I.MX8MP STANDALONE APPLICATION RUNNING TIPS

CAS MAGGIE JIANG

2024 FEB

Topics

- Some customer need to run standalone application in i.MX side.
- This article describe how to run standalone application in uboot and kernel, how to improve application running performance.
- It takes i.MX8MP as example, which is also suitable for other i.MX platform.

Install Toolchain

- Produce i.MX Yocto SDK toolchain:

  DISTRO=fsl-imx-xwayland MACHINE=imx8mp-lpddr4-evk source imx-setup-release.sh -b build-xwayland

  bitbake imx-image-multimedia -c populate_sdk

  (or use bitbake meta-toolchain for small size toolchain)

- Install toolchain

  cd (yocto build directory..)/build-xwayland/tmp/deploy/sdk

  ./fsl-imx-xwayland-glibc-x86_64-imx-image-multimedia-armv8a-imx8mp-lpddr4-evk -toolchain-6.1-mickledore.sh

  Then it will install on default directory:

  /opt/fsl-imx-xwayland

RUN APPLICATION IN UBOOT

Put application to Uboot standalone directory

- In /uboot-imx/examples/standalone/

  There is already hello_world.c here, we can put test.c application in this directory,
- Inside hello_world.c, we add new function calling test() which from test.c

```
int hello_world(int argc, char *const argv[])
        {……
                + test();
        }
```

- Modify makefile to add new function compiling:

uboot-imx/examples/standalone$ vi Makefile

```
LIB     = $(obj)/libstubs.o
LIBOBJS-$(CONFIG_PPC) += ppc_longjmp.o ppc_setjmp.o
LIBOBJS-y += stubs.o test.o
```

Increase CPU frequency

- If we want to increase application running performance in uboot, we can increase i.MX8MP CPU frequency to maximal 1.8GHz. By default, it is setting at 1.2GHz.

b/arch/arm/mach-imx/imx8m/clock_imx8mm.c

```
-       /* Configure ARM at 1.2GHz */
+       /* Configure ARM at 1.8GHz */
 -      intpll_configure(ANATOP_ARM_PLL, MHZ(1200));
+       intpll_configure(ANATOP_ARM_PLL, MHZ(1800));
```

Then the uboot log print out 1.8G information:

```
U-Boot 2023.04-dirty (Jan 23 2024 - 18:09:51 +0800)

CPU:   i.MX8MP[8] rev1.1 at 1800MHz
CPU:   Commercial temperature grade (0C to 95C) at 38C
Reset cause: POR
Model: NXP i.MX8MPlus LPDDR4 EVK board
```

Modify uboot config file- include API examples

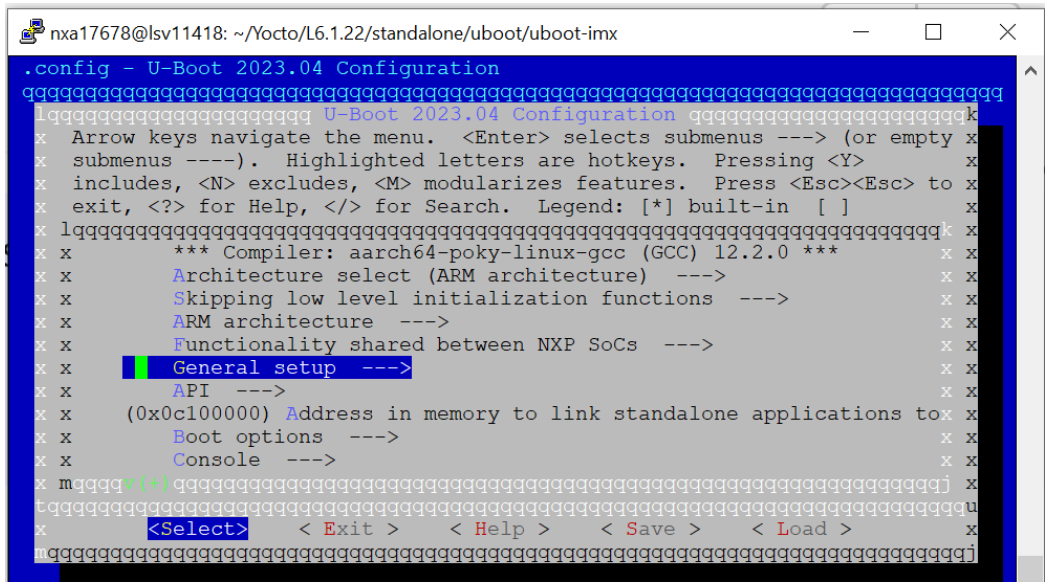
source /opt/fsl-imx-xwayland/6.1-mickledore/environment-setup-armv8a-poky-linux

