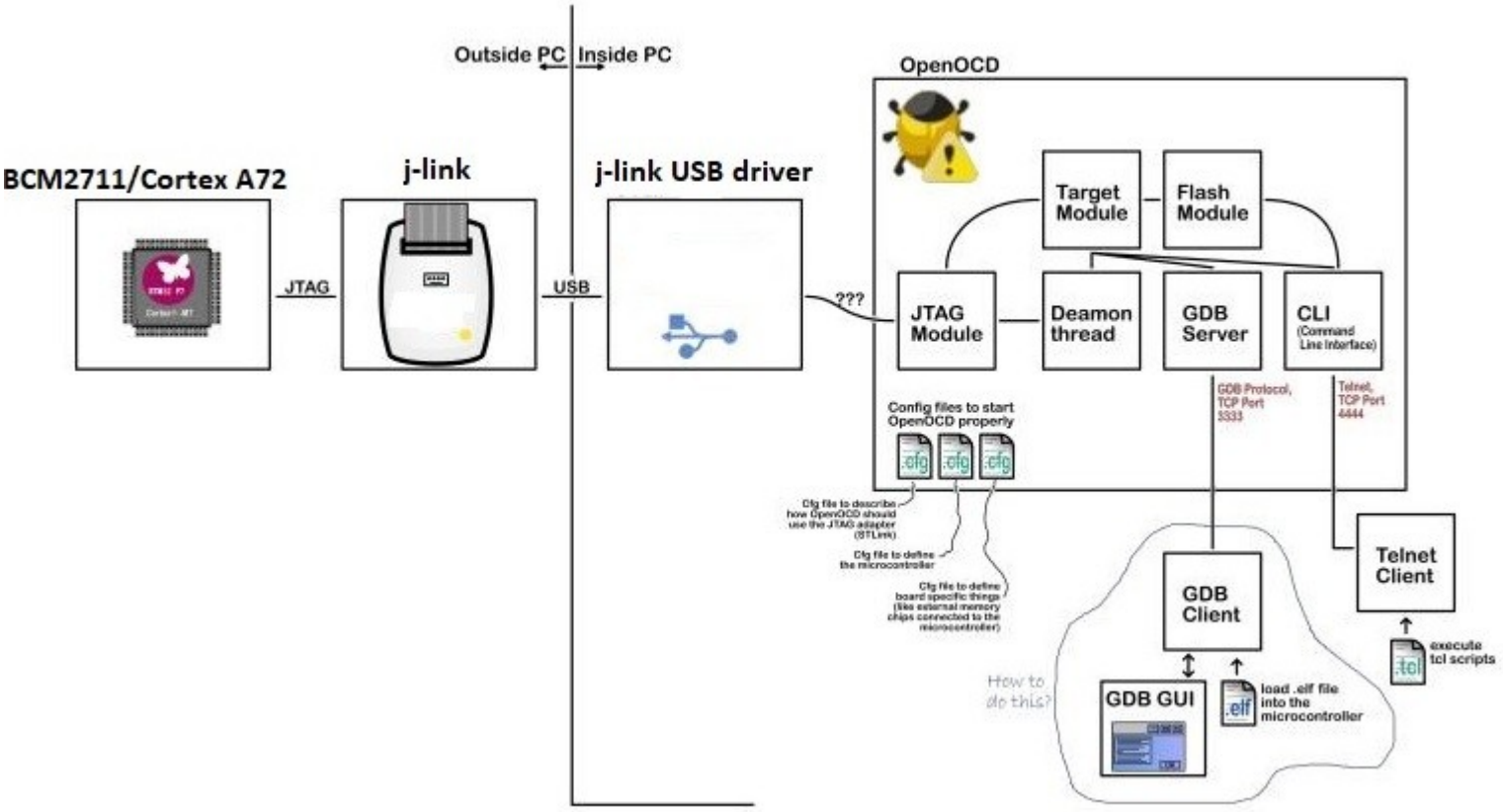
1. Motivation
2. HW- SW-Setup Overview
3. GDB
4. lldb
5. Demo

1. **Motivation**

learn the inner workings of the linux kernel

debug custom linux kernel modules from the beginning

2. HW- SW-Setup Overview



2. HW- SW-Setup Software

Host software:

OpenOCD

GDB

lldb

Target software:

