ECE 8400 / ENGI 9875 Lab 5: Bootstrapping

Tousif Habib, Mohammed Shakeel

TOTAL POINTS

54 / 60

QUESTION 1

Prelab: Altera SoC boot overview 8 pts

1.1 HPS 1 / 1

✓ **- 0 pts** *Correct*

- 0.5 pts Incomplete definition of Hard Processor System

- 1 pts Failed in defining the hard processor system (HPS)

1.2 HPS boot process 2 / 3

- 0 pts Correct

- 1 pts Poor description on hardware limitations

- 1 pts Not explained about multi and single-staged boot

- 2 pts Missing a review on boot procedure and different stages

✓ **- 1 pts** *True, but the explanation about the procedure and stages is not complete.*

1.3 Cold vs warm boot 1 / 1

✓ **- 0 pts** *Correct*

1.4 Offset 0x40 0 / 1

- 0 pts Correct

✓ **- 1 pts** *0x40 contains the validation word*

1.5 Preloader image size 1 / 1

✓ **- 0 pts** *Correct*

- 0.5 pts Correct but not complete

- 1 pts The maximum size if 60K plus the header which is 4k. So, 64K

1.6 Boot ROM 0 / 1

- 0 pts Correct

✓ **- 1 pts** *Poor explanation. Failed to find where at which the boot ROM finds the address of the preloader image's entry point to start*

- 0.5 pts Not clear about the entry point and its address which is usually located at 0xffff004C

- 0.5 pts Partially true. Boot ROM jumps to the address of the entry point which is usually located ate 0xFFFF004C

QUESTION 2

Prelab: disk partitions 6 pts

2.1 FAT partition IDs 1 / 1

✓ **- 0 pts** *Correct*

2.2 LBA vs CHS 2 / 2

✓ **- 0 pts** *Correct*

- 0.5 pts Not explained why LBA is more used today

- 1 pts LBA is used today as it's more efficient and flexible in addressing, also can address larger disks than CHS

- 1 pts Incomplete with explaining each

addressing method

2.3 Linux partition IDs 1 / 1

✓ **- 0 pts** *Correct*

2.4 FreeBSD slice partition ID 1 / 1

✓ **- 0 pts** *Correct*

- 1 pts FreeBSD slice Partition ID is A5 in base-10, 165 in base-16

2.5 Partition ID 0xA2 0 / 1

- 0 pts Correct

✓ **- 1 pts** *The partition is for hard processor system (HPS) ARMP preloader image*

QUESTION 3

Prelab: disk and file inspection tools 14 pts

3.1 dd 2 / 2

✓ **- 0 pts** *Correct*

3.2 lsblk 2 / 2

✓ **- 0 pts** *Correct*

3.3 hexdump 6 / 6

✓ **- 0 pts** *Correct*

3.4 objdump 4 / 4

✓ **- 0 pts** *Correct*

- 1 pts The value stored in r3 is 8894

QUESTION 4

Prelab: interrupt handling 3 pts

4.1 Number of interrupt handlers 1 / 1

✓ **- 0 pts** *Correct*

- 1 pts There are 7 interrupt handlers

4.2 CPSR mode bits 2 / 2

✓ **- 0 pts** *Correct*

- 1 pts Mention the modes, like which one is set to supervisor mode, and what about other modes?

- 0.5 pts For reset, CPSR mode bits would be 10011

QUESTION 5

Prelab: serial communication 7 pts

5.1 UART 1 / 1

✓ **- 0 pts** *Correct*

5.2 cu(1) 1 / 1

✓ **- 0 pts** *Correct*

5.3 Baud rate 1 / 1

✓ **- 0 pts** *Correct*

5.4 Device 1 / 1

✓ **- 0 pts** *Correct*

5.5 Command line 1 / 1

✓ **- 0 pts** *Correct*

5.6 DE1-SoC baud rate 1 / 1

✓ **- 0 pts** *Correct*

- 1 pts It'd be 115200

5.7 Closing a connection 1 / 1

✓ **- 0 pts** *Correct*

- 1 pts It'd be "~"

QUESTION 6

Examine the MicroSD card 9 pts

6.1 lsblk 2 / 2

✓ **- 0 pts** *Correct*

6.2 lsblk details 2 / 2

✓ **- 0 pts** *Correct*

6.3 fdisk 3 / 3

✓ **- 0 pts** *Correct*

6.4 Extract preloader image 2 / 2

✓ **- 0 pts** *Correct*

QUESTION 7

Machine code 4 pts

7.1 hexdump 1 / 1

✓ **- 0 pts** *Correct*

7.2 Bytes at offset 0x40 0 / 1

- 0 pts Correct

✓ **- 1 pts** *The value is different because of little-endian format*

7.3 Entry point 1 / 2

- 0 pts Correct

✓ **- 1 pts** *The value is invalid because it's referring to an out of bound address*

QUESTION 8

Disassembly 9 pts

8.1 objdump failure 1 / 1

✓ **- 0 pts** *Correct*

8.2 objdump --target 2 / 2

✓ **- 0 pts** *Correct*

- 1 pts Missing evidence of the work

8.3 objdump failure (again) 1 / 1

✓ **- 0 pts** *Correct*

- 1 pts Not answered/completed

8.4 Disassemble the preloader 3 / 3

✓ **- 0 pts** *Correct*

- 1 pts The binary is not disassembled completely

- 3 pts Failed to finish the section

8.5 Instructions in exception vector table 2 / 2

✓ **- 0 pts** *Correct*

- 2 pts Not answered

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

1.1 HPS **1 / 1**

✓ **- 0 pts** *Correct*

- 0.5 pts Incomplete definition of Hard Processor System

- 1 pts Failed in defining the hard processor system (HPS)

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

1.2 HPS boot process **2 / 3**

- **0 pts** Correct
- **1 pts** Poor description on hardware limitations
- **1 pts** Not explained about multi and single-staged boot
- **2 pts** Missing a review on boot procedure and different stages

✓ **- 1 pts** *True, but the explanation about the procedure and stages is not complete.*

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

1.3 Cold vs warm boot **1 / 1**

✓ **- 0 pts** *Correct*

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

1.4 Offset 0x40 **0 / 1**

- 0 pts Correct

✓ **- 1 pts** *0x40 contains the validation word*

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

1.5 Preloader image size **1 / 1**

✓ **- 0 pts** *Correct*

- 0.5 pts Correct but not complete

- 1 pts The maximum size if 60K plus the header which is 4k. So, 64K

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

1.6 Boot ROM **0 / 1**

- 0 pts Correct

✓ **- 1 pts** *Poor explanation. Failed to find where at which the boot ROM finds the address of the preloader image's entry point to start*

- 0.5 pts Not clear about the entry point and its address which is usually located at 0xffff004C

- 0.5 pts Partially true. Boot ROM jumps to the address of the entry point which is usually located ate 0xFFFF004C

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

2.1 FAT partition IDs **1 / 1**

✓ **- 0 pts** *Correct*

Prelab

Altera SoC boot overview

1. An "HPS" refers to the Hard Processor System, which is an embedded processor system that includes an ARM Cortex-A9 processor, peripheral controllers, and memory interfaces.

2. The HPS boot process in Cyclone-V SoC boards is a multi-stage process due to hardware limitations. The Boot ROM is limited in size and cannot accommodate the entire boot process. Therefore, it loads the preloader, which performs additional initialization tasks before loading the main bootloader or application.

3. When the Cyclone-V system is turned on from a state where it was completely shut down, it goes through a cold boot process. This process includes initializing all necessary components such as the boot ROM and loading the operating system into memory. In contrast, a warm boot involves restarting the system from a state where it was not completely shut down. This does not require re-initializing the ROM boot components and only the operating system needs to be restarted. The cold boot process for Cyclone-V, as described in its manual, includes loading the preloader, U-Boot Loader, Operating System and finally an application.

4. The address for the first stage of the bootloader is found at offset 0x40 in the preloader image.
5. It is 60kB in size. If a preloader image exceeds 60kB in size, it will occupy more than one block in the system's memory.
6. The boot ROM reads the value stored at offset 0x44 of the preloader image to determine the address to start running code from.

Disk Partitions

1. The partition IDs 06h, 08h, 11h, 14h, 24h, 56h, e6h, efh and f2h are used to represent different variations of the FAT file system.

2. LBA (Logical Block Addressing) and CHS (Cylinder-Head-Sector) are two different methods of addressing data on a storage device. CHS came first, and it uses a tuple of cylinder, head, and sector numbers to specify a location. LBA came later and uses a linear addressing scheme that simplifies the process by using a single number to address data. LBA is more commonly used today because it allows for larger storage devices and is easier to work with.

