❖ Introduction

- This document contains a brief information about the different commands supported by KVPStorage application.
- It also discusses about the approach taken to customize and add KVPStorage application as a package in buildroot for QEMU ARM platform.
- Buildroot produces rootfs which contains KVPStorage application.
- The application is then run on the emulated environment (QEMU in this case)

❖ KVPStorage Application

Stores device and its relevant information in the form of key value pair database.

❖ Testing supported commands

Note:

- ✓ There should be an exact single space between command & different arguments
- ✓ No spaces are allowed after the last argument fed to each command
- ✓ If application is closed abruptly, changes made during the application run won't be committed to database
- ✓ Use Q or QUIT command to exit the application. This step is important to save the changes to database
- ✓ SET command overwrites the device info if device is already present in database

➢ SET/set

- Set device info in data database.

```
>
> SET DEV0 0
OK
> SET DEV1 1
OK
> SET DEV2 2
OK
> SET DEV3 3
OK
```

- If the number of arguments are not enough

```
>
> SET DEV4
Not enough arguments to SET
Usage: SET KEY VAL

> SET
Not enough arguments to SET
Usage: SET KEY VAL
```

- If the number of arguments are more than expected

```
> set VAL8 9 0
Too many arguments to SET
Usage: SET KEY VAL

>
>
```

➢ **GET/get**

- Get the device info for DEV0

```
>
>
> GET DEV0
0
>
>
```

- If number of arguments are not enough

```
>
> GET
Not enough arguments to GET
Usage: GET DEVICE_NAME
```

- If the number of arguments are more than expected

```
> get DEV0 0 1
Too many arguments to GET
Usage: GET DEVICE_NAME

> GET DEV0 0
Too many arguments to GET
Usage: GET DEVICE_NAME
```

➢ **DELETE/delete**

- To delete a device present in database

```
>
> DELETE DEV0
Deleting device DEV0
OK
>
```

- If a device is not present in database and DELETE/delete command is used

```
>
> delete DEV8
Operation not allowed!
DEV8 is not present in the device list
>
>
>
```

- If enough number of arguments are not passed

```
>
> DELETE
Not enough arguments to DELETE
Usage: DELETE DEVICE_NAME

>
```

- If the number of arguments are more than expected

```
>
> DELETE VAL0 0 0
Too many arguments to DELETE
Usage: DELETE DEVICE_NAME

> DELETE VAL0 0
Too many arguments to DELETE
Usage: DELETE DEVICE_NAME

>
```

- **LIST/list**

- To see the list of devices

```
> LIST

Available device list:

Dev             Config
DEV1            1
DEV4            4
DEV0            0
DEV3            3
DEV2            2
>
>
```

- If device list is empty

```
Please enter a command
> list
Device list is empty
>
>
>
```

- **Q/QUIT**

- To quit the application and commit changes to database on storage

```
> Q
Saving key value pair database in storage
Successfully saved the key value pair database to storage disk
Quitting database application
```

- **If a command is not supported, application shall emit below logs**

```
> del
Wrong command !!
Please enter a proper command

*****************************************************************
Allowed Commands:
*****************************************************************
SET/set         : Set the device info in database
GET/set         : Get the device info from database
DELETE/delete   : Delete a device from database
Q or QUIT       : Quit and save the database application
*****************************************************************
>
```

❖ Configuring and building buildroot for QEMU ARM

Note: Below steps are performed after installing all the packages required by buildroot. Please refer Chapter 2 of buildroot manual to get a list of mandatory packages
https://buildroot.org/downloads/manual/manual.html#requirement-mandatory

- Pull the buildroot sources from GIT

```
git clone https://github.com/buildroot/buildroot.git
```

- Create a separate folder in buildroot to keep a separate build relevant data for our package

```
anup@linux:~$ cd buildroot/
anup@linux:~/buildroot$ mkdir clay
```