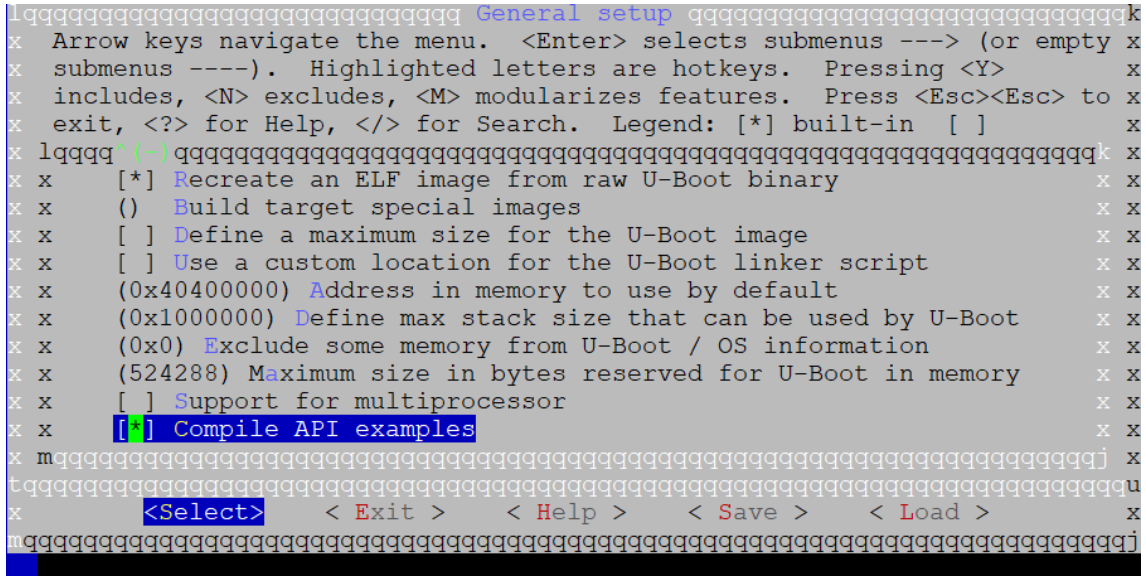
cd uboot-imx：

make imx8mp_evk_defconfig

make menuconfig

Choose “General setup”- “**Compile API examples**”





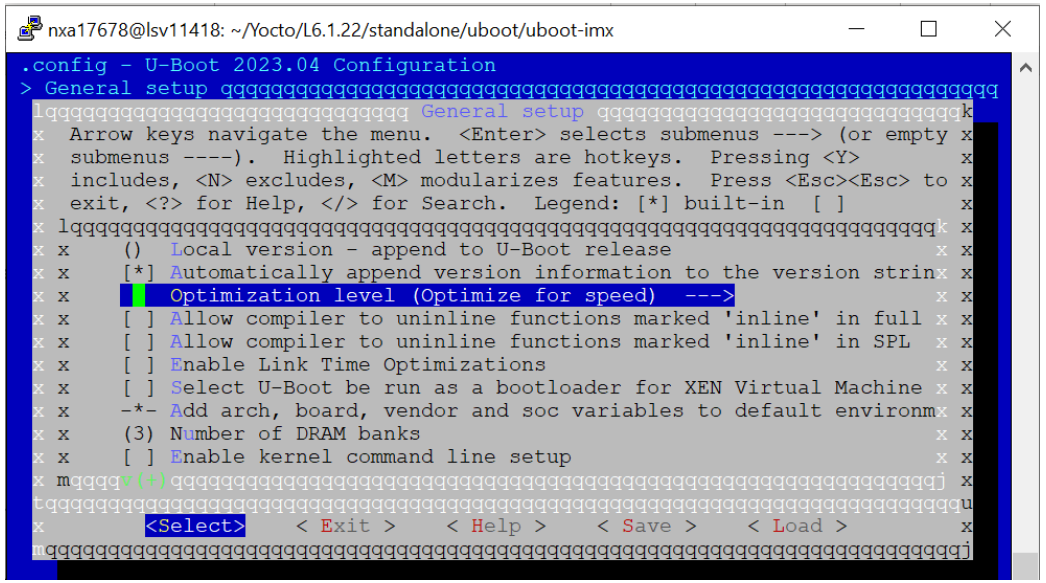
Modify uboot config file – increase optimization level

