6-制作测试固件验证串口打印 | TimerのBlog

yanglianoo.github.io/2023/06/17/QEMU中自定义开发板6-制作测试固件验证串口打印

2023年6月17日

> 参考地址：基于qemu-riscv从0开始构建嵌入式linux系统ch4. 制作测试固件验证串口打印 — 主页 (quard-star-tutorial.readthedocs.io)

在`quard-star`目录下新建boot文件夹，在此文件夹下新建`boot.lds`和`start.s`文件，如下

```
timer@DESKTOP-JI9EVEH:~$ cd quard-star/
timer@DESKTOP-JI9EVEH:~/quard-star$ ls
README.md  boot  build.sh  output  qemu-8.0.2  quard_star_tutorial  run.sh
timer@DESKTOP-JI9EVEH:~/quard-star$ cd boot
timer@DESKTOP-JI9EVEH:~/quard-star/boot$ ls
boot.lds  start.s
```

`start.s`文件

avrasm

```
        .section .text              //定义数据段名为.text
        .globl _start               //定义全局符号_start
        .type _start,@function      //_start为函数

_start:                             //函数入口
    csrr    a0, mhartid             //csr是riscv专有的内核私有寄存器，独立编地在12位地址
                                    //mhartid寄存是定义了内核的hart id，这里读取到a0寄存器里
    li          t0,     0x0             //li是伪指令，加载立即数0到t0
        beq             a0, t0, _core0      //比较a0和t0,相等则跳转到_core0地址处，否则
向下执行
_loop:                              //定义一个_loop符号
        j               _loop               //跳转到_loop，此处形成循环，用意为如果当前
cpu core不为
                                    //hart 0则循环等待，为hart 0则继续向下执行
_core0:                             //定义一个core0才能执行到此处
        li              t0,      0x100          //t0 = 0x100
        slli    t0,     t0, 20          //t0左移20位 t0 = 0x10000000
        li              t1,     'H'             //t1 = 'H' 字符的ASCII码值写入t1
        sb              t1, 0(t0)           //s是store写入的意思，b是byte，这里指的是写
入t1
                                    //的值到t0指向的地址，即为写入0x10000000这个寄存器
                                    //这个寄存器正是uart0的发送data寄存器，此时串口会输出"H"
        li              t1,     'e'             //接下来都是重复内容
        sb              t1, 0(t0)
        li              t1,     'l'
        sb              t1, 0(t0)
        li              t1,     'l'
        sb              t1, 0(t0)
        li              t1,     'o'
        sb              t1, 0(t0)
        li              t1,     ' '
        sb              t1, 0(t0)
        li              t1,     'Q'
        sb              t1, 0(t0)
        li              t1,     'u'
        sb              t1, 0(t0)
        li              t1,     'a'
        sb              t1, 0(t0)
        li              t1,     'r'
        sb              t1, 0(t0)
        li              t1,     'd'
        sb              t1, 0(t0)
        li              t1,     ' '
        sb              t1, 0(t0)
        li              t1,     'S'
        sb              t1, 0(t0)
        li              t1,     't'
        sb              t1, 0(t0)
        li              t1,     'a'
        sb              t1, 0(t0)
        li              t1,     'r'
        sb              t1, 0(t0)
        li              t1,     ' '
        sb              t1, 0(t0)
        li              t1,     'b'
        sb              t1, 0(t0)
        li              t1,     'o'
        sb              t1, 0(t0)
```

```
        li              t1,     'a'
        sb              t1, 0(t0)
        li              t1,     'r'
        sb              t1, 0(t0)
        li              t1,     'd'
        sb              t1, 0(t0)
        li              t1,     '!'
        sb              t1, 0(t0)
        li              t1,     '\n'
        sb              t1, 0(t0)          //到这里就会输出"Hello Quard Star board!"
        j               _loop              //完成后进入loop

    .end                        //汇编文件结束符号
```

这里做的事情比较简单，首先读取hartid的值，如果是编号为0的hart就跳转到_core0处执行，如果不是就进入_loop循环卡住。在core0函数中就是往uart0=0x10000000处一个字节一个字节的写数据，数据会输出到主机的控制台上。

boot.lds文件

plaintext

```
OUTPUT_ARCH( "riscv" )  /*输出可执行文件平台*/

ENTRY( _start )         /*程序入口函数*/

MEMORY                  /*定义内存域*/
{
    /*定义名为flash的内存域属性以及起始地址，大小等*/
        flash (rxai!w) : ORIGIN = 0x20000000, LENGTH = 512k
}

SECTIONS                /*定义段域*/
{
  .text :               /*.text段域*/
  {
    KEEP(*(.text))      /*将所有.text段链接在此域内，keep是保持防止优化，即无论如何都保留此段*/
  } >flash              /*段域的地址(LMA和VMA相同)位于名为flash内存域*/
}
```