2.2 LBA vs CHS **2 / 2**

✓ **- 0 pts** *Correct*

- 0.5 pts Not explained why LBA is more used today

- 1 pts LBA is used today as it's more efficient and flexible in addressing, also can address larger disks than CHS

- 1 pts Incomplete with explaining each addressing method

3. The partition ID for modern Linux filesystems, such as ext2, ext3, ext4, is 0x83 (131 in base-10).

4. The partition ID for a FreeBSD slice is 0xA5 (165 in base-10).

5. The partition will store the metadata for the logical volume manager. This metadata contains important information about the disk data, such as the location and size of the physical volume. The partition plays a crucial role in managing the storage of the system.

Disk and File Inspection Tools

1. A. dd: This tool copies and converts files or devices, allowing for specified input and output block sizes, and optional conversions.
   To copy 32 KiB from one file to another, the below command can be used:

```
dd if=input_file of=output_file bs=32K count=64
```

2. lsblk: This tool lists information about all available or specified block devices.
   To retrieve the requested details, pass the following options to lsblk:

```
lsblk -o NAME,PATH,LOG-SEC,PTTYPE,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
```

3. a) Command Output:

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 28 00 01 00 00 00  00 00 00 00 00 00 00 00  |..(.............|
00000020  74 01 00 00 00 00 00 05  34 00 00 00 00 00 28 00  |t.......4.....(.|
00000030  0c 00 09 00 80 b5 00 af  40 f2 00 00 c0 f2 00 00  |........@.......|
00000040  42 f2 be 21 ff f7 fe ff  00 23 18 46 80 bd 00 bf  |B..!.....#.F....|
00000050  43 6f 6e 67 72 61 74 75  6c 61 74 69 6f 6e 73 20  |Congratulations |
00000060  6f 6e 20 63 6f 6d 70 6c  65 74 69 6e 67 20 45 4e  |on completing EN|
00000070  47 49 20 25 64 21 0a 00  00 47 43 43 3a 20 28 63  |GI %d!...GCC: (c|
00000080  72 6f 73 73 74 6f 6f 6c  2d 4e 47 20 6c 69 6e 61  |rosstool-NG lina|
00000090  72 6f 2d 31 2e 31 33 2e  31 2d 34 2e 38 2d 32 30  |ro-1.13.1-4.8-20|
000000a0  31 34 2e 30 34 20 2d 20  4c 69 6e 61 72 6f 20 47  |14.04 - Linaro G|
000000b0  43 43 20 34 2e 38 2d 32  30 31 34 2e 30 34 29 20  |CC 4.8-2014.04) |
```

From Offset 0xb0: 43 43 20 34 2e 38 2d 32 30 31 34 2e 30 34 29 20 |CC 4.8-2014.04) |

b) After executing the command readelf -h 8894.o, the following information was obtained:

2.3 Linux partition IDs **1 / 1**

✓ **- 0 pts** *Correct*

3. The partition ID for modern Linux filesystems, such as ext2, ext3, ext4, is 0x83 (131 in base-10).

4. The partition ID for a FreeBSD slice is 0xA5 (165 in base-10).

5. The partition will store the metadata for the logical volume manager. This metadata contains important information about the disk data, such as the location and size of the physical volume. The partition plays a crucial role in managing the storage of the system.

Disk and File Inspection Tools

1. A. dd: This tool copies and converts files or devices, allowing for specified input and output block sizes, and optional conversions.
   To copy 32 KiB from one file to another, the below command can be used:

```
dd if=input_file of=output_file bs=32K count=64
```

2. lsblk: This tool lists information about all available or specified block devices.
   To retrieve the requested details, pass the following options to lsblk:

```
lsblk -o NAME,PATH,LOG-SEC,PTTYPE,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
```

3. a) Command Output:

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 28 00 01 00 00 00  00 00 00 00 00 00 00 00  |..(.............|
00000020  74 01 00 00 00 00 00 05  34 00 00 00 00 00 28 00  |t.......4.....(.|
00000030  0c 00 09 00 80 b5 00 af  40 f2 00 00 c0 f2 00 00  |........@.......|
00000040  42 f2 be 21 ff f7 fe ff  00 23 18 46 80 bd 00 bf  |B..!.....#.F....|
00000050  43 6f 6e 67 72 61 74 75  6c 61 74 69 6f 6e 73 20  |Congratulations |
00000060  6f 6e 20 63 6f 6d 70 6c  65 74 69 6e 67 20 45 4e  |on completing EN|
00000070  47 49 20 25 64 21 0a 00  00 47 43 43 3a 20 28 63  |GI %d!...GCC: (c|
00000080  72 6f 73 73 74 6f 6f 6c  2d 4e 47 20 6c 69 6e 61  |rosstool-NG lina|
00000090  72 6f 2d 31 2e 31 33 2e  31 2d 34 2e 38 2d 32 30  |ro-1.13.1-4.8-20|
000000a0  31 34 2e 30 34 20 2d 20  4c 69 6e 61 72 6f 20 47  |14.04 - Linaro G|
000000b0  43 43 20 34 2e 38 2d 32  30 31 34 2e 30 34 29 20  |CC 4.8-2014.04) |
```

From Offset 0xb0: 43 43 20 34 2e 38 2d 32 30 31 34 2e 30 34 29 20 |CC 4.8-2014.04) |

b) After executing the command readelf -h 8894.o, the following information was obtained:

2.4 FreeBSD slice partition ID **1 / 1**

✓ **- 0 pts** *Correct*

- 1 pts FreeBSD slice Partition ID is A5 in base-10, 165 in base-16

3. The partition ID for modern Linux filesystems, such as ext2, ext3, ext4, is 0x83 (131 in base-10).

4. The partition ID for a FreeBSD slice is 0xA5 (165 in base-10).

5. The partition will store the metadata for the logical volume manager. This metadata contains important information about the disk data, such as the location and size of the physical volume. The partition plays a crucial role in managing the storage of the system.

Disk and File Inspection Tools

1. A. dd: This tool copies and converts files or devices, allowing for specified input and output block sizes, and optional conversions.
   To copy 32 KiB from one file to another, the below command can be used:

```
dd if=input_file of=output_file bs=32K count=64
```

2. lsblk: This tool lists information about all available or specified block devices.
   To retrieve the requested details, pass the following options to lsblk:

```
lsblk -o NAME,PATH,LOG-SEC,PTTYPE,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
```

3. a) Command Output:

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 28 00 01 00 00 00  00 00 00 00 00 00 00 00  |..(.............|
00000020  74 01 00 00 00 00 00 05  34 00 00 00 00 00 28 00  |t.......4.....(.|
00000030  0c 00 09 00 80 b5 00 af  40 f2 00 00 c0 f2 00 00  |........@.......|
00000040  42 f2 be 21 ff f7 fe ff  00 23 18 46 80 bd 00 bf  |B..!.....#.F....|
00000050  43 6f 6e 67 72 61 74 75  6c 61 74 69 6f 6e 73 20  |Congratulations |
00000060  6f 6e 20 63 6f 6d 70 6c  65 74 69 6e 67 20 45 4e  |on completing EN|
00000070  47 49 20 25 64 21 0a 00  00 47 43 43 3a 20 28 63  |GI %d!...GCC: (c|
00000080  72 6f 73 73 74 6f 6f 6c  2d 4e 47 20 6c 69 6e 61  |rosstool-NG lina|
00000090  72 6f 2d 31 2e 31 33 2e  31 2d 34 2e 38 2d 32 30  |ro-1.13.1-4.8-20|
000000a0  31 34 2e 30 34 20 2d 20  4c 69 6e 61 72 6f 20 47  |14.04 - Linaro G|
000000b0  43 43 20 34 2e 38 2d 32  30 31 34 2e 30 34 29 20  |CC 4.8-2014.04) |
```

From Offset 0xb0: 43 43 20 34 2e 38 2d 32 30 31 34 2e 30 34 29 20 |CC 4.8-2014.04) |

b) After executing the command readelf -h 8894.o, the following information was obtained:

2.5 Partition ID 0xA2 **0 / 1**

- 0 pts Correct

✓ **- 1 pts** *The partition is for hard processor system (HPS) ARMP preloader image*

3. The partition ID for modern Linux filesystems, such as ext2, ext3, ext4, is 0x83 (131 in base-10).

4. The partition ID for a FreeBSD slice is 0xA5 (165 in base-10).

5. The partition will store the metadata for the logical volume manager. This metadata contains important information about the disk data, such as the location and size of the physical volume. The partition plays a crucial role in managing the storage of the system.

Disk and File Inspection Tools

1. A. dd: This tool copies and converts files or devices, allowing for specified input and output block sizes, and optional conversions.
   To copy 32 KiB from one file to another, the below command can be used:

```
dd if=input_file of=output_file bs=32K count=64
```

2. lsblk: This tool lists information about all available or specified block devices.
   To retrieve the requested details, pass the following options to lsblk:

