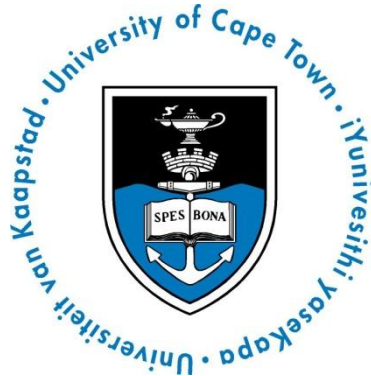


# QEMU ARM Emulator and Kernel Building

---

## Practical 4



Prepared by:  
**Ivan Tchekashkine**  
TCHIVA001

**23 May 2012**

# 1. QEMU ARM Emulator

---

## 1.1 Introduction

This part of the practical was concerned with using QEMU to emulate an ARM processor. More specifically a versatile platform baseboard was emulated and a Linux kernel was loaded on to it in order to facilitate a simple Linux test program. Section 1.2 will explain the procedures that were taken to achieve this. Following that, Section 1.3 will provide the results obtained in this part of the practical.

## 1.2 Method

Before the test application was created, a bootloader shell had to be loaded onto the emulated processor and then a Linux Kernel together with the test program could be loaded. The test program was written and saved in the **test.c** file; the steps which were done to compile and load the program are shown in Figure 1.1.

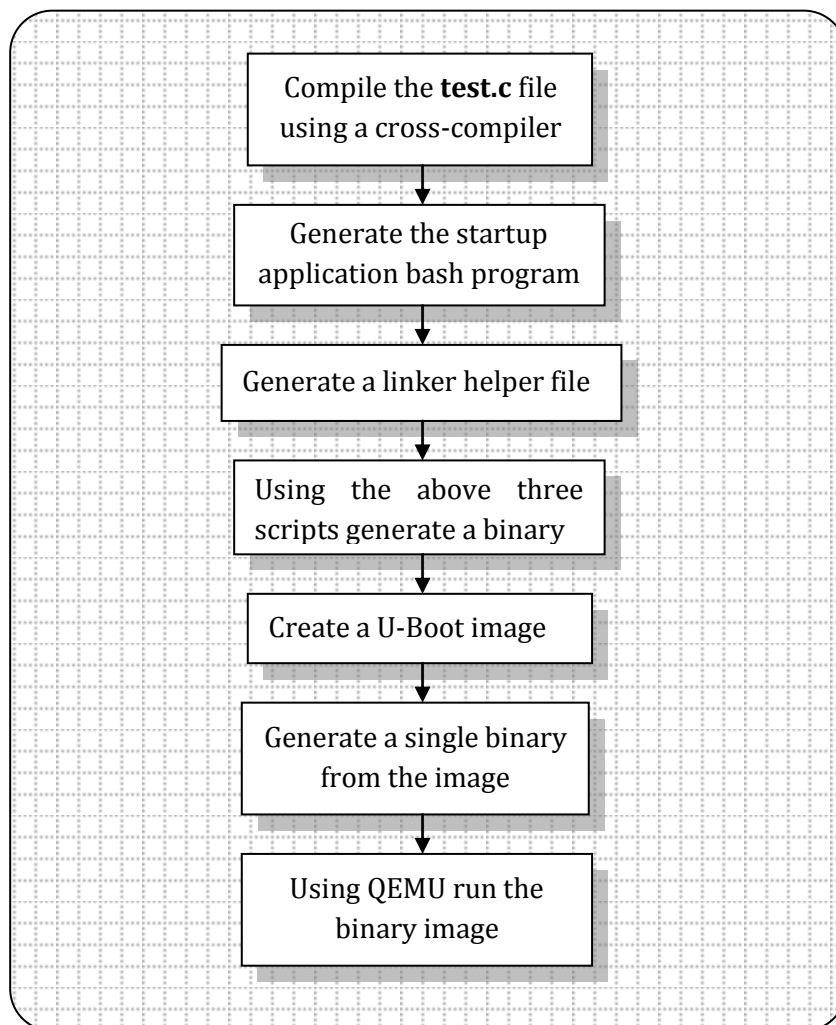
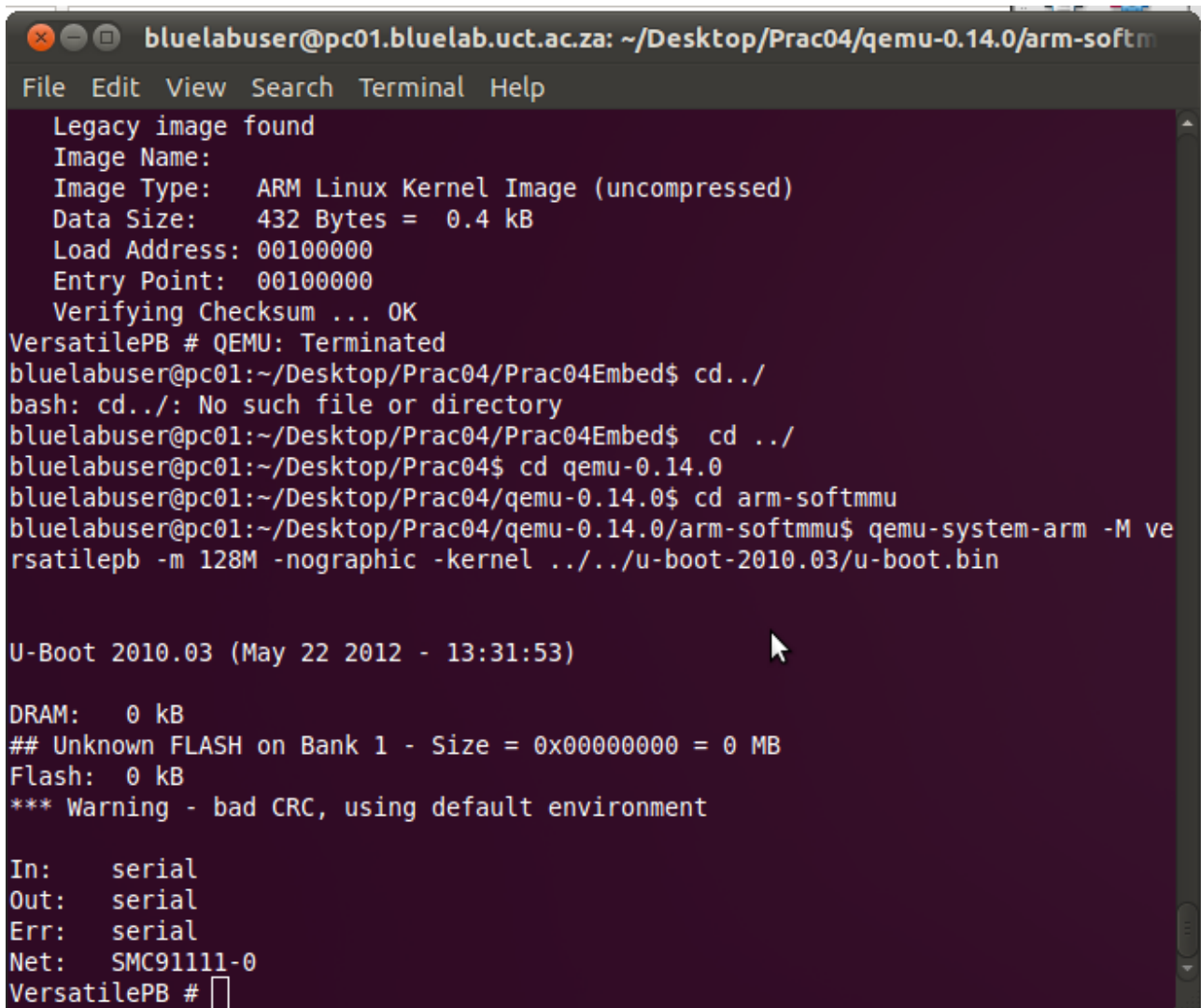