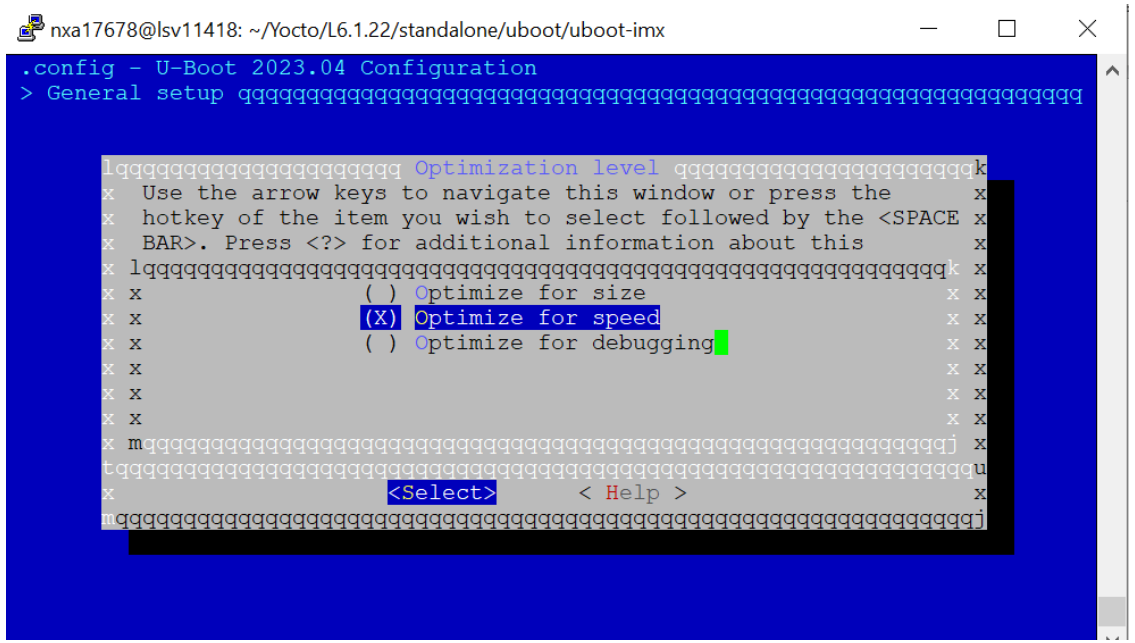
- Choose “General setup”-”Optimization level”-”**Optimize for speed**”

Default setting is “Optimize for size”. If customer have higher performance requirement, we can change this config.

- run “**CONFIG_STANDALONE_LOAD_ADDR=0x43000000 make**” to compile

(CONFIG_STANDALONE_LOAD_ADDR=0x43000000 is parameter pass to uboot)





Check compile output file

Check compile output file to confirm it make effect:

- Config file: uboot-imx/.config:

  CONFIG_EXAMPLES=y

  CONFIG_CC_OPTIMIZE_FOR_SPEED=y

- Compile command parameter:uboot-imx/examples/standalone/.hello_world.o.cmd:

cmd_examples/standalone/hello_world.o := aarch64-poky-linux-gcc -Wp,-MD,examples/standalone/.hello_world.o.d ….. -fno-PIE -Ofast

- GCC compile optimization level(low to high, -Ofast is highest level):

  -O/-O1 -O2 -Os -O3 -Ofast

Produce uboot and application binary

- After change config file and compile uboot, it produce new u-boot.bin, combine uboot/ATF/firmware to flash.bin, then burn flash.bin to the board.
- It also produce hello_world.bin under /uboot-imx/examples/standalone at the same time. We need to send hello_world.bin to board and run it.
- When customer modify their application in hello_world.c/test.c （In help_world.c, it call functions from test.c), hello_world.bin will be updated , they can download hello_world.bin to the board next time. No need to update flash.bin every time.
- It need to make sure uboot inside flash.bin and hello_world.bin are in same uboot environment.