```
lsblk -o NAME,PATH,LOG-SEC,PTTYPE,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
```

3. a) Command Output:

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 28 00 01 00 00 00  00 00 00 00 00 00 00 00  |..(.............|
00000020  74 01 00 00 00 00 00 05  34 00 00 00 00 00 28 00  |t.......4.....(.|
00000030  0c 00 09 00 80 b5 00 af  40 f2 00 00 c0 f2 00 00  |........@.......|
00000040  42 f2 be 21 ff f7 fe ff  00 23 18 46 80 bd 00 bf  |B..!.....#.F....|
00000050  43 6f 6e 67 72 61 74 75  6c 61 74 69 6f 6e 73 20  |Congratulations |
00000060  6f 6e 20 63 6f 6d 70 6c  65 74 69 6e 67 20 45 4e  |on completing EN|
00000070  47 49 20 25 64 21 0a 00  00 47 43 43 3a 20 28 63  |GI %d!...GCC: (c|
00000080  72 6f 73 73 74 6f 6f 6c  2d 4e 47 20 6c 69 6e 61  |rosstool-NG lina|
00000090  72 6f 2d 31 2e 31 33 2e  31 2d 34 2e 38 2d 32 30  |ro-1.13.1-4.8-20|
000000a0  31 34 2e 30 34 20 2d 20  4c 69 6e 61 72 6f 20 47  |14.04 - Linaro G|
000000b0  43 43 20 34 2e 38 2d 32  30 31 34 2e 30 34 29 20  |CC 4.8-2014.04) |
```

From Offset 0xb0: 43 43 20 34 2e 38 2d 32 30 31 34 2e 30 34 29 20 |CC 4.8-2014.04) |

b) After executing the command readelf -h 8894.o, the following information was obtained:

3.1 dd **2 / 2**

✓ **- 0 pts** *Correct*

3. The partition ID for modern Linux filesystems, such as ext2, ext3, ext4, is 0x83 (131 in base-10).

4. The partition ID for a FreeBSD slice is 0xA5 (165 in base-10).

5. The partition will store the metadata for the logical volume manager. This metadata contains important information about the disk data, such as the location and size of the physical volume. The partition plays a crucial role in managing the storage of the system.

Disk and File Inspection Tools

1. A. dd: This tool copies and converts files or devices, allowing for specified input and output block sizes, and optional conversions.
   To copy 32 KiB from one file to another, the below command can be used:

```
dd if=input_file of=output_file bs=32K count=64
```

2. lsblk: This tool lists information about all available or specified block devices.
   To retrieve the requested details, pass the following options to lsblk:

```
lsblk -o NAME,PATH,LOG-SEC,PTTYPE,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
```

3. a) Command Output:

```
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 28 00 01 00 00 00  00 00 00 00 00 00 00 00  |..(.............|
00000020  74 01 00 00 00 00 00 05  34 00 00 00 00 00 28 00  |t.......4.....(.|
00000030  0c 00 09 00 80 b5 00 af  40 f2 00 00 c0 f2 00 00  |........@.......|
00000040  42 f2 be 21 ff f7 fe ff  00 23 18 46 80 bd 00 bf  |B..!.....#.F....|
00000050  43 6f 6e 67 72 61 74 75  6c 61 74 69 6f 6e 73 20  |Congratulations |
00000060  6f 6e 20 63 6f 6d 70 6c  65 74 69 6e 67 20 45 4e  |on completing EN|
00000070  47 49 20 25 64 21 0a 00  00 47 43 43 3a 20 28 63  |GI %d!...GCC: (c|
00000080  72 6f 73 73 74 6f 6f 6c  2d 4e 47 20 6c 69 6e 61  |rosstool-NG lina|
00000090  72 6f 2d 31 2e 31 33 2e  31 2d 34 2e 38 2d 32 30  |ro-1.13.1-4.8-20|
000000a0  31 34 2e 30 34 20 2d 20  4c 69 6e 61 72 6f 20 47  |14.04 - Linaro G|
000000b0  43 43 20 34 2e 38 2d 32  30 31 34 2e 30 34 29 20  |CC 4.8-2014.04) |
```

From Offset 0xb0: 43 43 20 34 2e 38 2d 32 30 31 34 2e 30 34 29 20 |CC 4.8-2014.04) |

b) After executing the command readelf -h 8894.o, the following information was obtained:

3.2 lsblk **2 / 2**

✓ **- 0 pts** *Correct*

1. The ELF version is 1 (0x01).
2. The class is Elf32, indicating that it is a 32-bit file.
3. The data row indicates that the file is compiled for a two's complement little-endian machine.
4. The type row shows that the file is of type relocatable.

Console output:

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2&apos;s complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          372 (bytes into file)
  Flags:                             0x5000000, Version5 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         12
  Section header string table index: 9
```

5. The presence of the ASCII text "Congratulations on completing ENGI %d!" at offset 0x50 indicates that the program is designed to output a message.

4. a) The objdump -s command is used to display the full contents of all sections requested in a binary file.

b) The objdump -d command is used to disassemble the binary code in the specified sections of an object file.

c) The value stored in r3 is loaded using the instruction at offset 0x28, which is movw r3, #8894. This instruction moves the immediate value 0x22BE (8894 in decimal) into the r3 register.

3.3 hexdump **6 / 6**

✓ **- 0 pts** *Correct*

1. The ELF version is 1 (0x01).
2. The class is Elf32, indicating that it is a 32-bit file.
3. The data row indicates that the file is compiled for a two's complement little-endian machine.
4. The type row shows that the file is of type relocatable.

Console output:

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2&apos;s complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           ARM
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          372 (bytes into file)
  Flags:                             0x5000000, Version5 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         12
  Section header string table index: 9
```

5. The presence of the ASCII text "Congratulations on completing ENGI %d!" at offset 0x50 indicates that the program is designed to output a message.

4. a) The objdump -s command is used to display the full contents of all sections requested in a binary file.

b) The objdump -d command is used to disassemble the binary code in the specified sections of an object file.

c) The value stored in r3 is loaded using the instruction at offset 0x28, which is movw r3, #8894. This instruction moves the immediate value 0x22BE (8894 in decimal) into the r3 register.

3.4 objdump **4 / 4**

✓ **- 0 pts** *Correct*

- 1 pts The value stored in r3 is 8894

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

4.1 Number of interrupt handlers **1 / 1**

✓ **- 0 pts** *Correct*

- 1 pts There are 7 interrupt handlers

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

4.2 CPSR mode bits **2 / 2**

✓ **- 0 pts** *Correct*

- 1 pts Mention the modes, like which one is set to supervisor mode, and what about other modes?

- 0.5 pts For reset, CPSR mode bits would be 10011

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

5.1 UART **1 / 1**

✓ **- 0 pts** *Correct*

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

5.2 cu(1) **1 / 1**

✓ **- 0 pts** *Correct*

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

5.3 Baud rate **1 / 1**

✓ **- 0 pts** *Correct*

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

5.4 Device **1 / 1**

✓ **- 0 pts** *Correct*

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

5.5 Command line **1 / 1**

✓ **- 0 pts** *Correct*

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

5.6 DE1-SoC baud rate **1 / 1**

✓ **- 0 pts** *Correct*

- 1 pts It'd be 115200

Interrupt Handling

1. There are 7 interrupt handlers: Reset, Unimplemented instruction, Undefined, Software interrupt, Data access, Instruction access violation, IRQ, FIQ
2. A. Reset: The mode bits will be set to Supervisor mode.
   B. Software Interrupt: The mode bits will be set to Supervisor mode.
   C. Accessing an instruction at non-existent memory: The mode bits will be set to Abort mode.
   D. External Interrupt Request (IRQ): The mode bits will be set to IRQ mode.

Serial Communication

1. UART (Universal Asynchronous Receiver/Transmitter) is a hardware communication protocol that translates data between parallel and serial forms. A UART allows for serial communication between devices, such as microcontrollers and computers. I believe this would be a useful reference for someone in the future (maybe?): https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

2. The cu(1) is a command-line program that is used for establishing and managing connections to remote systems over serial lines.

3. The command-line option that controls the baud rate is -s

4. The command-line option that controls the device being called is -l followed by the device name

5. To connect to a system's only USB serial port at 19,200 baud, you would use the following command line:

```
cu -l /dev/ttyUSB0 -s 19200
```

6. According to the DE1-SoC Getting Started Guide, the baud rate that should be used to communicate serially with the board is 115200 baud.
7. To close the connection cleanly: ~.