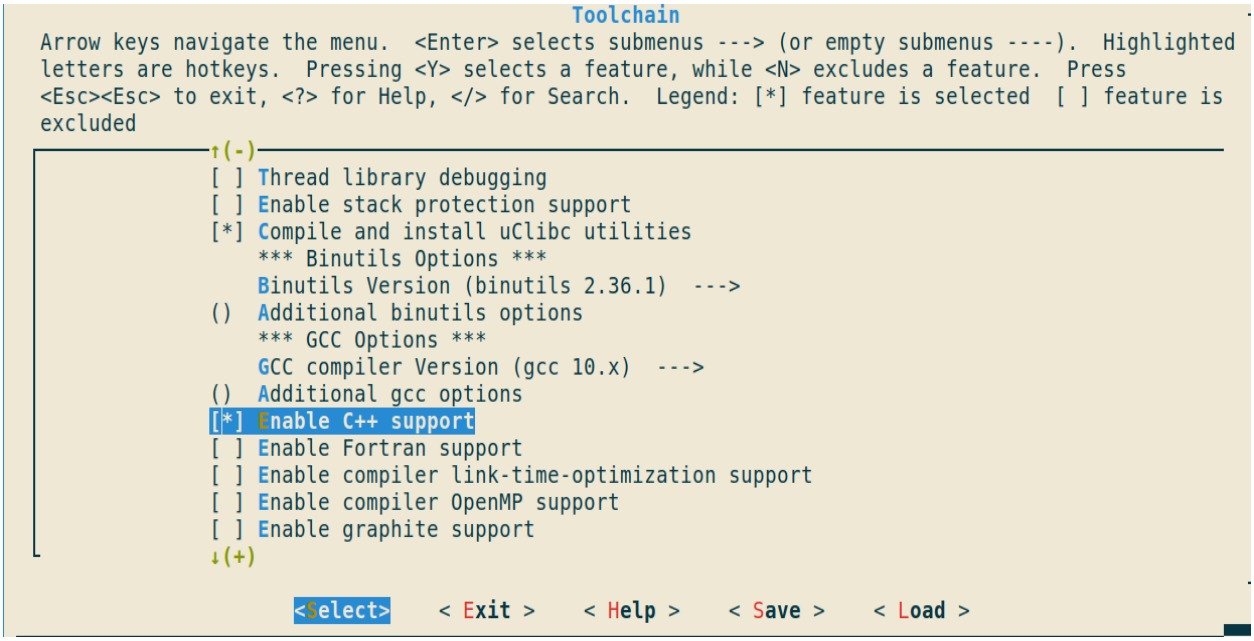
- Apply the default configuration file for QEMU arm versatile platform.

```
anup@linux:~/buildroot$ make O=clay qemu_arm_versatile_defconfig
```

- Make changes in the configuration if required.

```
anup@linux:~/buildroot$ make O=clay menuconfig
```

- For building C++ application, the highlighted option “Enable C++ support” should be enabled. Save the configuration.



- Start the build

```
anup@linux:~/buildroot$ make O=clay
```

- Once the build has completed, following directories will be populated inside “clay” folder

```
anup@linux:~/buildroot$ cd clay/
anup@linux:~/buildroot/clay$ ls
build  clay_defconfig  clay_defconfig.old  host  images  Makefile  staging  target
anup@linux:~/buildroot/clay$
anup@linux:~/buildroot/clay$
```

- “images” folder contains the images which will be flashed or run on target (QEMU emulator in our case)

```
anup@linux:~/buildroot/clay/images$ ls
rootfs.ext2  start-qemu.sh  versatile-pb.dtb  zImage
anup@linux:~/buildroot/clay/images$
```

❖ KVPStorage application as a package in buildroot

Below steps were followed to add KVPStorage application as a package in buildroot.

- Create a new folder under buildroot/package folder

```
anup@linux:~$ cd buildroot/package/
anup@linux:~/buildroot/package$ mkdir KVPStorage
```

- Now we need to create two files inside the newly created “KVPStorage” folder: Config.in and KVPStorage.mk

```
anup@linux:~/buildroot/package/KVPStorage$
anup@linux:~/buildroot/package/KVPStorage$ touch Config.in
```

```
anup@linux:~/buildroot/package/KVPStorage$
anup@linux:~/buildroot/package/KVPStorage$ touch KVPStorage.mk
```