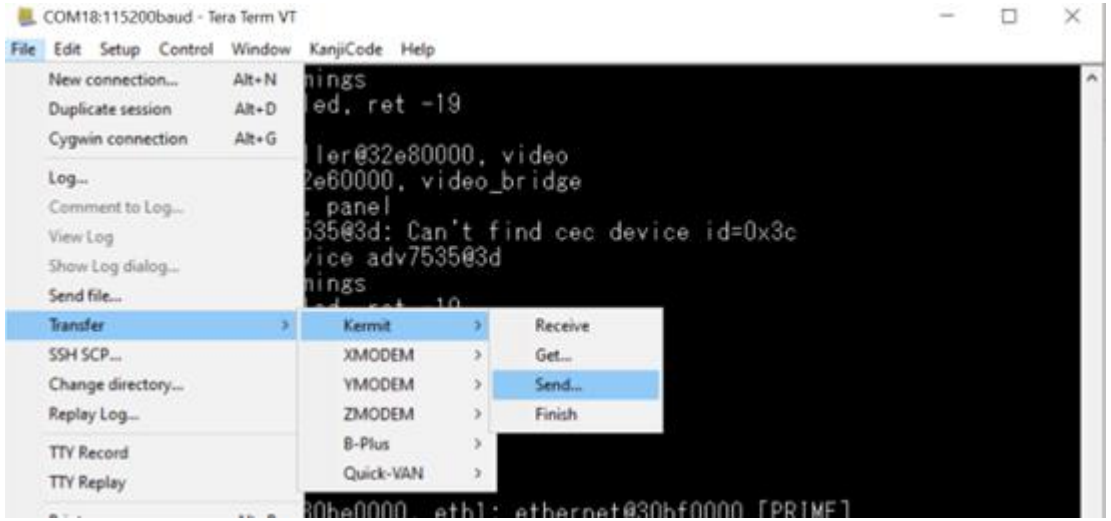
```
nxa17678@lsv11418:~/Yocto/L6.1.22/standalone/uboot/uboot-imx/examples/standalone
$ ls
atmel_df_pow2.c  hello_world.srec  ppc_setjmp.S            stubs.o   test.su
hello_world      hello_world.su    README.smc91111_eeprom  stubs.su
hello_world.bin  libstubs.o        sched.c                 test_2.c
hello_world.c    Makefile          sparc.lds               test.c
hello_world.o    ppc_longjmp.S     stubs.c                 test.o
```

Download application to board

- We can use “loadb” command to download hello_world.bin to a blank address, such as 0x43000000, which can’t affect uboot running.
- **loadb 0x43000000**

```
u-boot=> loadb 0x43000000
## Ready for binary (kermit) download to 0x43000000 at 115200 bps...
```

- Then choose UART tools Kermit protocol to send hello_world.bin to board.



Run application from specific address

- Then run the application from the address:

  go 0x43000000

  It output print information from application code:

```
u-boot=> go 0x43000000
## Starting application at 0x43000000 ...
Example expects ABI version 9
Actual U-Boot ABI version 9
Hello World
argc = 1
argv[0] = "0x43000000"
argv[1] = "<NULL>"
NXP build, duration = 8000002 ticks....
NXP build2, duration = 0 ticks....
NXP build3, c = 40000, duration = 0 ticks....
Hit any key to exit ...
```

Choose suitable blank address

- We choose 0x43000000 because it is a blank address in i.MX8MP when running uboot.
- This address need to be aligned with uboot definition on:

uboot-imx/examples/standalone/Makefile: LDFLAGS_STANDALONE += -Ttext $(CONFIG_STANDALONE_LOAD_ADDR)

While default value in include/config/auto.conf:CONFIG_STANDALONE_LOAD_ADDR=0x0c100000

This address conflict with i.MX8MP uboot running area.

- So we add following parameter when compile uboot:

make CONFIG_STANDALONE_LOAD_ADDR=0x43000000

- We can loadb hello_world.srec to the board to double check the load address and start address.

```
u-boot=> loadb
## Ready for binary (kermit) download to 0x40400000 at 115200 bps...
## Total Size      = 0x00001396 = 5014 Bytes
## Start Addr      = 0x40400000
u-boot=>
```

RUN APPLICATION IN KERNEL

Compile standalone application

- Setup toolchain: source /opt/fsl-imx-xwayland/6.1-mickledore/environment-setup-armv8a-poky-linux
- mkdir application/ directory, put all applications demo.c test.c… here. demo.c is main function.