5.7 Closing a connection **1 / 1**

✓ **- 0 pts** *Correct*

- 1 pts It'd be "~"

Procedure

Examine the SD card

Without USB card reader:

```
msshakeel@slbncsf2112pc46:~$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0    7:0    0     4K  1 loop /snap/bare/5
loop1    7:1    0 218.4M  1 loop /snap/gnome-3-34-1804/90
loop2    7:2    0  55.1M  1 loop /snap/cups/836
loop3    7:3    0 164.8M  1 loop /snap/gnome-3-28-1804/194
loop4    7:4    0    46M  1 loop /snap/snap-store/638
loop5    7:5    0  63.3M  1 loop /snap/core20/1822
loop6    7:6    0  54.2M  1 loop /snap/snap-store/558
loop7    7:7    0 148.5M  1 loop /snap/chromium/2406
loop8    7:8    0   219M  1 loop /snap/gnome-3-34-1804/77
loop9    7:9    0 314.6M  1 loop /snap/eclipse/66
loop11   7:11   0  55.6M  1 loop /snap/core18/2721
loop12   7:12   0 164.8M  1 loop /snap/gnome-3-28-1804/161
loop13   7:13   0 349.7M  1 loop /snap/gnome-3-38-2004/137
loop14   7:14   0 148.5M  1 loop /snap/chromium/2415
loop16   7:16   0 307.8M  1 loop /snap/eclipse/64
loop17   7:17   0  49.8M  1 loop /snap/snapd/17950
loop18   7:18   0    55M  1 loop /snap/cups/872
loop19   7:19   0  91.7M  1 loop /snap/gtk-common-themes/1535
loop20   7:20   0  81.3M  1 loop /snap/gtk-common-themes/1534
loop21   7:21   0 346.3M  1 loop /snap/gnome-3-38-2004/119
loop22   7:22   0  55.6M  1 loop /snap/core18/2679
loop23   7:23   0  49.9M  1 loop /snap/snapd/18596
loop24   7:24   0  63.3M  1 loop /snap/core20/1852
sda      8:0    0   120G  0 disk
├─sda1   8:1    0    10M  0 part
└─sda2   8:2    0   120G  0 part /
```

Sda disk with two partitions

With USB card reader:

```
msshakeel@slbncsf2112pc46:~$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0    7:0    0     4K  1 loop /snap/bare/5
loop1    7:1    0 218.4M  1 loop /snap/gnome-3-34-1804/90
loop2    7:2    0  55.1M  1 loop /snap/cups/836
loop3    7:3    0 164.8M  1 loop /snap/gnome-3-28-1804/194
loop4    7:4    0    46M  1 loop /snap/snap-store/638
loop5    7:5    0  63.3M  1 loop /snap/core20/1822
loop6    7:6    0  54.2M  1 loop /snap/snap-store/558
loop7    7:7    0 148.5M  1 loop /snap/chromium/2406
loop8    7:8    0   219M  1 loop /snap/gnome-3-34-1804/77
loop9    7:9    0 314.6M  1 loop /snap/eclipse/66
loop11   7:11   0  55.6M  1 loop /snap/core18/2721
loop12   7:12   0 164.8M  1 loop /snap/gnome-3-28-1804/161
loop13   7:13   0 349.7M  1 loop /snap/gnome-3-38-2004/137
loop14   7:14   0 148.5M  1 loop /snap/chromium/2415
loop16   7:16   0 307.8M  1 loop /snap/eclipse/64
loop17   7:17   0  49.8M  1 loop /snap/snapd/17950
loop18   7:18   0    55M  1 loop /snap/cups/872
loop19   7:19   0  91.7M  1 loop /snap/gtk-common-themes/1535
loop20   7:20   0  81.3M  1 loop /snap/gtk-common-themes/1534
loop21   7:21   0 346.3M  1 loop /snap/gnome-3-38-2004/119
loop22   7:22   0  55.6M  1 loop /snap/core18/2679
loop23   7:23   0  49.9M  1 loop /snap/snapd/18596
loop24   7:24   0  63.3M  1 loop /snap/core20/1852
sda      8:0    0   120G  0 disk
├─sda1   8:1    0    10M  0 part
└─sda2   8:2    0   120G  0 part /
sdb      8:16   1   1.9G  0 disk
└─sdb1   8:17   1   1.9G  0 part /media/msshakeel/disk
```

New disk, sdb with 1 partition. Partition is 1.9 G in size

Partition inspected:

```
msshakeel@slbncsf2112pc46:~$ lsblk -bPo NAME,PATH,PHY-SEC,SERIAL,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
NAME="loop0" PATH="/dev/loop0" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/bare/5"
NAME="loop1" PATH="/dev/loop1" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-34-1804/90"
NAME="loop2" PATH="/dev/loop2" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/cups/836"
NAME="loop3" PATH="/dev/loop3" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-28-1804/194"
NAME="loop4" PATH="/dev/loop4" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snap-store/638"
NAME="loop5" PATH="/dev/loop5" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core20/1822"
NAME="loop6" PATH="/dev/loop6" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snap-store/558"
NAME="loop7" PATH="/dev/loop7" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/chromium/2406"
NAME="loop8" PATH="/dev/loop8" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-34-1804/77"
NAME="loop9" PATH="/dev/loop9" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/eclipse/66"
NAME="loop11" PATH="/dev/loop11" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core18/2721"
NAME="loop12" PATH="/dev/loop12" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-28-1804/161"
NAME="loop13" PATH="/dev/loop13" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-38-2004/137"
NAME="loop14" PATH="/dev/loop14" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/chromium/2415"
NAME="loop16" PATH="/dev/loop16" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/eclipse/64"
NAME="loop17" PATH="/dev/loop17" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snapd/17950"
NAME="loop18" PATH="/dev/loop18" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/cups/872"
NAME="loop19" PATH="/dev/loop19" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gtk-common-themes/1535"
NAME="loop20" PATH="/dev/loop20" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gtk-common-themes/1534"
NAME="loop21" PATH="/dev/loop21" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-38-2004/119"
NAME="loop22" PATH="/dev/loop22" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core18/2679"
NAME="loop23" PATH="/dev/loop23" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snapd/18596"
NAME="loop24" PATH="/dev/loop24" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core20/1852"
NAME="sda" PATH="/dev/sda" PHY-SEC="512" SERIAL="4fc1a7b2-c90f-4fe3-88fc-e96de4c6348a" PARTTYPE="" PARTLABEL="" FSTYPE="" MOUNTPOINT=""
NAME="sda1" PATH="/dev/sda1" PHY-SEC="512" SERIAL="" PARTTYPE="c12a7328-f81f-11d2-ba4b-00a0c93ec93b" PARTLABEL="BOOT" FSTYPE="" MOUNTPOINT=""
NAME="sda2" PATH="/dev/sda2" PHY-SEC="512" SERIAL="" PARTTYPE="0fc63daf-8483-4772-8e79-3d69d8477de4" PARTLABEL="ROOT" FSTYPE="ext4" MOUNTPOINT="/"
NAME="sdb" PATH="/dev/sdb" PHY-SEC="512" SERIAL="000000001536" PARTTYPE="" PARTLABEL="" FSTYPE="" MOUNTPOINT=""
NAME="sdb1" PATH="/dev/sdb1" PHY-SEC="512" SERIAL="" PARTTYPE="0x6" PARTLABEL="" FSTYPE="vfat" MOUNTPOINT="/media/msshakeel/disk"
```

After running fsdisk command:

```
msshakeel@slbncsf2112pc46:~$ fdisk -l ~p15jra/bluedot.img
Disk /users/labnet5/fs1/p15jra/bluedot.img: 14.45 GiB, 15502147584 bytes, 30277632 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x0005a3a8

Device                                  Boot   Start      End  Sectors   Size Id Type
/users/labnet5/fs1/p15jra/bluedot.img1        2121728  3799447  1677720 819.2M  b W95 FAT32
/users/labnet5/fs1/p15jra/bluedot.img2        3801088 30277631 26476544  12.6G 83 Linux
/users/labnet5/fs1/p15jra/bluedot.img3           2048     4095     2048     1M a2 unknown

Partition table entries are not in disk order.
```

There are 3 partitions, bluedot.img1, bluedot.img2, bluedot.img3

Bluedot.img1 is of size 819.2M

Bluedot.img2 is of size 12.6G

Bluedot.img3 is of size 1M

Img3 is the preloader because the Id is a2 from our prelab.

extracting :

```
msshakeel@slbncsf2112pc46:~$ man dd
msshakeel@slbncsf2112pc46:~$ dd if=~p15jra/bluedot.img of=preloader.img bs=512 skip=2048 count=120
120+0 records in
120+0 records out
61440 bytes (61 kB, 60 KiB) copied, 0.0155691 s, 3.9 MB/s
```

