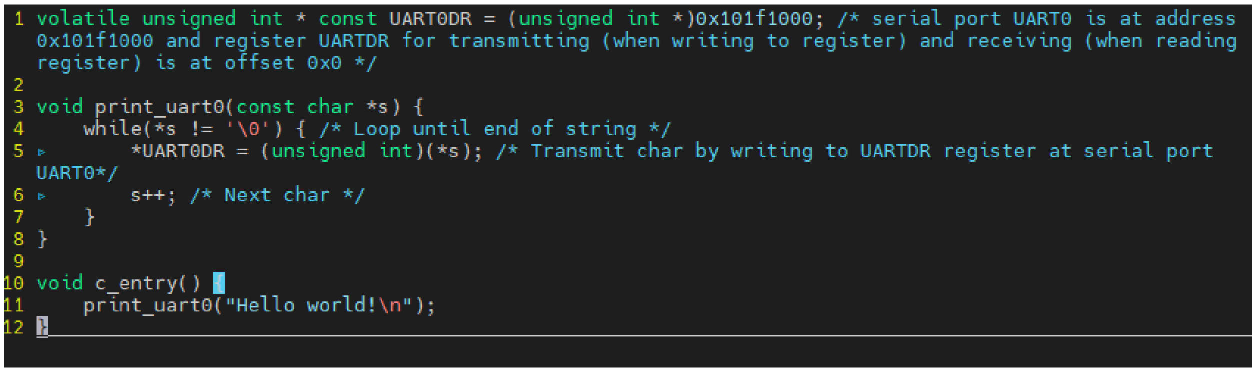
Andrew Ma

Lab 12

CPE 435

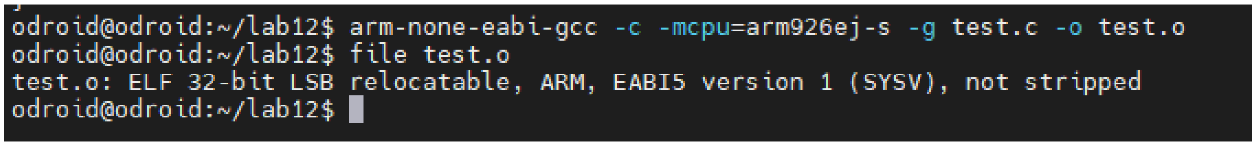
4/5/21

1. Installing gcc-arm-none-eabi and qemu-system-arm with `sudo apt-get install gcc-arm-none-eabi qemu-system-arm`
2. The VersatilePB platform supports 4 serial ports and we use the first serial port UART0 at address 0x101f1000. The register used to transmit and receive the bytes is UARTDR, and it is at offset 0x0.



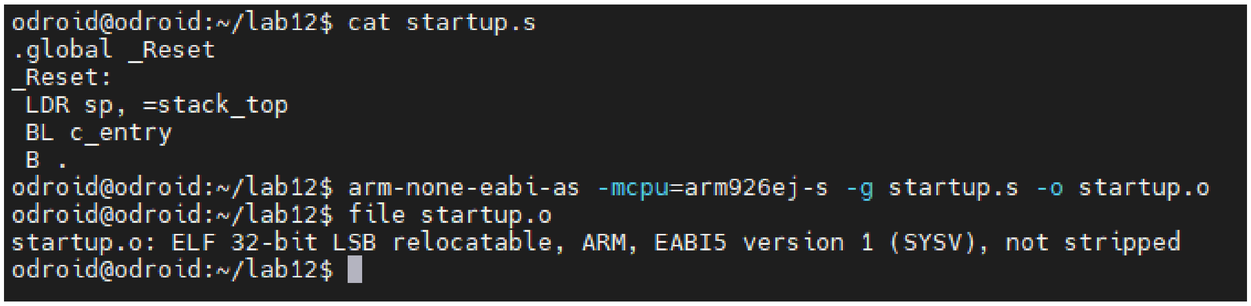
```
volatile unsigned int * const UART0DR = (unsigned int *)0x101f1000; /* serial port UART0 is at address 0x101f1000 and register UARTDR for transmitting (when writing to register) and receiving (when reading register) is at offset 0x0 */

void print_uart0(const char *s) {
    while(*s != '\0') { /* Loop until end of string */
        *UART0DR = (unsigned int)(*s); /* Transmit char by writing to UARTDR register at serial port UART0*/
        s++; /* Next char */
    }
}

void c_entry() {
    print_uart0("Hello world!\n");
}
```