Figure 1.1 Basic steps taken to emulate a test program

The startup application bash program was implemented in **startup.s** file and had the name of the function *testfunc* replaced after the initial run to *LetterFunc*. That was done to refrain from modifying the original test function in **test.c** file, so that the original output could be easily replicated at will.

### 1.3 Results

First the ARM processor along with the versatilepb board were emulated and the bootloader was loaded (u-boot image). The output of the operation can be seen in Figure 1.2.



```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/qemu-0.14.0/arm-softm
File Edit View Search Terminal Help
Legacy image found
Image Name:
Image Type:  ARM Linux Kernel Image (uncompressed)
Data Size:   432 Bytes =  0.4 kB
Load Address: 00100000
Entry Point: 00100000
Verifying Checksum ... OK
VersatilePB # QEMU: Terminated
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ cd../
bash: cd../: No such file or directory
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ cd ../
bluelabuser@pc01:~/Desktop/Prac04$ cd qemu-0.14.0
bluelabuser@pc01:~/Desktop/Prac04/qemu-0.14.0$ cd arm-softmmu
bluelabuser@pc01:~/Desktop/Prac04/qemu-0.14.0/arm-softmmu$ qemu-system-arm -M ve
rsatilepb -m 128M -nographic -kernel ../../u-boot-2010.03/u-boot.bin

U-Boot 2010.03 (May 22 2012 - 13:31:53)

DRAM:  0 kB
## Unknown FLASH on Bank 1 - Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   SMC91111-0
VersatilePB #
```