6.1 lsblk **2 / 2**

✓ **- 0 pts** *Correct*

Procedure

Examine the SD card

Without USB card reader:

```
msshakeel@slbncsf2112pc46:~$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0    7:0    0     4K  1 loop /snap/bare/5
loop1    7:1    0 218.4M  1 loop /snap/gnome-3-34-1804/90
loop2    7:2    0  55.1M  1 loop /snap/cups/836
loop3    7:3    0 164.8M  1 loop /snap/gnome-3-28-1804/194
loop4    7:4    0    46M  1 loop /snap/snap-store/638
loop5    7:5    0  63.3M  1 loop /snap/core20/1822
loop6    7:6    0  54.2M  1 loop /snap/snap-store/558
loop7    7:7    0 148.5M  1 loop /snap/chromium/2406
loop8    7:8    0   219M  1 loop /snap/gnome-3-34-1804/77
loop9    7:9    0 314.6M  1 loop /snap/eclipse/66
loop11   7:11   0  55.6M  1 loop /snap/core18/2721
loop12   7:12   0 164.8M  1 loop /snap/gnome-3-28-1804/161
loop13   7:13   0 349.7M  1 loop /snap/gnome-3-38-2004/137
loop14   7:14   0 148.5M  1 loop /snap/chromium/2415
loop16   7:16   0 307.8M  1 loop /snap/eclipse/64
loop17   7:17   0  49.8M  1 loop /snap/snapd/17950
loop18   7:18   0    55M  1 loop /snap/cups/872
loop19   7:19   0  91.7M  1 loop /snap/gtk-common-themes/1535
loop20   7:20   0  81.3M  1 loop /snap/gtk-common-themes/1534
loop21   7:21   0 346.3M  1 loop /snap/gnome-3-38-2004/119
loop22   7:22   0  55.6M  1 loop /snap/core18/2679
loop23   7:23   0  49.9M  1 loop /snap/snapd/18596
loop24   7:24   0  63.3M  1 loop /snap/core20/1852
sda      8:0    0   120G  0 disk
├─sda1   8:1    0    10M  0 part
└─sda2   8:2    0   120G  0 part /
```

Sda disk with two partitions

With USB card reader:

```
msshakeel@slbncsf2112pc46:~$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
loop0    7:0    0     4K  1 loop /snap/bare/5
loop1    7:1    0 218.4M  1 loop /snap/gnome-3-34-1804/90
loop2    7:2    0  55.1M  1 loop /snap/cups/836
loop3    7:3    0 164.8M  1 loop /snap/gnome-3-28-1804/194
loop4    7:4    0    46M  1 loop /snap/snap-store/638
loop5    7:5    0  63.3M  1 loop /snap/core20/1822
loop6    7:6    0  54.2M  1 loop /snap/snap-store/558
loop7    7:7    0 148.5M  1 loop /snap/chromium/2406
loop8    7:8    0   219M  1 loop /snap/gnome-3-34-1804/77
loop9    7:9    0 314.6M  1 loop /snap/eclipse/66
loop11   7:11   0  55.6M  1 loop /snap/core18/2721
loop12   7:12   0 164.8M  1 loop /snap/gnome-3-28-1804/161
loop13   7:13   0 349.7M  1 loop /snap/gnome-3-38-2004/137
loop14   7:14   0 148.5M  1 loop /snap/chromium/2415
loop16   7:16   0 307.8M  1 loop /snap/eclipse/64
loop17   7:17   0  49.8M  1 loop /snap/snapd/17950
loop18   7:18   0    55M  1 loop /snap/cups/872
loop19   7:19   0  91.7M  1 loop /snap/gtk-common-themes/1535
loop20   7:20   0  81.3M  1 loop /snap/gtk-common-themes/1534
loop21   7:21   0 346.3M  1 loop /snap/gnome-3-38-2004/119
loop22   7:22   0  55.6M  1 loop /snap/core18/2679
loop23   7:23   0  49.9M  1 loop /snap/snapd/18596
loop24   7:24   0  63.3M  1 loop /snap/core20/1852
sda      8:0    0   120G  0 disk
├─sda1   8:1    0    10M  0 part
└─sda2   8:2    0   120G  0 part /
sdb      8:16   1   1.9G  0 disk
└─sdb1   8:17   1   1.9G  0 part /media/msshakeel/disk
```

New disk, sdb with 1 partition. Partition is 1.9 G in size

Partition inspected:

```
msshakeel@slbncsf2112pc46:~$ lsblk -bPo NAME,PATH,PHY-SEC,SERIAL,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
NAME="loop0" PATH="/dev/loop0" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/bare/5"
NAME="loop1" PATH="/dev/loop1" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-34-1804/90"
NAME="loop2" PATH="/dev/loop2" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/cups/836"
NAME="loop3" PATH="/dev/loop3" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-28-1804/194"
NAME="loop4" PATH="/dev/loop4" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snap-store/638"
NAME="loop5" PATH="/dev/loop5" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core20/1822"
NAME="loop6" PATH="/dev/loop6" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snap-store/558"
NAME="loop7" PATH="/dev/loop7" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/chromium/2406"
NAME="loop8" PATH="/dev/loop8" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-34-1804/77"
NAME="loop9" PATH="/dev/loop9" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/eclipse/66"
NAME="loop11" PATH="/dev/loop11" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core18/2721"
NAME="loop12" PATH="/dev/loop12" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-28-1804/161"
NAME="loop13" PATH="/dev/loop13" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-38-2004/137"
NAME="loop14" PATH="/dev/loop14" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/chromium/2415"
NAME="loop16" PATH="/dev/loop16" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/eclipse/64"
NAME="loop17" PATH="/dev/loop17" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snapd/17950"
NAME="loop18" PATH="/dev/loop18" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/cups/872"
NAME="loop19" PATH="/dev/loop19" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gtk-common-themes/1535"
NAME="loop20" PATH="/dev/loop20" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gtk-common-themes/1534"
NAME="loop21" PATH="/dev/loop21" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-38-2004/119"
NAME="loop22" PATH="/dev/loop22" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core18/2679"
NAME="loop23" PATH="/dev/loop23" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snapd/18596"
NAME="loop24" PATH="/dev/loop24" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core20/1852"
NAME="sda" PATH="/dev/sda" PHY-SEC="512" SERIAL="4fc1a7b2-c90f-4fe3-88fc-e96de4c6348a" PARTTYPE="" PARTLABEL="" FSTYPE="" MOUNTPOINT=""
NAME="sda1" PATH="/dev/sda1" PHY-SEC="512" SERIAL="" PARTTYPE="c12a7328-f81f-11d2-ba4b-00a0c93ec93b" PARTLABEL="BOOT" FSTYPE="" MOUNTPOINT=""
NAME="sda2" PATH="/dev/sda2" PHY-SEC="512" SERIAL="" PARTTYPE="0fc63daf-8483-4772-8e79-3d69d8477de4" PARTLABEL="ROOT" FSTYPE="ext4" MOUNTPOINT="/"
NAME="sdb" PATH="/dev/sdb" PHY-SEC="512" SERIAL="000000001536" PARTTYPE="" PARTLABEL="" FSTYPE="" MOUNTPOINT=""
NAME="sdb1" PATH="/dev/sdb1" PHY-SEC="512" SERIAL="" PARTTYPE="0x6" PARTLABEL="" FSTYPE="vfat" MOUNTPOINT="/media/msshakeel/disk"
```

After running fsdisk command:

```
msshakeel@slbncsf2112pc46:~$ fdisk -l ~p15jra/bluedot.img
Disk /users/labnet5/fs1/p15jra/bluedot.img: 14.45 GiB, 15502147584 bytes, 30277632 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x0005a3a8

Device                                  Boot   Start      End  Sectors   Size Id Type
/users/labnet5/fs1/p15jra/bluedot.img1        2121728  3799447  1677720 819.2M  b W95 FAT32
/users/labnet5/fs1/p15jra/bluedot.img2        3801088 30277631 26476544  12.6G 83 Linux
/users/labnet5/fs1/p15jra/bluedot.img3           2048     4095     2048     1M a2 unknown

Partition table entries are not in disk order.
```

There are 3 partitions, bluedot.img1, bluedot.img2, bluedot.img3

Bluedot.img1 is of size 819.2M

Bluedot.img2 is of size 12.6G

Bluedot.img3 is of size 1M

Img3 is the preloader because the Id is a2 from our prelab.

extracting :

```
msshakeel@slbncsf2112pc46:~$ man dd
msshakeel@slbncsf2112pc46:~$ dd if=~p15jra/bluedot.img of=preloader.img bs=512 skip=2048 count=120
120+0 records in
120+0 records out
61440 bytes (61 kB, 60 KiB) copied, 0.0155691 s, 3.9 MB/s
```