3. test.c compiled to test.o



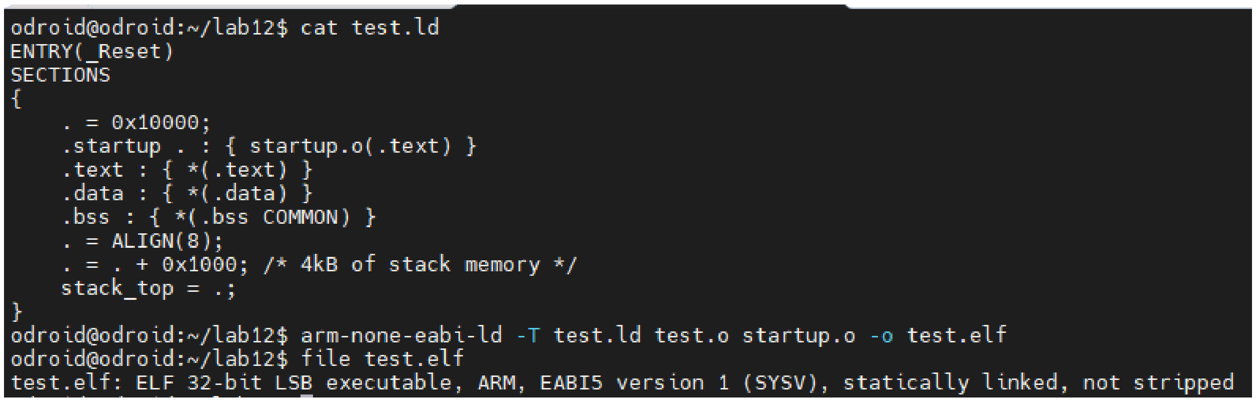
```
odroid@odroid:~/lab12$ arm-none-eabi-gcc -c -mcpu=arm926ej-s -g test.c -o test.o
odroid@odroid:~/lab12$ file test.o
test.o: ELF 32-bit LSB relocatable, ARM, EABI5 version 1 (SYSV), not stripped
odroid@odroid:~/lab12$
```

4. startup.s assembled to startup.o



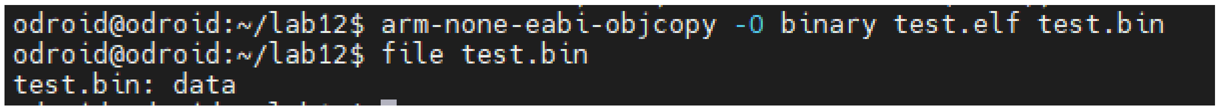
```
odroid@odroid:~/lab12$ cat startup.s
.global _Reset
_Reset:
 LDR sp, =stack_top
 BL c_entry
 B .
odroid@odroid:~/lab12$ arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o
odroid@odroid:~/lab12$ file startup.o
startup.o: ELF 32-bit LSB relocatable, ARM, EABI5 version 1 (SYSV), not stripped
odroid@odroid:~/lab12$
```