Figure 1.2 The simulated output of loading the Linux Kernel on versatilepb platform

Once the shell was loaded, the command prompt was used to carry out user commands. The list of possible commands was accessed using the “help” and “?” commands (no quotation marks when used in the command prompt). The use of the two commands is demonstrated in Figures 1.3 and 1.4.

```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/qemu-0.14.0/arm-softm
File Edit View Search Terminal Help
Net: SMC91111-0
VersatilePB # help
? - alias for 'help'
base - print or set address offset
bdinfo - print Board Info structure
bootm - boot application image from memory
bootp - boot image via network using BOOTP/TFTP protocol
cmp - memory compare
cp - memory copy
crc32 - checksum calculation
dhcp - boot image via network using DHCP/TFTP protocol
erase - erase FLASH memory
flinfo - print FLASH memory information
go - start application at address 'addr'
help - print command description/usage
iminfo - print header information for application image
loop - infinite loop on address range
md - memory display
mm - memory modify (auto-incrementing address)
mtest - simple RAM read/write test
mw - memory write (fill)
nm - memory modify (constant address)
ping - send ICMP ECHO_REQUEST to network host
printenv - print environment variables
protect - enable or disable FLASH write protection
rarpboot - boot image via network using RARP/TFTP protocol
reset - Perform RESET of the CPU
saveenv - save environment variables to persistent storage
setenv - set environment variables
tftpboot - boot image via network using TFTP protocol
version - print monitor version
VersatilePB #
```

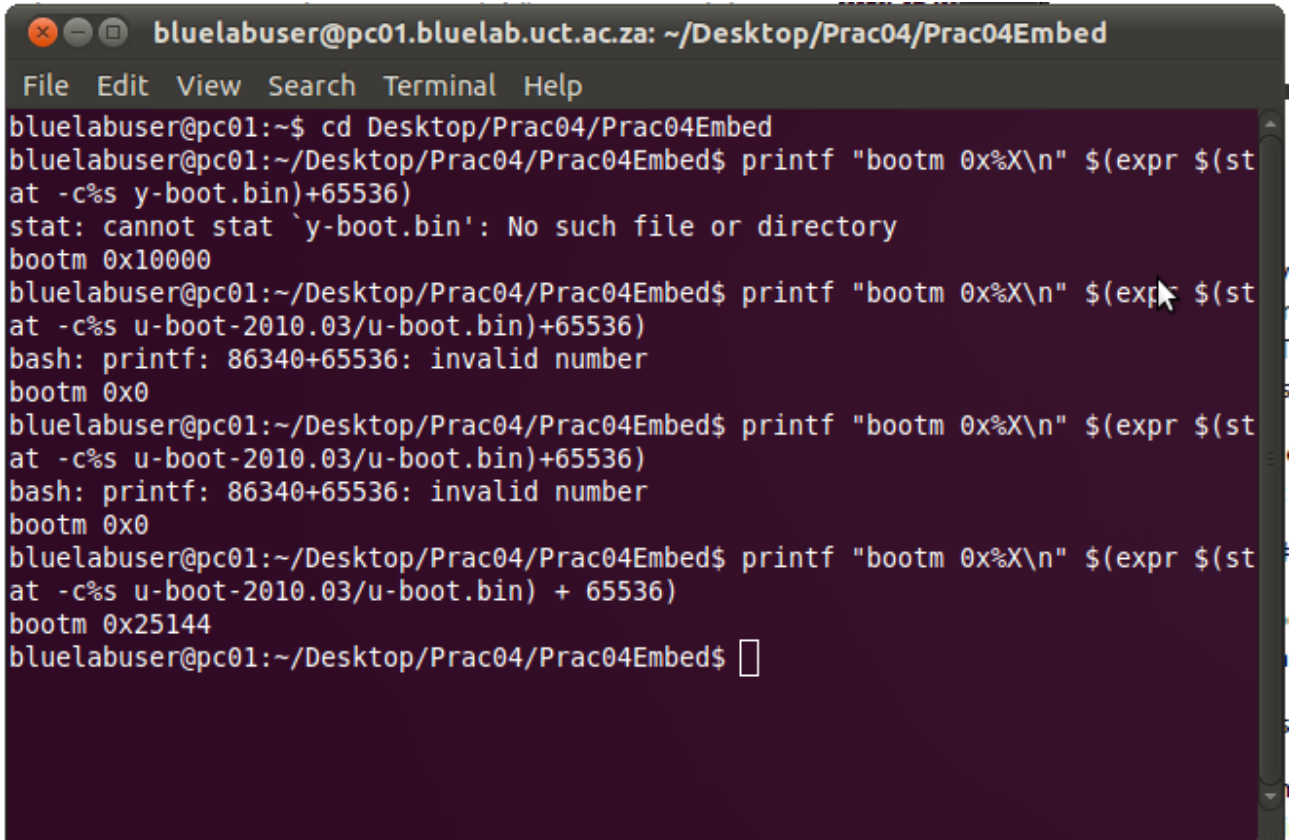
Figure 1.3 Use of “help” command

```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/qemu-0.14.0/arm-softm
File Edit View Search Terminal Help
tftpboot- boot image via network using TFTP protocol
version - print monitor version
VersatilePB # ?
? - alias for 'help'
base - print or set address offset
bdinfo - print Board Info structure
bootm - boot application image from memory
bootp - boot image via network using BOOTP/TFTP protocol
cmp - memory compare
cp - memory copy
crc32 - checksum calculation
dhcp - boot image via network using DHCP/TFTP protocol
erase - erase FLASH memory
flinfo - print FLASH memory information
go - start application at address 'addr'
help - print command description/usage
iminfo - print header information for application image
loop - infinite loop on address range
md - memory display
mm - memory modify (auto-incrementing address)
mtest - simple RAM read/write test
mw - memory write (fill)
nm - memory modify (constant address)
ping - send ICMP ECHO_REQUEST to network host
printenv - print environment variables
protect - enable or disable FLASH write protection
rarpboot - boot image via network using RARP/TFTP protocol
reset - Perform RESET of the CPU
saveenv - save environment variables to persistent storage
setenv - set environment variables
tftpboot - boot image via network using TFTP protocol
version - print monitor version
VersatilePB #
```