6.2 lsblk details **2 / 2**

✓ **- 0 pts** *Correct*

6.3 fdisk **3 / 3**

✓ **- 0 pts** *Correct*

Partition inspected:

```
msshakeel@slbncsf2112pc46:~$ lsblk -bPo NAME,PATH,PHY-SEC,SERIAL,PARTTYPE,PARTLABEL,FSTYPE,MOUNTPOINT
NAME="loop0" PATH="/dev/loop0" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/bare/5"
NAME="loop1" PATH="/dev/loop1" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-34-1804/90"
NAME="loop2" PATH="/dev/loop2" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/cups/836"
NAME="loop3" PATH="/dev/loop3" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-28-1804/194"
NAME="loop4" PATH="/dev/loop4" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snap-store/638"
NAME="loop5" PATH="/dev/loop5" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core20/1822"
NAME="loop6" PATH="/dev/loop6" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snap-store/558"
NAME="loop7" PATH="/dev/loop7" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/chromium/2406"
NAME="loop8" PATH="/dev/loop8" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-34-1804/77"
NAME="loop9" PATH="/dev/loop9" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/eclipse/66"
NAME="loop11" PATH="/dev/loop11" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core18/2721"
NAME="loop12" PATH="/dev/loop12" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-28-1804/161"
NAME="loop13" PATH="/dev/loop13" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-38-2004/137"
NAME="loop14" PATH="/dev/loop14" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/chromium/2415"
NAME="loop16" PATH="/dev/loop16" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/eclipse/64"
NAME="loop17" PATH="/dev/loop17" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snapd/17950"
NAME="loop18" PATH="/dev/loop18" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/cups/872"
NAME="loop19" PATH="/dev/loop19" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gtk-common-themes/1535"
NAME="loop20" PATH="/dev/loop20" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gtk-common-themes/1534"
NAME="loop21" PATH="/dev/loop21" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/gnome-3-38-2004/119"
NAME="loop22" PATH="/dev/loop22" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core18/2679"
NAME="loop23" PATH="/dev/loop23" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/snapd/18596"
NAME="loop24" PATH="/dev/loop24" PHY-SEC="512" SERIAL="" PARTTYPE="" PARTLABEL="" FSTYPE="squashfs" MOUNTPOINT="/snap/core20/1852"
NAME="sda" PATH="/dev/sda" PHY-SEC="512" SERIAL="4fc1a7b2-c90f-4fe3-88fc-e96de4c6348a" PARTTYPE="" PARTLABEL="" FSTYPE="" MOUNTPOINT=""
NAME="sda1" PATH="/dev/sda1" PHY-SEC="512" SERIAL="" PARTTYPE="c12a7328-f81f-11d2-ba4b-00a0c93ec93b" PARTLABEL="BOOT" FSTYPE="" MOUNTPOINT=""
NAME="sda2" PATH="/dev/sda2" PHY-SEC="512" SERIAL="" PARTTYPE="0fc63daf-8483-4772-8e79-3d69d8477de4" PARTLABEL="ROOT" FSTYPE="ext4" MOUNTPOINT="/"
NAME="sdb" PATH="/dev/sdb" PHY-SEC="512" SERIAL="000000001536" PARTTYPE="" PARTLABEL="" FSTYPE="" MOUNTPOINT=""
NAME="sdb1" PATH="/dev/sdb1" PHY-SEC="512" SERIAL="" PARTTYPE="0x6" PARTLABEL="" FSTYPE="vfat" MOUNTPOINT="/media/msshakeel/disk"
```

After running fsdisk command:

```
msshakeel@slbncsf2112pc46:~$ fdisk -l ~p15jra/bluedot.img
Disk /users/labnet5/fs1/p15jra/bluedot.img: 14.45 GiB, 15502147584 bytes, 30277632 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x0005a3a8

Device                                  Boot   Start      End  Sectors   Size Id Type
/users/labnet5/fs1/p15jra/bluedot.img1        2121728  3799447  1677720 819.2M  b W95 FAT32
/users/labnet5/fs1/p15jra/bluedot.img2        3801088 30277631 26476544  12.6G 83 Linux
/users/labnet5/fs1/p15jra/bluedot.img3           2048     4095     2048     1M a2 unknown

Partition table entries are not in disk order.
```

There are 3 partitions, bluedot.img1, bluedot.img2, bluedot.img3

Bluedot.img1 is of size 819.2M

Bluedot.img2 is of size 12.6G

Bluedot.img3 is of size 1M

Img3 is the preloader because the Id is a2 from our prelab.

extracting :

```
msshakeel@slbncsf2112pc46:~$ man dd
msshakeel@slbncsf2112pc46:~$ dd if=~p15jra/bluedot.img of=preloader.img bs=512 skip=2048 count=120
120+0 records in
120+0 records out
61440 bytes (61 kB, 60 KiB) copied, 0.0155691 s, 3.9 MB/s
```

hexdump() the first 80 bytes of the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 80
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050
```

The first four bytes at offset 0x40 is 41h, 53h, 30h, 31h.

Lab result:

```
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
```

The result is as expected. As we see the address from the prelab and the lab result is the same which is 0x31305341.

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 200
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050  40 00 00 01 e4 94 00 00  e4 94 00 00 88 95 00 00  |@...............|
00000060  e4 94 00 00 de c0 ad 0b  de c0 ad 0b de c0 ad 0b  |................|
00000070  42 00 00 eb 00 00 0f e1  1f 00 c0 e3 d3 00 80 e3  |B...............|
00000080  00 f0 29 e1 10 0f 11 ee  02 0a c0 e3 10 0f 01 ee  |..).............|
00000090  c8 00 9f e5 10 0f 0c ee  08 00 00 eb 18 00 00 eb  |................|
000000a0  41 06 00 eb 15 0f 07 ee  9a 0f 07 ee 95 0f 07 ee  |A...............|
000000b0  a8 00 9f e5 10 0f 0c ee  1e ff 2f e1 1e ff 2f e1  |........../.../.|
000000c0  00 00 a0 e3 17 0f 08 ee                           |........|
```

0xFFFF004C is entry point

Disassembly

Attempting to disassemble the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

Target is binary as img files are binary:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

6.4 Extract preloader image **2 / 2**

✓ **- 0 pts** *Correct*

hexdump() the first 80 bytes of the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 80
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050
```

The first four bytes at offset 0x40 is 41h, 53h, 30h, 31h.

Lab result:

```
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
```

The result is as expected. As we see the address from the prelab and the lab result is the same which is 0x31305341.

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 200
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050  40 00 00 01 e4 94 00 00  e4 94 00 00 88 95 00 00  |@...............|
00000060  e4 94 00 00 de c0 ad 0b  de c0 ad 0b de c0 ad 0b  |................|
00000070  42 00 00 eb 00 00 0f e1  1f 00 c0 e3 d3 00 80 e3  |B...............|
00000080  00 f0 29 e1 10 0f 11 ee  02 0a c0 e3 10 0f 01 ee  |..).............|
00000090  c8 00 9f e5 10 0f 0c ee  08 00 00 eb 18 00 00 eb  |................|
000000a0  41 06 00 eb 15 0f 07 ee  9a 0f 07 ee 95 0f 07 ee  |A...............|
000000b0  a8 00 9f e5 10 0f 0c ee  1e ff 2f e1 1e ff 2f e1  |........../.../.|
000000c0  00 00 a0 e3 17 0f 08 ee                           |........|
```

0xFFFF004C is entry point

Disassembly

Attempting to disassemble the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

Target is binary as img files are binary:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

7.1 hexdump **1 / 1**

✓ **- 0 pts** *Correct*

hexdump() the first 80 bytes of the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 80
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050
```

The first four bytes at offset 0x40 is 41h, 53h, 30h, 31h.

Lab result:

```
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
```

The result is as expected. As we see the address from the prelab and the lab result is the same which is 0x31305341.

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 200
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050  40 00 00 01 e4 94 00 00  e4 94 00 00 88 95 00 00  |@...............|
00000060  e4 94 00 00 de c0 ad 0b  de c0 ad 0b de c0 ad 0b  |................|
00000070  42 00 00 eb 00 00 0f e1  1f 00 c0 e3 d3 00 80 e3  |B...............|
00000080  00 f0 29 e1 10 0f 11 ee  02 0a c0 e3 10 0f 01 ee  |..).............|
00000090  c8 00 9f e5 10 0f 0c ee  08 00 00 eb 18 00 00 eb  |................|
000000a0  41 06 00 eb 15 0f 07 ee  9a 0f 07 ee 95 0f 07 ee  |A...............|
000000b0  a8 00 9f e5 10 0f 0c ee  1e ff 2f e1 1e ff 2f e1  |........../.../.|
000000c0  00 00 a0 e3 17 0f 08 ee                           |........|
```

0xFFFF004C is entry point

Disassembly

Attempting to disassemble the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

Target is binary as img files are binary:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

7.2 Bytes at offset 0x40 **0 / 1**

- 0 pts Correct

✓ **- 1 pts** *The value is different because of little-endian format*

hexdump() the first 80 bytes of the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 80
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050
```

The first four bytes at offset 0x40 is 41h, 53h, 30h, 31h.

Lab result:

```
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
```

The result is as expected. As we see the address from the prelab and the lab result is the same which is 0x31305341.

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 200
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050  40 00 00 01 e4 94 00 00  e4 94 00 00 88 95 00 00  |@...............|
00000060  e4 94 00 00 de c0 ad 0b  de c0 ad 0b de c0 ad 0b  |................|
00000070  42 00 00 eb 00 00 0f e1  1f 00 c0 e3 d3 00 80 e3  |B...............|
00000080  00 f0 29 e1 10 0f 11 ee  02 0a c0 e3 10 0f 01 ee  |..).............|
00000090  c8 00 9f e5 10 0f 0c ee  08 00 00 eb 18 00 00 eb  |................|
000000a0  41 06 00 eb 15 0f 07 ee  9a 0f 07 ee 95 0f 07 ee  |A...............|
000000b0  a8 00 9f e5 10 0f 0c ee  1e ff 2f e1 1e ff 2f e1  |........../.../.|
000000c0  00 00 a0 e3 17 0f 08 ee                           |........|
```