- Now let’s populate these files with what will eventually add KVPStorage as a new package in buildroot and application can be activated or deactivated from menuconfig

config.in:

```
config BR2_PACKAGE_KVPSTORAGE
        bool "KVPSTORAGE"
        help
          Clay's Key Value pair storage application

comment "Needs C++ support to be activated in toolchain"
```

KVPStorage.mk

```
KVPSTORAGE_VERSION = 1.0
KVPSTORAGE_SITE = ./package/KVPStorage/src
KVPSTORAGE_SITE_METHOD = local


define KVPSTORAGE_BUILD_CMDS
        $(MAKE) CXX="$(TARGET_CXX)" LD="$(TARGET_LD)" -C $(@D)
endef

define KVPSTORAGE_INSTALL_TARGET_CMDS
        $(INSTALL) -D -m 0755 $(@D)/KVPStorage $(TARGET_DIR)/usr/bin
endef

$(eval $(generic-package))
```

In the above file, we need to select the cross compiler which is generated by buildroot. Buildroot fetches the application source from local and the path has to be mentioned in KVPStorage.mk. The mentioned path in above example is relative with respect to buildroot (./package/KVPStorage/src). We can also specify an absolute path. The application executable produced by build system will be installed in the path mentioned in above file ($(TARGET_DIR)/usr/bin)

- The directory structure of application package is shown below

```
anup@linux:~/buildroot/package/KVPStorage$ tree
.
|-- Config.in
|-- KVPStorage.mk
`-- src
    |-- KVPStorage.cpp
    `-- Makefile

1 directory, 4 files
```

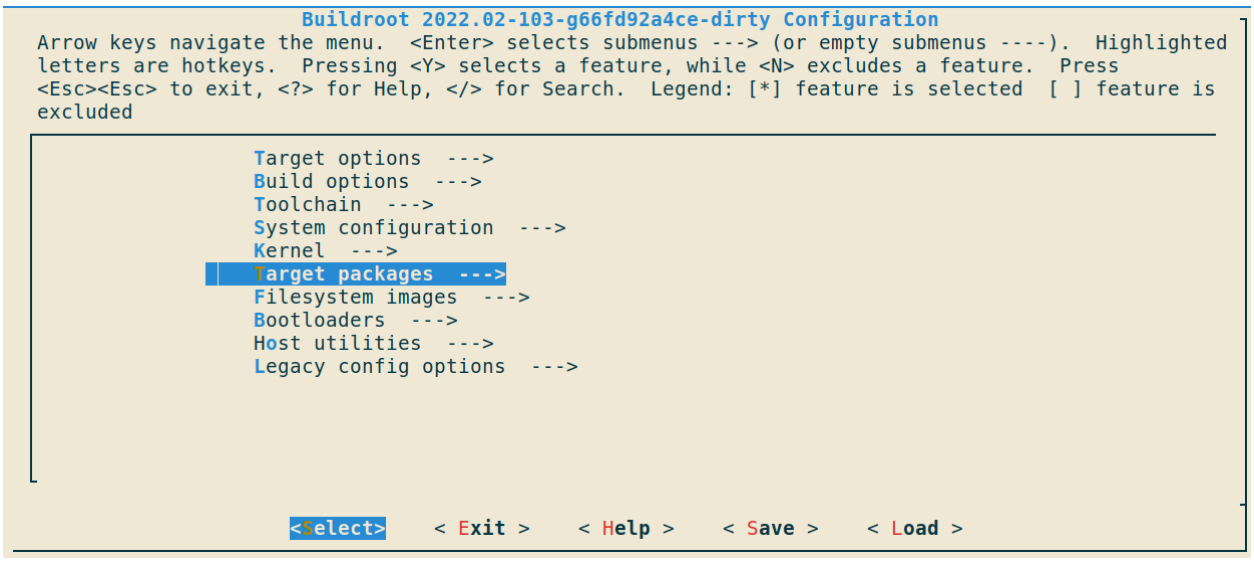
- Now we have to tell the package system that we want to be added. Hence we do the below modification in the package config file.

```
menu "Clay's Applications"
        source "package/KVPStorage/Config.in"
endmenu
```