Figure 1.4 Use of “?” command

From the list of commands that are displayed in Figures 1.3 and 1.4, it was determined that to display the header information of the application image the command “iminfo” could be used. Furthermore, the command “bootm” could be used to boot an application image from memory.

At this point the test application was compiled using the procedure outlined in Figure 1.1. Once the application image was made and the binary of the image created, the gnu-shell could be used to load the file to the processor, however, to be able to do that the start address of the image in the file was needed to be calculated. The command used as well as the calculated address can be seen in Figure 1.5.

A terminal window titled 'bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/Prac04Embed'. The window contains a series of commands and their outputs. The user navigates to the directory ~/Desktop/Prac04/Prac04Embed. They attempt to use the 'bootm' command with a file path, but receive an error: 'stat: cannot stat `y-boot.bin': No such file or directory'. They then try to calculate the start address using 'expr' and 'stat', but receive an error: 'bash: printf: 86340+65536: invalid number'. Finally, they successfully calculate the start address as 0x25144.

```
bluelabuser@pc01:~$ cd Desktop/Prac04/Prac04Embed
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ printf "bootm 0x%X\n" $(expr $(stat
at -c%s y-boot.bin)+65536)
stat: cannot stat `y-boot.bin': No such file or directory
bootm 0x10000
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ printf "bootm 0x%X\n" $(expr $(stat
at -c%s u-boot-2010.03/u-boot.bin)+65536)
bash: printf: 86340+65536: invalid number
bootm 0x0
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ printf "bootm 0x%X\n" $(expr $(stat
at -c%s u-boot-2010.03/u-boot.bin) + 65536)
bash: printf: 86340+65536: invalid number
bootm 0x0
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ printf "bootm 0x%X\n" $(expr $(stat
at -c%s u-boot-2010.03/u-boot.bin) + 65536)
bootm 0x25144
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$
```

Figure 1.5 Start address calculation

The outcome address was: 0x25144

To confirm that the above address indeed corresponds to the application image that was loaded onto the emulated processor the command “iminfo” can be used in the command prompt. The result can be seen in Figure 1.6.

```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/Prac04Embed
File Edit View Search Terminal Help
Starting kernel ...

Hello:Check this out.!QEMU: Terminated
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin

U-Boot 2010.03 (May 22 2012 - 13:31:53)

DRAM: 0 kB
## Unknown FLASH on Bank 1 - Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning - bad CRC, using default environment

In: serial
Out: serial
Err: serial
Net: SMC91111-0
VersatilePB # iminfo 0x25144

## Checking Image at 00025144 ...
Legacy image found
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 432 Bytes = 0.4 kB
Load Address: 00100000
Entry Point: 00100000
Verifying Checksum ... OK
VersatilePB #
```