In demo.c, it call test() from test.c and calculate running time.

```
void main()
{       t1 = get_ticks();
        test();
        t2 = get_ticks();
        printf("during ticks = %ld\n", t2 - t1);
}
```

- Compile app:

$CC demo.c test.c -Ofast -o demo (using -Ofast to get highest optimization level)

It produce demo.o finally.

Isolate CPU and run application in one dedicated CPU

- In uboot command line, we can add isolcpus=1-3 to isolate three cpu cores: 1, 2, 3

  (i.MX8MP have four cpu cores: 0 1 2 3)

  setenv mmcargs 'setenv bootargs ${jh_clk} ${mcore_clk} console=${console} root=${mmcroot} **isolcpus=1-3'**
- Put demo.o to the board, then run: chmod 777 demo
- **taskset -c 1 ./demo** (taskset means to combine to one CPU to run, -c 1 means CPU1, it is aligned with isolcpus =1 in uboot command line)
- Then it output running print information as following:

Set to CPU performance mode

- In order to run at maximal CPU frequency, suggest to set to performance mode to run at highest CPU frequency. It is ondemand mode by default.

Take CPU1 as example:

root@imx8mpevk:~# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor

```
root@imx8mpevk:~# cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
ondemand
root@imx8mpevk:~# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scali
g_governor
root@imx8mpevk:~# cat /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
performance
```

Then run application again：

root@imx8mpevk:~# **taskset -c 1 ./demo**

Run dhrystone benchmark test

We can run dhrystone benchmark as an example:

- DMIPS（Dhrystone Million Instructions Per Second)
- dhry2 is compiled with optimization level -O3 and download from NXP web:

https://www.nxp.com/docs/en/application-note-software/AN13917SW.zip

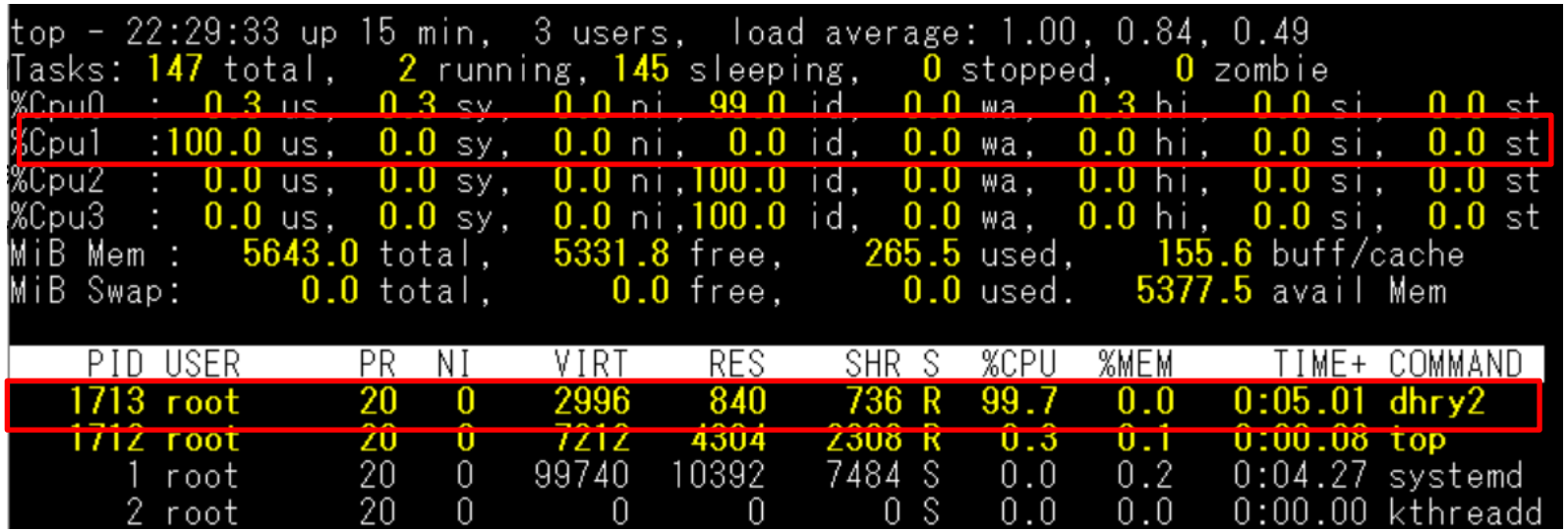
- We can use script **dhrystone.sh** to repeat running dhry2 on CPU 1 for stress test.

```
while true; do
./dhry2
done
```

```
root@imx8mpevk:~# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
root@imx8mpevk:~# taskset -c 1 ./dhrystone.sh
```