boot.lds是给链接器传参数的，目的是为了把start.s里的代码连接到flash处。

build.sh修改

shell

```
CROSS_PREFIX=riscv64-unknown-elf
if [ ! -d "$SHELL_FOLDER/output/lowlevelboot" ]; then
mkdir $SHELL_FOLDER/output/lowlevelboot
fi
cd boot
$CROSS_PREFIX-gcc -x assembler-with-cpp -c start.s -o
$SHELL_FOLDER/output/lowlevelboot/start.o
$CROSS_PREFIX-gcc -nostartfiles -T./boot.lds -Wl,-
Map=$SHELL_FOLDER/output/lowlevelboot/lowlevel_fw.map -Wl,--gc-sections
$SHELL_FOLDER/output/lowlevelboot/start.o -o
$SHELL_FOLDER/output/lowlevelboot/lowlevel_fw.elf
# 使用gnu工具生成原始的程序bin文件
$CROSS_PREFIX-objcopy -O binary -S $SHELL_FOLDER/output/lowlevelboot/lowlevel_fw.elf
$SHELL_FOLDER/output/lowlevelboot/lowlevel_fw.bin
# 使用gnu工具生成反汇编文件，方便调试分析（当然我们这个代码太简单，不是很需要）
$CROSS_PREFIX-objdump --source --demangle --disassemble --reloc --wide
$SHELL_FOLDER/output/lowlevelboot/lowlevel_fw.elf >
$SHELL_FOLDER/output/lowlevelboot/lowlevel_fw.lst

cd $SHELL_FOLDER/output/lowlevelboot
rm -rf fw.bin
dd of=fw.bin bs=1k count=32k if=/dev/zero
dd of=fw.bin bs=1k conv=notrunc seek=0 if=lowlevel_fw.bin
```

这里需要riscv的编译器，我使用的是`riscv64-unknown-elf-gcc`，在build.sh中执行编译生成固件用于qemu启动。

`run.sh`修改

plaintext

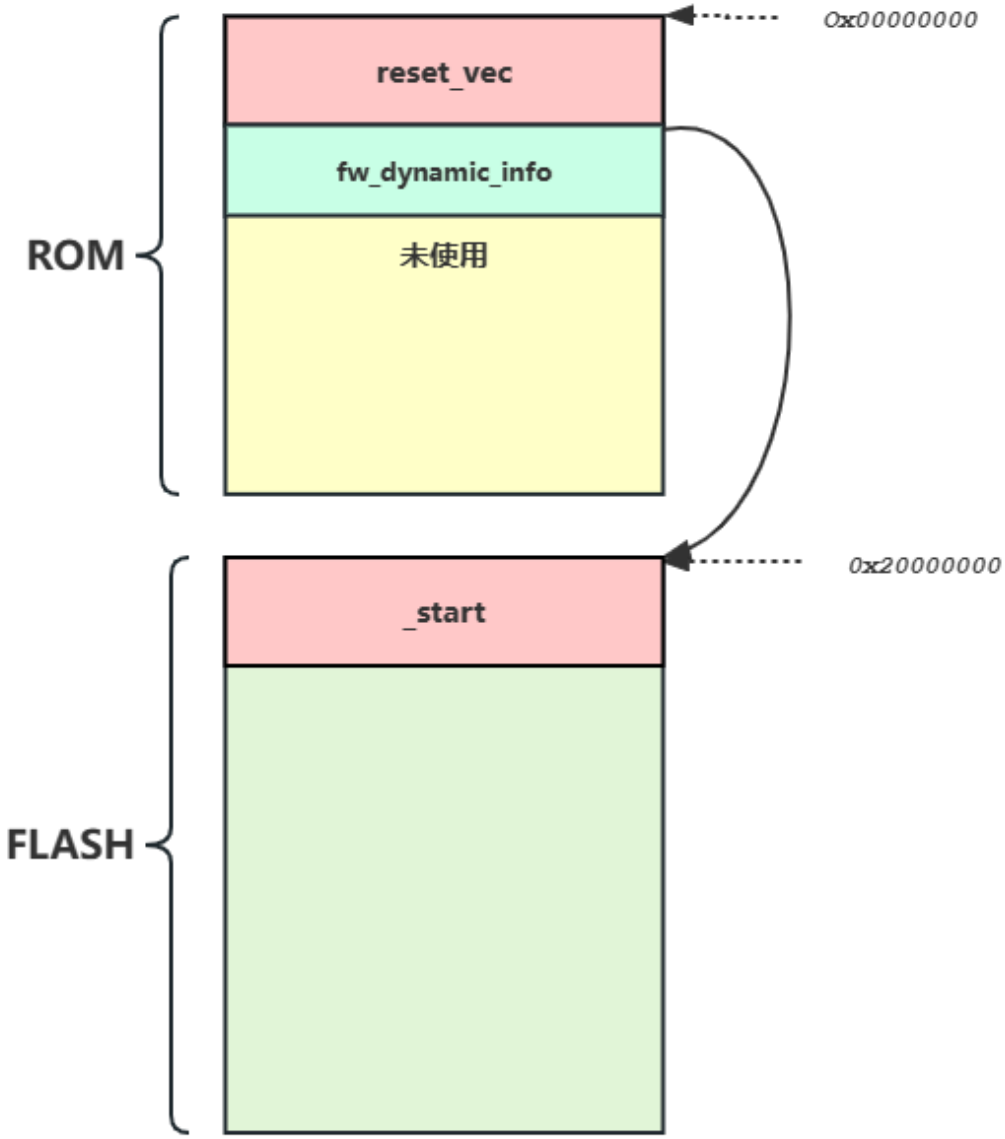
```
SHELL_FOLDER=$(cd "$(dirname "$0")";pwd)

$SHELL_FOLDER/output/qemu/bin/qemu-system-riscv64 \
-M quard-star \
-m 1G \
-smp 8 \
-bios none \
#-monitor stdio \
-drive
if=pflash,bus=0,unit=0,format=raw,file=$SHELL_FOLDER/output/lowlevelboot/fw.bin \
-nographic --parallel none
```

测试，执行 `./build.sh` 和 `./run.sh`，结果如下：可以看到在串口控制台输出了`Hello Quard Star board!`


QEMU
Machine View
serial0 console
Hello Quard Star board!

内存布局如下


0x00000000
reset_vec
fw_dynamic_info
未使用
ROM
0x20000000
_start
FLASH