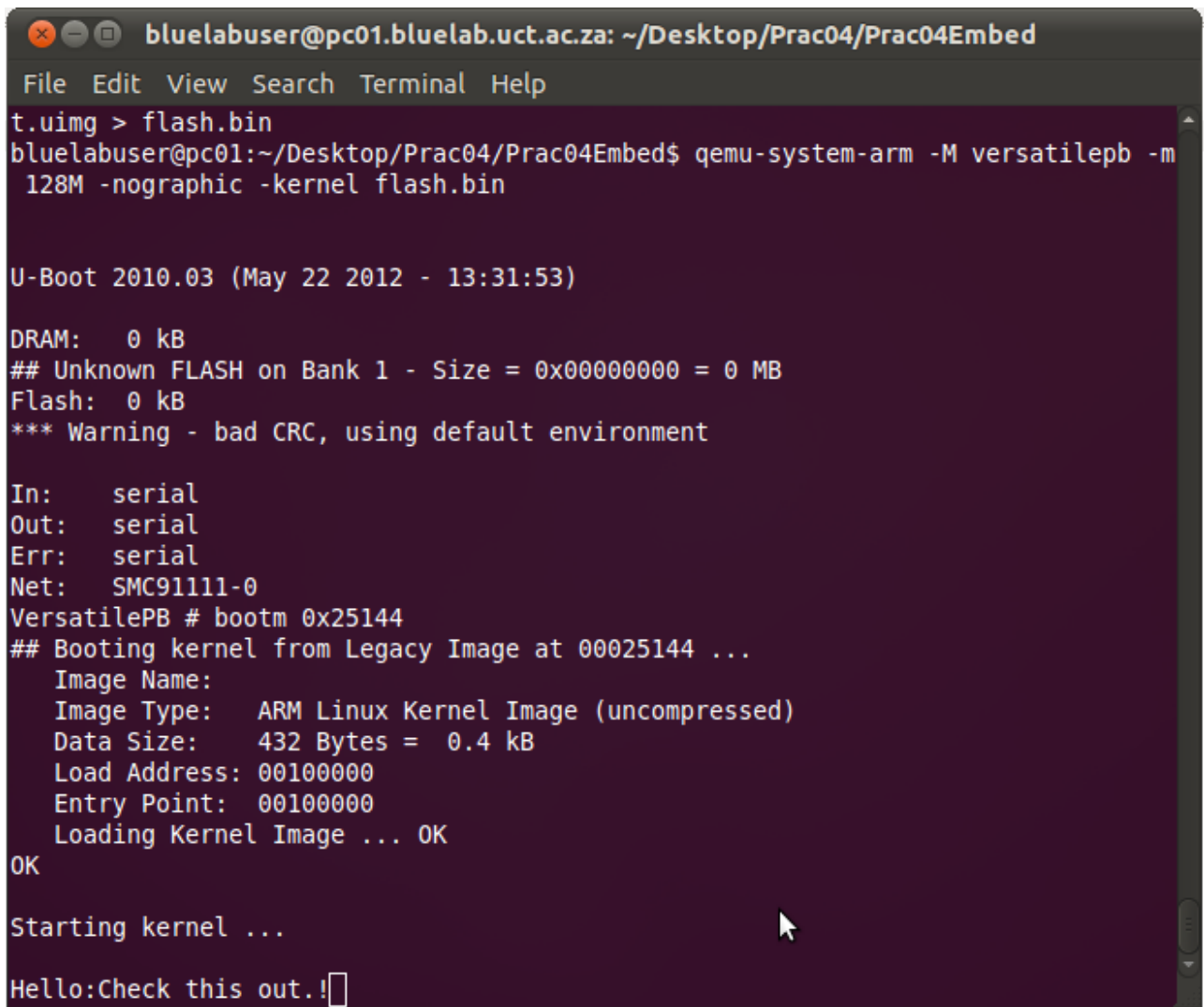
Figure 1.6 Address verification

Figure 1.6 shows that indeed the address 0x25144 corresponds with the start of the image as it can be seen that the image type is indeed the ARM Linux Kernel Image and has a point of entry at 0x00100000, as specified in the following command which was used to create the binary file:

```
mkimage -A arm -C none -O linux -T kernel -d test.bin -a 0x00100000 -e 0x00100000 test.uimg
cat u-boot-2010.03/u-boot.bin test.uimg > flash.bin
```

(Can be seen that address is determined by -a and set to 0x00100000 and -e shows the entry point, which also corresponds with that of Figure 1.6)

Having established that address 0x25144 corresponds to the image, the image was loaded onto the emulated processor using the command “bootm”. The result can be seen in Figure 1.7.



```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/Prac04Embed
File Edit View Search Terminal Help
t.uing > flash.bin
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ qemu-system-arm -M versatilepb -m
128M -nographic -kernel flash.bin

U-Boot 2010.03 (May 22 2012 - 13:31:53)

DRAM: 0 kB
## Unknown FLASH on Bank 1 - Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning - bad CRC, using default environment

In: serial
Out: serial
Err: serial
Net: SMC91111-0
VersatilePB # bootm 0x25144
## Booting kernel from Legacy Image at 00025144 ...
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 432 Bytes = 0.4 kB
Load Address: 00100000
Entry Point: 00100000
Loading Kernel Image ... OK
OK

Starting kernel ...

Hello:Check this out.!
```

Figure 1.7 The result of booting the image that contained the test program

Figure 1.7 shows the expected result as the message that function *testfunc* printed using the *print\_uart0* function is shown at the bottom. However, it must be noted that the screen shot was taken after all the coding was complete, hence the file **test.c** contained the function *LetterFunc* as well, therefore the resultant data size of the image is larger than the one expected if the program only contained *testfunc*.