(custom) U-Boot

RPI4-64bit- patched upstream linux kernel

ATTENTION: use Master RB5 SD-Card + Image 2.4 MB /srv/tftp

NFS-mounted rootfs

2. HW- SW-Setup openocd

Version:

xPack OpenOCD x86_64 Open On-Chip Debugger 0.11.0+dev (2022-03-25-17:31)

interface: jlink.cfg

target: rpi4_neu_2.cfg

2. HW- SW-Setup openocd rpi4_neu_2.cfg

```
set _CHIPNAME bcm2711
set _DAP_TAPID 0x4ba00477
adapter_khz 1000
transport select jtag
reset_config trst_and_srst
telnet_port 4444
# create tap
jtag newtap auto0 tap -irlen 4 -expected-id $_DAP_TAPID
# create dap
dap create auto0.dap -chain-position auto0.tap
set CTIBASE {0x80420000 0x80520000 0x80620000 0x80720000}
set DBGBASE {0x80410000 0x80510000 0x80610000 0x80710000}
set _cores 4
set _TARGETNAME $_CHIPNAME.a72
set _CTINAME $_CHIPNAME.cti
set _smp_command "„
for {set _core 0} {$_core < $_cores} { incr _core} {
        cti create $_CTINAME.$_core -dap auto0.dap -ap-num 0 -ctibase [lindex $CTIBASE $_core]

        set _command "target create ${_TARGETNAME}.$_core aarch64 \
                                -dap auto0.dap  -dbgbase [lindex $DBGBASE $_core] \
                        -coreid $_core -cti $_CTINAME.$_core"
        if {$_core != 0} {
                set _smp_command "$_smp_command $_TARGETNAME.$_core"
        }
        else {
                set _smp_command "target smp $_TARGETNAME.$_core"
        }

        eval $_command

#       $_TARGETNAME.$_core configure -event reset-assert-post "aarch64 dbginit„
#       $_TARGETNAME.$_core configure -event gdb-attach { halt }
}

eval $_smp_command
targets $_TARGETNAME.0
```

3. GDB Steps setup gdb debugging

agent-proxy: split uart stream into serial and debugger streams

poweron RPI4 and boot to U-boot console (Terminal 1)

start openocd and connect to j-link JTAG probe (Terminal 2)

start gdb & connect to gdb-server from openocd (Terminal 3)

launch linux kernel (waiting for breakpoint _start_kernel() to trigger) (Terminal 3)

boot linux kernel via „> boot“ (Terminal 1)

-> breakpoint „_start_kernel()“ triggers

4. lldb　　building lldb

git clone https://github.com/llvm/llvm-project.git

mkdir llvm-project/llvm/build

cd llvm-project/llvm/build

cmake .. -DCMAKE_BUILD_TYPE=Release -G Ninja
-DCMAKE_C_COMPILER=/usr/lib/llvm-11/bin/clang
-DCMAKE_CXX_COMPILER=/usr/lib/llvm-11/bin/clang++
-DLLVM_ENABLE_LLD=ON
-DLLVM_ENABLE_PROJECTS="clang;lld;compiler-rt;**lldb**"
-DLLVM_TARGETS_TO_BUILD="AArch64;ARM;X86"
-DLLDB_ENABLE_LIBEDIT=ON
-DLLVM_ENABLE_ASSERTIONS=OFF
-DCMAKE_CROSSCOMPILING=True
-DLLVM_TARGET_ARCH=AArch64
-DLLVM_DEFAULT_TARGET_TRIPLE=aarch64-rey-linux

ninja
sudo ninja install

4. lldb Voltron

Voltron is an extensible debugger UI toolkit written in Python

Install

git clone https://github.com/snare/voltron
cd voltron
./install.sh

patch (for matching „aarch64“ in stead of „arm64“)
~/.local/lib/python3.6/site-packages/Voltron/dbg.py
added 2x aarch64
~/.local/lib/python3.6/site-packages/Voltron/plugins/view/register.py
added 2x aarch64
~/.local/lib/python3.6/site-packages/Voltron/plugins/debugger/dbg_lldb.py
replaced “xrange” with “range” line 197

4. lldb Steps setup lldb debugging

agent-proxy: split uart stream into serial and debugger

poweron RPI4 and boot to U-boot console (Terminal 1)

start openocd and connect to j-link JTAG probe (Terminal 2)

„> tmuxinator start voltron“ and connect to gdb-server from openocd (Terminal 3)

launch linux kernel (waiting for breakpoint _start_kernel() to trigger) (Terminal 3)

boot linux kernel via „> boot“ (Terminal 1)

-> breakpoint „_start_kernel()“ triggers