Dhrystone result in i.MX8MP

- We can get the benchmark data from the log.
- Use “top” command to check. The dhry2 application is running on CPU1 . CPU1 is almost 100%, CPU0/2/3 are spare.



SOME OTHER OPTIMIZATION WAYS

Dynamic link and static link library

- Normally it is dynamic linked library by default when compiling application. Its advantage is executable file size is small. But it may affect application loading time and affect performance. If customer has strict requirement for application loading time, we may try following way:
- Using default dynamic link compile, loading linked library to cache in advance;
- Using static link compile to compile image.

Dynamic link library

- It is dymamic linked by default when compiling application. Use file command to check:

```
root@imx8mpevk:~# file demo
demo: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux-aarch64.so.1, BuildID[sha1]=beb13fd62764b301af
85c33eadb17b447f9c2c7f, for GNU/Linux 3.14.0, with debug_info, not stripped
root@imx8mpevk:~#
```

- Check which dynamica library are linked.

```
root@imx8mpevk:~# ldd demo
        linux-vdso.so.1 (0x0000ffffaa1e8000)
        libc.so.6 => /lib/libc.so.6 (0x0000ffffa9fd0000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffffaa1ab000)
```

- Using vmtouch tool to fix dynanica linked library to cache in advance.

```
root@imx8mpevk:~# ./vmtouch -l -d -w -t /lib/ld-linux-aarch64.so.1
LOCKED 50 pages (200K)
root@imx8mpevk:~# ./vmtouch -l -d -w -t /lib/libc.so.6
LOCKED 404 pages (1M)
root@imx8mpevk:~# ./vmtouch -l -d -w -t demo
LOCKED 19 pages (76K)
```

- Then run application:

  root@imx8mpevk:~# taskset -c 1 ./demo

add mlockall() function

- Try to add mlockall() before calling timer start and customer function.
- It locks all pages mapped into the address space of the calling process. It can help to reduce the page fault latency and avoid potential page reclaim.

```
 + #include <sys/mman.h>
void main()
{
+     mlockall(MCL_CURRENT);
    t1 = get_ticks();
    test();
     t2 = get_ticks();
     printf("during ticks = %ld\n", t2 - t1)
}
```

Use linaro toolchain to realize static link compile

i.MX Yocto default toolchain don’t support static link compile. We can also use another toolchain to have a test.

- Download linaro toolchain which support static link compile:

https://releases.linaro.org/components/toolchain/binaries/latest-7/aarch64-linux-gnu/

download gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz

tar –xvf gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu.tar.xz

- Install toolchain and add to PATH

export PATH=~/Yocto/tool/gcc-linaro-7.5.0-2019.12-x86_64_aarch64-linux-gnu/bin:$PATH

- Compile application using static link

aarch64-linux-gnu-gcc demo.c test.c -Ofast -static -o demo_static

- Use “strip” command to reduce demo_static.o file size:

aarch64-linux-gnu-strip demo_static

Run static linked compiled file

- Check demo_static file status on board using “file” command

```
root@imx8mpevk:~# file demo_static
demo_static: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), statical
y linked, for GNU/Linux 3.7.0, BuildID[sha1]=f189f617b6f2a8828c74f8fa94a3bad008d
2048a, stripped
```

- Run application on board:

```
root@imx8mpevk:~# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scalin
g_governor
root@imx8mpevk:~# taskset -c 1 ./demo_static
hello world
during ticks = 8003040
```

Conclusion

- It describe how to run standalone application in i.MX8MP board, including in uboot & kernel.
- If customer want to get better performance, they can consider following way:
  - Check application compile parameters, including optimization level;
  - Set board status to run at maximal CPU frequency and DDR frequency;
  - Check dynamic link library status.
  - Check application memory allocation status.
  - ……

SECURE CONNECTIONS
FOR A SMARTER WORLD