The next part required a program to take a small letter and output its upper case form (in case if the letter was in upper case already, it would just be duplicated). The program made use of *LetterFunc* to change the letter from lower to upper case and functions *print\_uart0* and *print\_uart1* to print the result. The two cases can be seen in Figure 1.8 and 1.9.

The code for the first test function as well as the code for the program that prints the upper case of a given letter is attached with this report and can be found in the CODE folder in the file **test.c**. A possible improvement to the program that occurred to the author during the write up process is to include an additional *if statement* that checks that the input character is indeed a letter, for example:

```
If ((c>='a' && c<='z') || (c>='A' && c<='Z'))
{ //proceed with the letter case conversion... }
```

```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/Prac04Embed
File Edit View Search Terminal Help
t.uing > flash.bin
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin

U-Boot 2010.03 (May 22 2012 - 13:31:53)

DRAM:  0 kB
## Unknown FLASH on Bank 1 - Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning - bad CRC, using default environment

In:     serial
Out:    serial
Err:    serial
Net:    SMC91111-0
VersatilePB # bootm 0x25144
## Booting kernel from Legacy Image at 00025144 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    432 Bytes =  0.4 kB
   Load Address: 00100000
   Entry Point:  00100000
   Loading Kernel Image ... OK
OK

Starting kernel ...

Given Character: y Upper Case: Y
```

Figure 1.8 Shows the given letter “y” and its upper case: “Y” (output)

```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/Prac04Embed
File Edit View Search Terminal Help
t.uing > flash.bin
bluelabuser@pc01:~/Desktop/Prac04/Prac04Embed$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel flash.bin

U-Boot 2010.03 (May 22 2012 - 13:31:53)

DRAM:  0 kB
## Unknown FLASH on Bank 1 - Size = 0x00000000 = 0 MB
Flash: 0 kB
*** Warning - bad CRC, using default environment

In:     serial
Out:    serial
Err:    serial
Net:    SMC91111-0
VersatilePB # bootm 0x25144
## Booting kernel from Legacy Image at 00025144 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    384 Bytes =  0.4 kB
   Load Address: 00100000
   Entry Point:  00100000
   Loading Kernel Image ... OK
OK

Starting kernel ...

Given Character: S Upper Case: S
```

Figure 1.9 Shows the given letter: S, which is already upper case, hence the output is just a duplicate of the input

## 2. SQLite as an Embedded DBMS

### 2.1 Introduction

This part of the practical is concerned with the use of SQLite to create a database for an embedded system. Section 2.2 will present the database Entity Relational Diagram for the database that contains three tables, which describe the relationships between the different factors that affect the storage of high quality wood planks. Section 2.3 then proceeds to use the SQL language to create a database **sensordb** and find the particular factors linked to a certain time and rack number.

### 2.2 Database Entity Relational Diagram (ERD)

The database **sensordb** consists of three tables: the **RACK** table, **SensorAssignment** table and **SensorCondition** table. The ERD was constructed according to the rules outlines in [1] and [2]. The diagram consists of entities, attributes of entities and relationships between the entities. The cardinality (relationship between entities) between the entities is 1:M (one-to-many) because:

- The **RACK** entity has attribute *rackID*, which has two attributes assigned to it in **SensorAssignment** entity
- The **SensorCondition** entity has the attribute *sensorID* which has two attributes that correspond to it in the **SensorAssignment** entity
- The **SensorCondition** also has the *Time* attribute that has two attributes that are allocated to it in the **RACK** entity.

Therefore, the **SensorAssignment** entity serves as a junction entity. The resultant ERD can be seen in Figure 2.1.