0xFFFF004C is entry point

Disassembly

Attempting to disassemble the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

Target is binary as img files are binary:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

7.3 Entry point **1 / 2**

- 0 pts Correct

✓ **- 1 pts** *The value is invalid because it's referring to an out of bound address*

hexdump() the first 80 bytes of the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 80
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050
```

The first four bytes at offset 0x40 is 41h, 53h, 30h, 31h.

Lab result:

```
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
```

The result is as expected. As we see the address from the prelab and the lab result is the same which is 0x31305341.

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 200
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050  40 00 00 01 e4 94 00 00  e4 94 00 00 88 95 00 00  |@...............|
00000060  e4 94 00 00 de c0 ad 0b  de c0 ad 0b de c0 ad 0b  |................|
00000070  42 00 00 eb 00 00 0f e1  1f 00 c0 e3 d3 00 80 e3  |B...............|
00000080  00 f0 29 e1 10 0f 11 ee  02 0a c0 e3 10 0f 01 ee  |..).............|
00000090  c8 00 9f e5 10 0f 0c ee  08 00 00 eb 18 00 00 eb  |................|
000000a0  41 06 00 eb 15 0f 07 ee  9a 0f 07 ee 95 0f 07 ee  |A...............|
000000b0  a8 00 9f e5 10 0f 0c ee  1e ff 2f e1 1e ff 2f e1  |........../.../.|
000000c0  00 00 a0 e3 17 0f 08 ee                           |........|
```

0xFFFF004C is entry point

Disassembly

Attempting to disassemble the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

Target is binary as img files are binary:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

Changing path for objdump:

```
msshakeel@slbncsf2112pc46:~$ export PATH=/usr/lib/arm-none-eabi/bin:$PATH
msshakeel@slbncsf2112pc46:~$ which objdump
/usr/lib/arm-none-eabi/bin/objdump
```

Finding the objdump info:

```
msshakeel@slbncsf2112pc46:~$ objdump --info
BFD header file version (2.34-4ubuntu1+13ubuntu1) 2.34
elf32-littlearm
 (header little endian, data little endian)
  arm
elf32-littlearm-fdpic
 (header little endian, data little endian)
  arm
elf32-bigarm
 (header big endian, data big endian)
  arm
elf32-bigarm-fdpic
 (header big endian, data big endian)
  arm
elf32-little
 (header little endian, data little endian)
  arm
elf32-big
 (header big endian, data big endian)
  arm
srec
 (header endianness unknown, data endianness unknown)
  arm
symbolsrec
 (header endianness unknown, data endianness unknown)
  arm
verilog
 (header endianness unknown, data endianness unknown)
  arm
tekhex
 (header endianness unknown, data endianness unknown)
  arm
binary
 (header endianness unknown, data endianness unknown)
  arm
ihex
 (header endianness unknown, data endianness unknown)
  arm
plugin
 (header little endian, data little endian)

         elf32-littlearm elf32-littlearm-fdpic elf32-bigarm elf32-bigarm-fdpic
     arm elf32-littlearm elf32-littlearm-fdpic elf32-bigarm elf32-bigarm-fdpic

         elf32-little elf32-big srec symbolsrec verilog tekhex binary ihex
     arm elf32-little elf32-big srec symbolsrec verilog tekhex binary ihex

         plugin
     arm ------
msshakeel@slbncsf2112pc46:~$ export/users/labnet/st3/msshakeel/.linuxbrew/bin/objdump
```

8.1 objdump failure **1 / 1**

✓ **- 0 pts** *Correct*

Changing path for objdump:

```
msshakeel@slbncsf2112pc46:~$ export PATH=/usr/lib/arm-none-eabi/bin:$PATH
msshakeel@slbncsf2112pc46:~$ which objdump
/usr/lib/arm-none-eabi/bin/objdump
```

Finding the objdump info:

```
msshakeel@slbncsf2112pc46:~$ objdump --info
BFD header file version (2.34-4ubuntu1+13ubuntu1) 2.34
elf32-littlearm
 (header little endian, data little endian)
  arm
elf32-littlearm-fdpic
 (header little endian, data little endian)
  arm
elf32-bigarm
 (header big endian, data big endian)
  arm
elf32-bigarm-fdpic
 (header big endian, data big endian)
  arm
elf32-little
 (header little endian, data little endian)
  arm
elf32-big
 (header big endian, data big endian)
  arm
srec
 (header endianness unknown, data endianness unknown)
  arm
symbolsrec
 (header endianness unknown, data endianness unknown)
  arm
verilog
 (header endianness unknown, data endianness unknown)
  arm
tekhex
 (header endianness unknown, data endianness unknown)
  arm
binary
 (header endianness unknown, data endianness unknown)
  arm
ihex
 (header endianness unknown, data endianness unknown)
  arm
plugin
 (header little endian, data little endian)

         elf32-littlearm elf32-littlearm-fdpic elf32-bigarm elf32-bigarm-fdpic
     arm elf32-littlearm elf32-littlearm-fdpic elf32-bigarm elf32-bigarm-fdpic

         elf32-little elf32-big srec symbolsrec verilog tekhex binary ihex
     arm elf32-little elf32-big srec symbolsrec verilog tekhex binary ihex

         plugin
     arm ------
msshakeel@slbncsf2112pc46:~$ export/users/labnet/st3/msshakeel/.linuxbrew/bin/objdump
```

It can be observed that the architecture is ARM.

As such:

```
msshakeel@slbncsf2112pc46:~$ objdump -D --target=binary --architecture=arm preloader.img

preloader.img:     file format binary


Disassembly of section .data:

00000000 <.data>:
       0:       ea00001a        b       0x70
       4:       e59ff014        ldr     pc, [pc, #20]   ; 0x20
       8:       e59ff014        ldr     pc, [pc, #20]   ; 0x24
       c:       e59ff014        ldr     pc, [pc, #20]   ; 0x28
      10:       e59ff014        ldr     pc, [pc, #20]   ; 0x2c
      14:       e59ff014        ldr     pc, [pc, #20]   ; 0x30
      18:       e59ff014        ldr     pc, [pc, #20]   ; 0x34
      1c:       e59ff014        ldr     pc, [pc, #20]   ; 0x38
      20:       ffff0020                        ; <UNDEFINED> instruction: 0xffff0020
      24:       ffff0024                        ; <UNDEFINED> instruction: 0xffff0024
      28:       ffff0028                        ; <UNDEFINED> instruction: 0xffff0028
      2c:       ffff002c                        ; <UNDEFINED> instruction: 0xffff002c
      30:       ffff0030                        ; <UNDEFINED> instruction: 0xffff0030
      34:       ffff0120                        ; <UNDEFINED> instruction: 0xffff0120
      38:       ffff0038                        ; <UNDEFINED> instruction: 0xffff0038
      3c:       12345678        eorsne  r5, r4, #120, 12        ; 0x7800000
      40:       31305341        teqcc   r0, r1, asr #6
      44:       25400000        strbcs  r0, [r0, #-0]
      48:       015a0000        cmpeq   sl, r0
      4c:       ea000007        b       0x70
```

What instructions do you find in the exception vector table?

```
00000000 <.data>:
       0:       ea00001a        b       0x70
       4:       e59ff014        ldr     pc, [pc, #20]   ; 0x20
       8:       e59ff014        ldr     pc, [pc, #20]   ; 0x24
       c:       e59ff014        ldr     pc, [pc, #20]   ; 0x28
      10:       e59ff014        ldr     pc, [pc, #20]   ; 0x2c
      14:       e59ff014        ldr     pc, [pc, #20]   ; 0x30
      18:       e59ff014        ldr     pc, [pc, #20]   ; 0x34
      1c:       e59ff014        ldr     pc, [pc, #20]   ; 0x38
      20:       ffff0020                        ; <UNDEFINED> instruction: 0xffff0020
      24:       ffff0024                        ; <UNDEFINED> instruction: 0xffff0024
      28:       ffff0028                        ; <UNDEFINED> instruction: 0xffff0028
      2c:       ffff002c                        ; <UNDEFINED> instruction: 0xffff002c
      30:       ffff0030                        ; <UNDEFINED> instruction: 0xffff0030
      34:       ffff0120                        ; <UNDEFINED> instruction: 0xffff0120
      38:       ffff0038                        ; <UNDEFINED> instruction: 0xffff0038
      3c:       12345678        eorsne  r5, r4, #120, 12        ; 0x7800000
      40:       31305341        teqcc   r0, r1, asr #6
```

8.2 objdump --target **2 / 2**

✓ **- 0 pts** *Correct*

- 1 pts Missing evidence of the work

hexdump() the first 80 bytes of the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 80
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050
```

The first four bytes at offset 0x40 is 41h, 53h, 30h, 31h.

Lab result:

```
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
```

The result is as expected. As we see the address from the prelab and the lab result is the same which is 0x31305341.