5. Linker script test.ld links the test.o and startup.o object files to a test.elf executable file test.elf



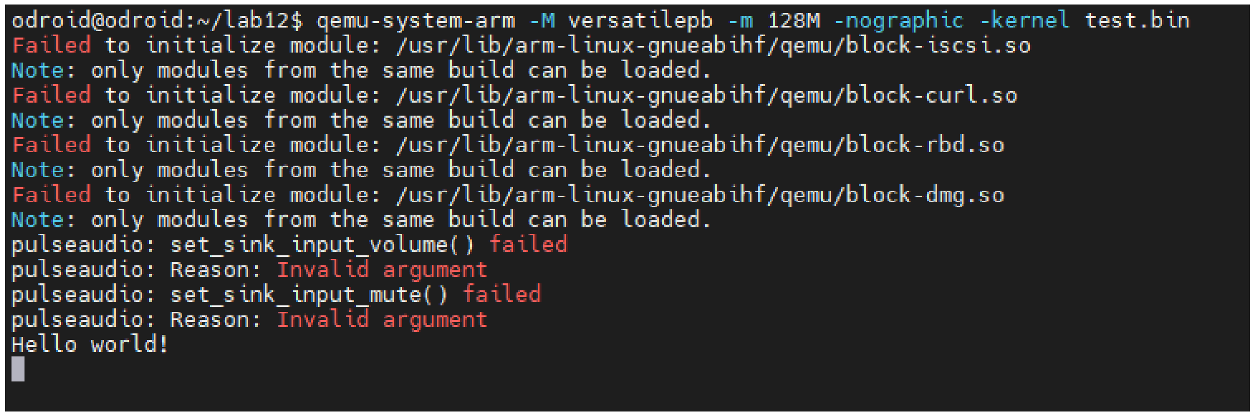
```
odroid@odroid:~/lab12$ cat test.ld
ENTRY(_Reset)
SECTIONS
{
    . = 0x10000;
    .startup . : { startup.o(.text) }
    .text : { *(.text) }
    .data : { *(.data) }
    .bss : { *(.bss COMMON) }
    . = ALIGN(8);
    . = . + 0x1000; /* 4kB of stack memory */
    stack_top = .;
}
odroid@odroid:~/lab12$ arm-none-eabi-ld -T test.ld test.o startup.o -o test.elf
odroid@odroid:~/lab12$ file test.elf
test.elf: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, not stripped
```

6. Copy contents of test.elf to test.bin



```
odroid@odroid:~/lab12$ arm-none-eabi-objcopy -O binary test.elf test.bin
odroid@odroid:~/lab12$ file test.bin
test.bin: data
```

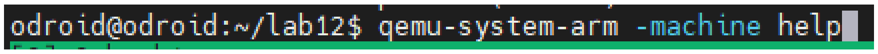
7. Running program in emulator. Emulated system is versatilepb