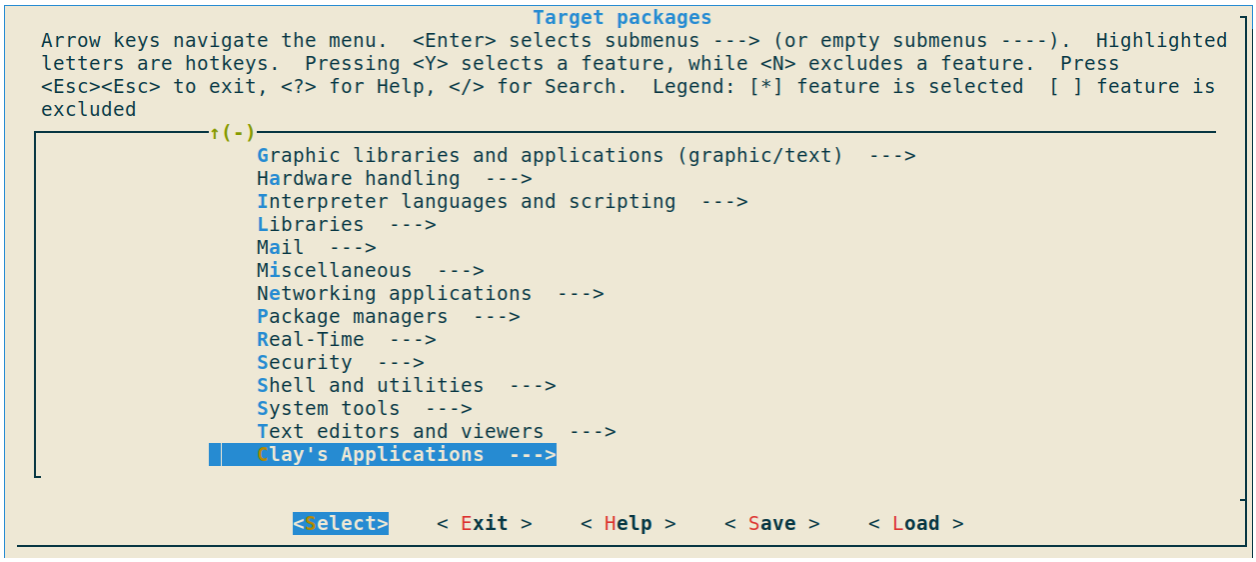
- After the above steps, we shall be able to see our application package in menuconfig. We need to enable KVPStorage package in menuconfig following below steps

```
anup@linux:~/buildroot$ make O=clay menuconfig
```

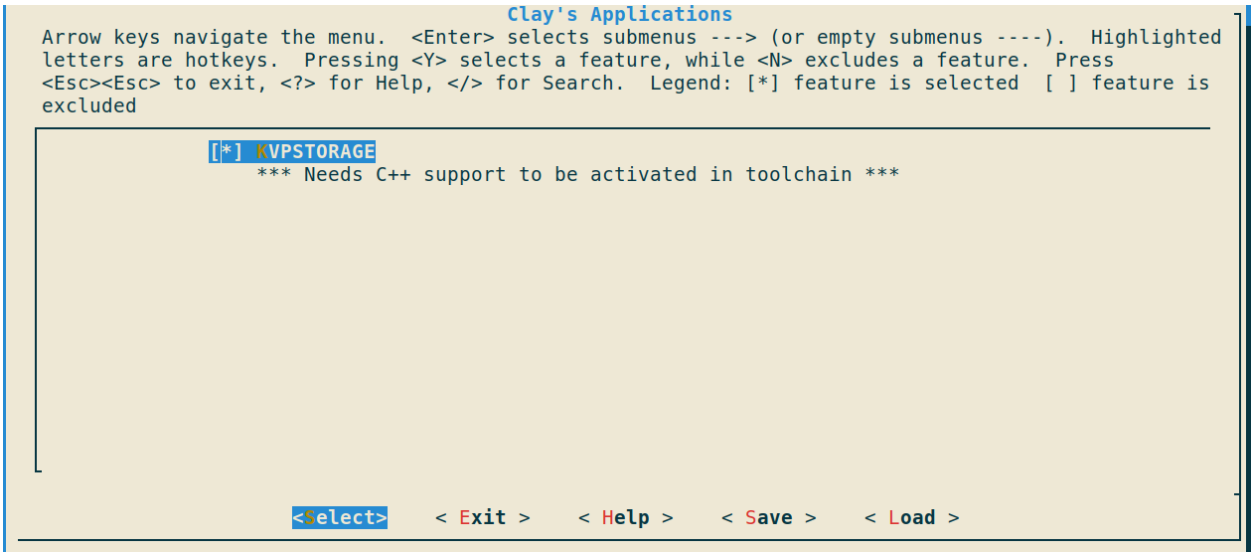
- Select “Target packages”