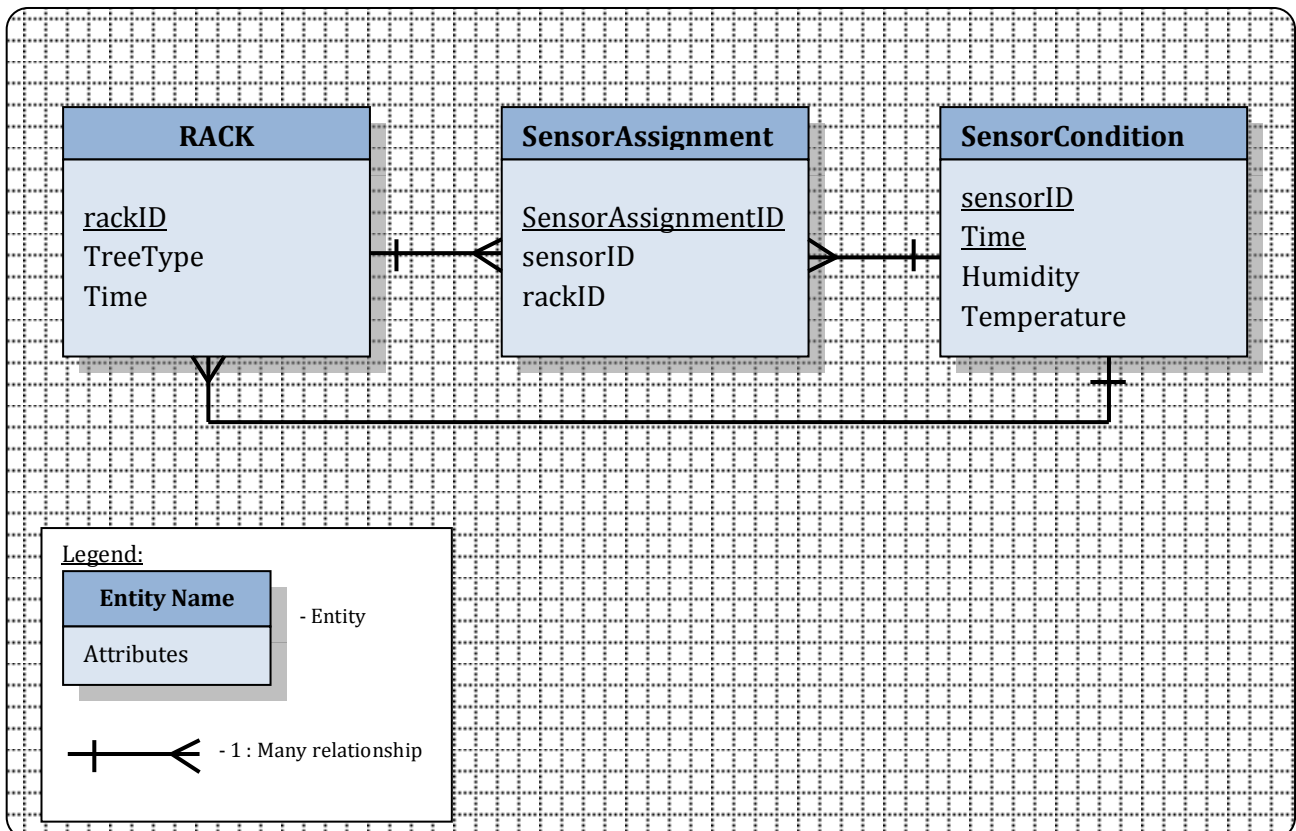
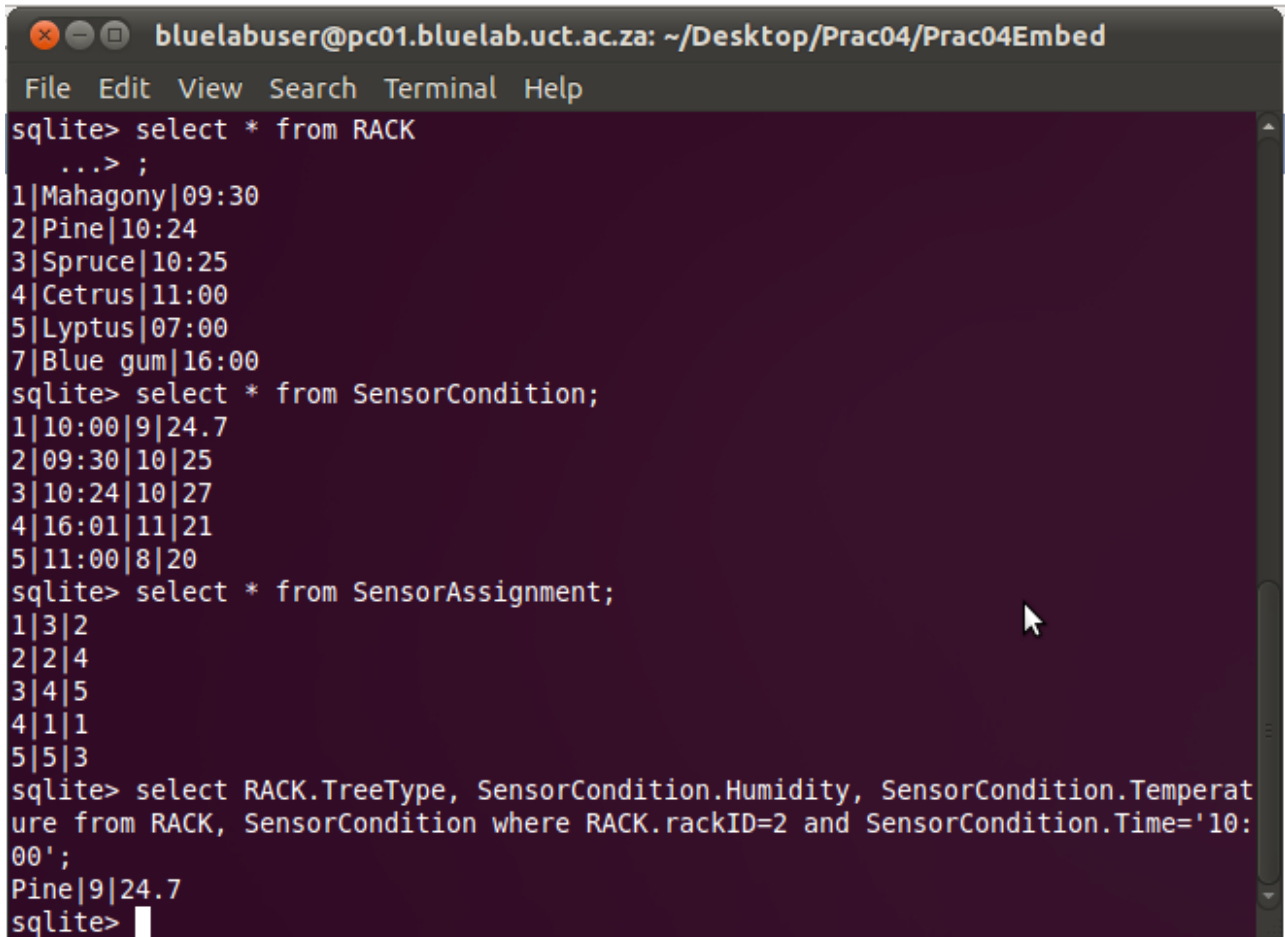