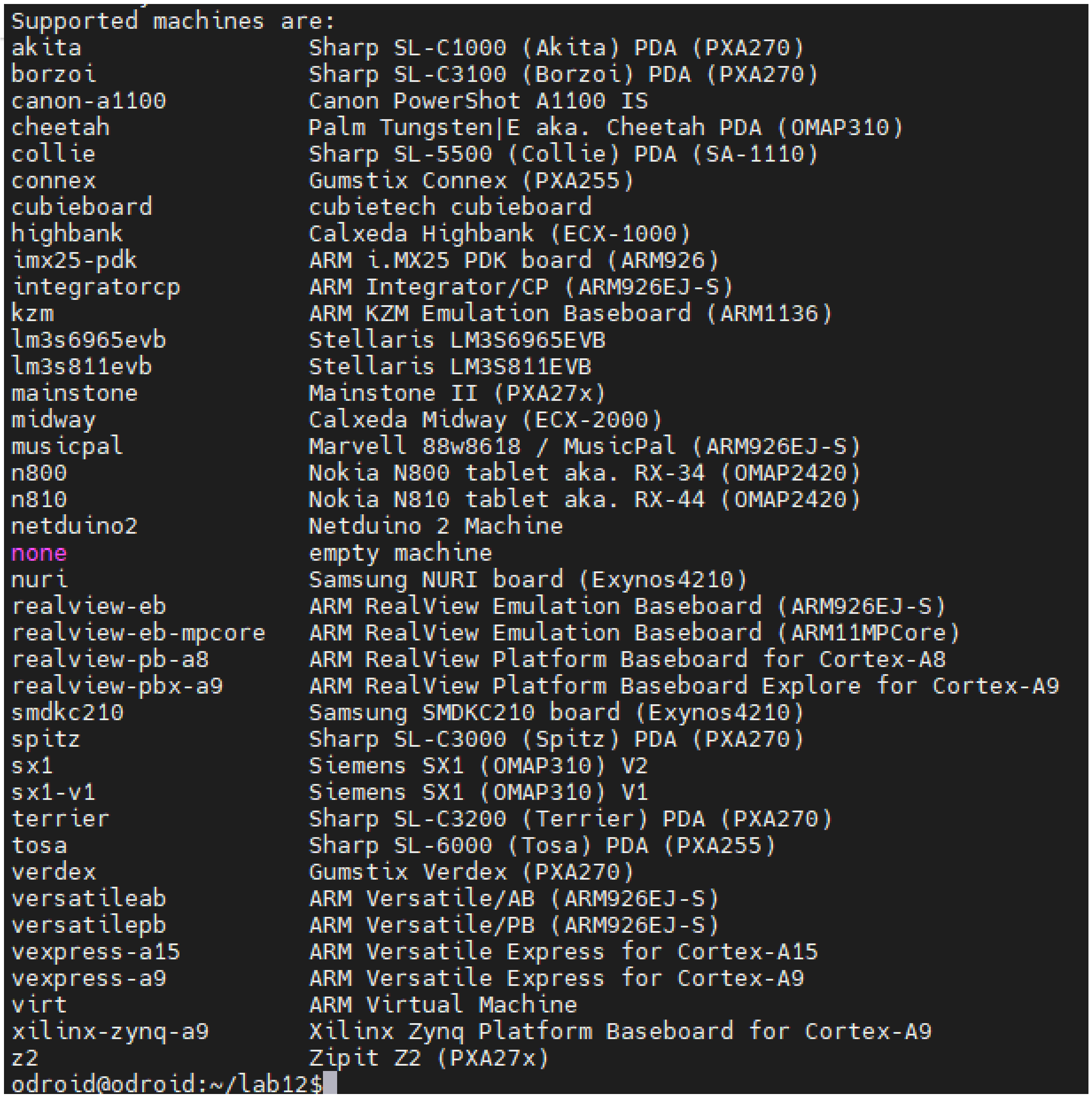
```
odroid@odroid:~/lab12$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel test.bin
Failed to initialize module: /usr/lib/arm-linux-gnueabihf/qemu/block-iscsi.so
Note: only modules from the same build can be loaded.
Failed to initialize module: /usr/lib/arm-linux-gnueabihf/qemu/block-curl.so
Note: only modules from the same build can be loaded.
Failed to initialize module: /usr/lib/arm-linux-gnueabihf/qemu/block-rbd.so
Note: only modules from the same build can be loaded.
Failed to initialize module: /usr/lib/arm-linux-gnueabihf/qemu/block-dmg.so
Note: only modules from the same build can be loaded.
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument
Hello world!
```

8. The first thing I did was install the toolchain for the arm-none-eabi type and also the qemu-system-arm which is the qemu emulator for arm systems. I then copied the test.c file which writes characters to the UARTDR register for serial port UART0, which is located at address 0x101f1000. Writing to this register transmits bytes to the UART0 serial port, and we send the bytes for “Hello world!\n”. We have a startup.s assembler file which exports the name _Reset to the linker to set the program entry point, and it also initializes the stack pointer before calling the c_entry function (that we created in test.c) and saving the return address in the link register. We use “B .” to stay in this program forever like an endless for loop. The qemu emulator emulates linux guest systems and the binary file should be loaded