- Select Clay’s application



- Enable KVPSTORAGE package and save the configuration



- Now build the package

```
anup@linux:~/buildroot$ make O=clay
```

- When the build is finished, we can see the executable binary produced inside clay/target/usr/bin. This acts as a confirmation that our application package is built successfully

```
~/buildroot/clay/target/usr/bin$ ls
 crc32       find      killall     md5sum     printf      shred       truncate   w
 crontab     flock     KVPStorage  mesg       readlink    sort        ts         wc
 cut         fold      last        microcom   realpath    strings     tty        wget
 dc          free      ldd         mkfifo     renice      svc         uniq       which
 deallocvt   fuser     less        mkpasswd   reset       svok        unix2dos   who
 diff        getconf   logger      nl         resize      tail        unlink     whoami
 dirname     head      logname     nohup      seq         tee         unlzma     xargs
 dos2unix    hexdump   lsof        nproc      setfattr    telnet      unlzop     xxd
 du          hexedit   lspci       nslookup   setkeycodes test        unxz       xz
 eject       hostid    lsscsi      od         setsid      tftp        unzip      xzcat
 env         id        lsusb       openvt     sha1sum     time        uptime     yes
 expr        install   lzcat       passwd     sha256sum   top         uudecode
 factor      ipcrm     lzma        paste      sha3sum     tr          uuencode
```

- We can confirm if the produced binary is meant for ARM architecture

```
anup@linux:~/buildroot/clay/target/usr/bin$
anup@linux:~/buildroot/clay/target/usr/bin$ file KVPStorage
KVPStorage: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/l
d-uClibc.so.0, stripped
```

- This binary will be added into rootfs by the build system once the build is completed.

```
anup@linux:~/buildroot/clay/images$
anup@linux:~/buildroot/clay/images$ ls
rootfs.ext2  start-qemu.sh  versatile-pb.dtb  zImage
anup@linux:~/buildroot/clay/images$
```

- Now, we need to run our binary on emulator (QEMU)

✓ First install QEMU ARM

```
anup@linux:~/buildroot$ sudo apt install qemu-system-arm
[sudo] password for anup:
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

✓ Launch QEMU from buildroot directory using below command

```
anup@linux:~/buildroot$ qemu-system-arm -M versatilepb -kernel clay/images/zImage -dtb clay/images/versatile-
pb.dtb -drive file=clay/images/rootfs.ext2,if=scsi -append "root=/dev/sda console=ttyAMA0,115200" -nographic
```

✓ Once QEMU has booted successfully, below login screen appears

```
Freeing unused kernel image (initmem) memory: 148K
Kernel memory protection not selected by kernel config.
Run /sbin/init as init process
EXT4-fs (sda): warning: mounting unchecked fs, running e2fsck is recommended
EXT4-fs (sda): re-mounted. Opts: (null). Quota mode: disabled.
Starting syslogd: OK
Starting klogd: OK
Running sysctl: OK
Initializing random number generator: OK
Saving random seed: random: dd: uninitialized urandom read (512 bytes read)
OK
Starting network: Waiting for interface eth0 to appear..... timeout!
run-parts: /etc/network/if-pre-up.d/wait_iface: exit status 1
FAIL

Welcome to Buildroot
buildroot login: root
#
#
```

✓ Launch KVPStorage application

```
#
# KVPStorage
```

```
************************************************************
   Clay: Key Value Storage database loaded successfully
************************************************************

Please enter a command
>
```