Figure 2.1 The database Entity Relational Diagram for plank warehouse database

## 2.3 Database creation and analysis

The database was created using SQLite program. The name of the database was **sensordb**. Once it was create, the command "create table RACK(rackID,TreeType,Time);" was used to create the first table, then the command "insert into RACK values(1,'Mahagony','09:30');" was used to create the first entry. The other entries were inserted in a similar fashion. Then the other two tables were also created using the first command. The table names were **SensorCondition** and **SensorAssignment**. The printout of the three tables is shown in Figure 2.2.



```
bluelabuser@pc01.bluelab.uct.ac.za: ~/Desktop/Prac04/Prac04Embed
File Edit View Search Terminal Help
sqlite> select * from RACK
...> ;
1|Mahagony|09:30
2|Pine|10:24
3|Spruce|10:25
4|Cetrus|11:00
5|Lyptus|07:00
7|Blue gum|16:00
sqlite> select * from SensorCondition;
1|10:00|9|24.7
2|09:30|10|25
3|10:24|10|27
4|16:01|11|21
5|11:00|8|20
sqlite> select * from SensorAssignment;
1|3|2
2|2|4
3|4|5
4|1|1
5|5|3
sqlite> select RACK.TreeType, SensorCondition.Humidity, SensorCondition.Temperature from RACK, SensorCondition where RACK.rackID=2 and SensorCondition.Time='10:00';
Pine|9|24.7
sqlite>
```

Figure 2.2 That shows the three tables, as well as the conditions and tree type on rack 2 at 10:24

Figure 2.2 also shows how the "select" command was used to find the values of humidity, temperature and tree type, found on rack 2 at time 10:00. The command and the variables are as follows:  
Select <the values of interest> from <tables that must be searched> where <special conditions that... must be taken into account>

The parameters that were returned are as follows:

- Tree Type: Pine
- Temperature: 24.7
- Humidity: 9

Attached to the report are the **sensordb.sql** file that contains the database and a text file (SensorDatabaseTables.txt) that contains the printout of the tables.

### 3. Conclusion

---

The practical made use of QEMU to emulate an ARM processor. A program for ARM architecture was cross compiled and run and the author obtained a good understanding of the process of configuring and cross compiling an ARM embedded system application on a PC. The program that was cross compiled was then loaded onto the emulated processor using QEMU and result was observed. Lastly, SQLite was used to create a database and find certain parameters, having the prior knowledge of factors such as rack number and time.

### 4. References

---

[1] A. Maakal, Database Design 101, makaal.com, 2008. [ONLINE]. Available: <http://www.makaal.com/maakalDB/Database101ERDpart3.htm> [Accessed: 22 May 2012].

[2] J.G. Zheng, Entity Relationship Diagram: Basics, 2010. [ONLINE]. Available: <http://jackzheng.net/teaching/cis3730/files/1.5-erd.pdf> [Accessed: 22 May 2012].