at address 0x00010000. In our linker script test.ld, we put the program at address 0x10000. We compile the test.c file and assemble to startup.s file, and the link the object files to create a test.elf file. We then copy the elf file data into a test.bin file, that we can run with qemu-system-arm on an emulated versatilepb machine with memory size 128M, console mode, and at address 0x1000.



```
odroid@odroid:~/lab12$ qemu-system-arm -machine help
```



```
Supported machines are:
akita                Sharp SL-C1000 (Akita) PDA (PXA270)
borzoi               Sharp SL-C3100 (Borzoi) PDA (PXA270)
canon-a1100          Canon PowerShot A1100 IS
cheetah              Palm Tungsten|E aka. Cheetah PDA (OMAP310)
collie               Sharp SL-5500 (Collie) PDA (SA-1110)
connex               Gumstix Connex (PXA255)
cubieboard           cubietech cubieboard
highbank             Calxeda Highbank (ECX-1000)
imx25-pdk            ARM i.MX25 PDK board (ARM926)
integratorcp         ARM Integrator/CP (ARM926EJ-S)
kzm                  ARM KZM Emulation Baseboard (ARM1136)
lm3s6965evb          Stellaris LM3S6965EVB
lm3s811evb           Stellaris LM3S811EVB
mainstone            Mainstone II (PXA27x)
midway               Calxeda Midway (ECX-2000)
musicpal             Marvell 88w8618 / MusicPal (ARM926EJ-S)
n800                 Nokia N800 tablet aka. RX-34 (OMAP2420)
n810                 Nokia N810 tablet aka. RX-44 (OMAP2420)
netduino2            Netduino 2 Machine
none                 empty machine
nuri                 Samsung NURI board (Exynos4210)
realview-eb          ARM RealView Emulation Baseboard (ARM926EJ-S)
realview-eb-mpcore   ARM RealView Emulation Baseboard (ARM11MPCore)
realview-pb-a8       ARM RealView Platform Baseboard for Cortex-A8
realview-pbx-a9      ARM RealView Platform Baseboard Explore for Cortex-A9
smdkc210             Samsung SMDKC210 board (Exynos4210)
spitz                Sharp SL-C3000 (Spitz) PDA (PXA270)
sx1                  Siemens SX1 (OMAP310) V2
sx1-v1               Siemens SX1 (OMAP310) V1
terrier              Sharp SL-C3200 (Terrier) PDA (PXA270)
tosa                 Sharp SL-6000 (Tosa) PDA (PXA255)
verdex               Gumstix Verdex (PXA270)
versatileab          ARM Versatile/AB (ARM926EJ-S)
versatilepb          ARM Versatile/PB (ARM926EJ-S)
vexpress-a15         ARM Versatile Express for Cortex-A15
vexpress-a9          ARM Versatile Express for Cortex-A9
virt                 ARM Virtual Machine
xilinx-zynq-a9       Xilinx Zynq Platform Baseboard for Cortex-A9
z2                   Zipit Z2 (PXA27x)
odroid@odroid:~/lab12$
```

9. Some benefits of virtualization include: saving money on IT costs (fewer physical servers), faster recovery time (since you can build new virtual instances) and

testing on different virtual environments. Some benefits of emulation include: faster execution speeds and design capacity because not limited by hardware, finding bugs by using as a testbed, replacing FPGA prototyping systems, and accelerating time to market.

10. Yes, it is possible to have nested virtualization in some scenarios. Full virtualization is virtualization where the guest OS is unaware that it is in a virtualized environment, so the guest thinks that it is running on actual hardware and makes commands to hardware when the hardware is actually virtualized. The hypervisor captures the calls to hardware. Paravirtualization is where the guest OS knows it is a guest and has custom drivers that don't issue hardware commands to hardware but directly to the hypervisor. Hardware virtualization is where the hardware has special instructions to help the virtualization of hardware. Some special instructions could allow virtual contexts so a guest OS could execute privileged instructions directly on the processor without affecting the host. The hypervisor accesses the virtual contexts in hardware virtualization through a higher privileged ring that only hypervisors have access to.