```
msshakeel@slbncsf2112pc46:~$ hexdump -C preloader.img -n 200
00000000  1a 00 00 ea 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000010  14 f0 9f e5 14 f0 9f e5  14 f0 9f e5 14 f0 9f e5  |................|
00000020  20 00 ff ff 24 00 ff ff  28 00 ff ff 2c 00 ff ff  | ...$...(...,...|
00000030  30 00 ff ff 20 01 ff ff  38 00 ff ff 78 56 34 12  |0... ...8...xV4.|
00000040  41 53 30 31 00 00 40 25  00 00 5a 01 07 00 00 ea  |AS01..@%..Z.....|
00000050  40 00 00 01 e4 94 00 00  e4 94 00 00 88 95 00 00  |@...............|
00000060  e4 94 00 00 de c0 ad 0b  de c0 ad 0b de c0 ad 0b  |................|
00000070  42 00 00 eb 00 00 0f e1  1f 00 c0 e3 d3 00 80 e3  |B...............|
00000080  00 f0 29 e1 10 0f 11 ee  02 0a c0 e3 10 0f 01 ee  |..).............|
00000090  c8 00 9f e5 10 0f 0c ee  08 00 00 eb 18 00 00 eb  |................|
000000a0  41 06 00 eb 15 0f 07 ee  9a 0f 07 ee 95 0f 07 ee  |A...............|
000000b0  a8 00 9f e5 10 0f 0c ee  1e ff 2f e1 1e ff 2f e1  |........../.../.|
000000c0  00 00 a0 e3 17 0f 08 ee                           |........|
```

0xFFFF004C is entry point

Disassembly

Attempting to disassemble the preloader.img:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

Target is binary as img files are binary:

```
msshakeel@slbncsf2112pc46:~$ objdump -d preloader.img
objdump: preloader.img: file format not recognized
msshakeel@slbncsf2112pc46:~$
```

8.3 objdump failure (again) **1 / 1**

✓ **- 0 pts** *Correct*

- 1 pts Not answered/completed

It can be observed that the architecture is ARM.

As such:

```
msshakeel@slbncsf2112pc46:~$ objdump -D --target=binary --architecture=arm preloader.img

preloader.img:     file format binary


Disassembly of section .data:

00000000 <.data>:
       0:       ea00001a        b       0x70
       4:       e59ff014        ldr     pc, [pc, #20]   ; 0x20
       8:       e59ff014        ldr     pc, [pc, #20]   ; 0x24
       c:       e59ff014        ldr     pc, [pc, #20]   ; 0x28
      10:       e59ff014        ldr     pc, [pc, #20]   ; 0x2c
      14:       e59ff014        ldr     pc, [pc, #20]   ; 0x30
      18:       e59ff014        ldr     pc, [pc, #20]   ; 0x34
      1c:       e59ff014        ldr     pc, [pc, #20]   ; 0x38
      20:       ffff0020                        ; <UNDEFINED> instruction: 0xffff0020
      24:       ffff0024                        ; <UNDEFINED> instruction: 0xffff0024
      28:       ffff0028                        ; <UNDEFINED> instruction: 0xffff0028
      2c:       ffff002c                        ; <UNDEFINED> instruction: 0xffff002c
      30:       ffff0030                        ; <UNDEFINED> instruction: 0xffff0030
      34:       ffff0120                        ; <UNDEFINED> instruction: 0xffff0120
      38:       ffff0038                        ; <UNDEFINED> instruction: 0xffff0038
      3c:       12345678        eorsne  r5, r4, #120, 12        ; 0x7800000
      40:       31305341        teqcc   r0, r1, asr #6
      44:       25400000        strbcs  r0, [r0, #-0]
      48:       015a0000        cmpeq   sl, r0
      4c:       ea000007        b       0x70
```

What instructions do you find in the exception vector table?

```
00000000 <.data>:
       0:       ea00001a        b       0x70
       4:       e59ff014        ldr     pc, [pc, #20]   ; 0x20
       8:       e59ff014        ldr     pc, [pc, #20]   ; 0x24
       c:       e59ff014        ldr     pc, [pc, #20]   ; 0x28
      10:       e59ff014        ldr     pc, [pc, #20]   ; 0x2c
      14:       e59ff014        ldr     pc, [pc, #20]   ; 0x30
      18:       e59ff014        ldr     pc, [pc, #20]   ; 0x34
      1c:       e59ff014        ldr     pc, [pc, #20]   ; 0x38
      20:       ffff0020                        ; <UNDEFINED> instruction: 0xffff0020
      24:       ffff0024                        ; <UNDEFINED> instruction: 0xffff0024
      28:       ffff0028                        ; <UNDEFINED> instruction: 0xffff0028
      2c:       ffff002c                        ; <UNDEFINED> instruction: 0xffff002c
      30:       ffff0030                        ; <UNDEFINED> instruction: 0xffff0030
      34:       ffff0120                        ; <UNDEFINED> instruction: 0xffff0120
      38:       ffff0038                        ; <UNDEFINED> instruction: 0xffff0038
      3c:       12345678        eorsne  r5, r4, #120, 12        ; 0x7800000
      40:       31305341        teqcc   r0, r1, asr #6
```

8.4 Disassemble the preloader **3 / 3**

✓ **- 0 pts** *Correct*

- 1 pts The binary is not disassembled completely

- 3 pts Failed to finish the section

It can be observed that the architecture is ARM.

As such:

```
msshakeel@slbncsf2112pc46:~$ objdump -D --target=binary --architecture=arm preloader.img

preloader.img:     file format binary


Disassembly of section .data:

00000000 <.data>:
       0:       ea00001a        b       0x70
       4:       e59ff014        ldr     pc, [pc, #20]   ; 0x20
       8:       e59ff014        ldr     pc, [pc, #20]   ; 0x24
       c:       e59ff014        ldr     pc, [pc, #20]   ; 0x28
      10:       e59ff014        ldr     pc, [pc, #20]   ; 0x2c
      14:       e59ff014        ldr     pc, [pc, #20]   ; 0x30
      18:       e59ff014        ldr     pc, [pc, #20]   ; 0x34
      1c:       e59ff014        ldr     pc, [pc, #20]   ; 0x38
      20:       ffff0020                        ; <UNDEFINED> instruction: 0xffff0020
      24:       ffff0024                        ; <UNDEFINED> instruction: 0xffff0024
      28:       ffff0028                        ; <UNDEFINED> instruction: 0xffff0028
      2c:       ffff002c                        ; <UNDEFINED> instruction: 0xffff002c
      30:       ffff0030                        ; <UNDEFINED> instruction: 0xffff0030
      34:       ffff0120                        ; <UNDEFINED> instruction: 0xffff0120
      38:       ffff0038                        ; <UNDEFINED> instruction: 0xffff0038
      3c:       12345678        eorsne  r5, r4, #120, 12        ; 0x7800000
      40:       31305341        teqcc   r0, r1, asr #6
      44:       25400000        strbcs  r0, [r0, #-0]
      48:       015a0000        cmpeq   sl, r0
      4c:       ea000007        b       0x70
```

What instructions do you find in the exception vector table?

```
00000000 <.data>:
       0:       ea00001a        b       0x70
       4:       e59ff014        ldr     pc, [pc, #20]   ; 0x20
       8:       e59ff014        ldr     pc, [pc, #20]   ; 0x24
       c:       e59ff014        ldr     pc, [pc, #20]   ; 0x28
      10:       e59ff014        ldr     pc, [pc, #20]   ; 0x2c
      14:       e59ff014        ldr     pc, [pc, #20]   ; 0x30
      18:       e59ff014        ldr     pc, [pc, #20]   ; 0x34
      1c:       e59ff014        ldr     pc, [pc, #20]   ; 0x38
      20:       ffff0020                        ; <UNDEFINED> instruction: 0xffff0020
      24:       ffff0024                        ; <UNDEFINED> instruction: 0xffff0024
      28:       ffff0028                        ; <UNDEFINED> instruction: 0xffff0028
      2c:       ffff002c                        ; <UNDEFINED> instruction: 0xffff002c
      30:       ffff0030                        ; <UNDEFINED> instruction: 0xffff0030
      34:       ffff0120                        ; <UNDEFINED> instruction: 0xffff0120
      38:       ffff0038                        ; <UNDEFINED> instruction: 0xffff0038
      3c:       12345678        eorsne  r5, r4, #120, 12        ; 0x7800000
      40:       31305341        teqcc   r0, r1, asr #6
```

8.5 Instructions in exception vector table **2 / 2**

✓ **- 0 pts** *Correct*

- 2 pts Not answered