Rev. 0.5.1, Aug. 2018

Samsung® KVSSD Quick Start Guide

Revision History

Revision No.	History	Draft Date	Remark
0.5	Initial Revision	Aug. 8, 2018	Preliminary

Table Of Contents

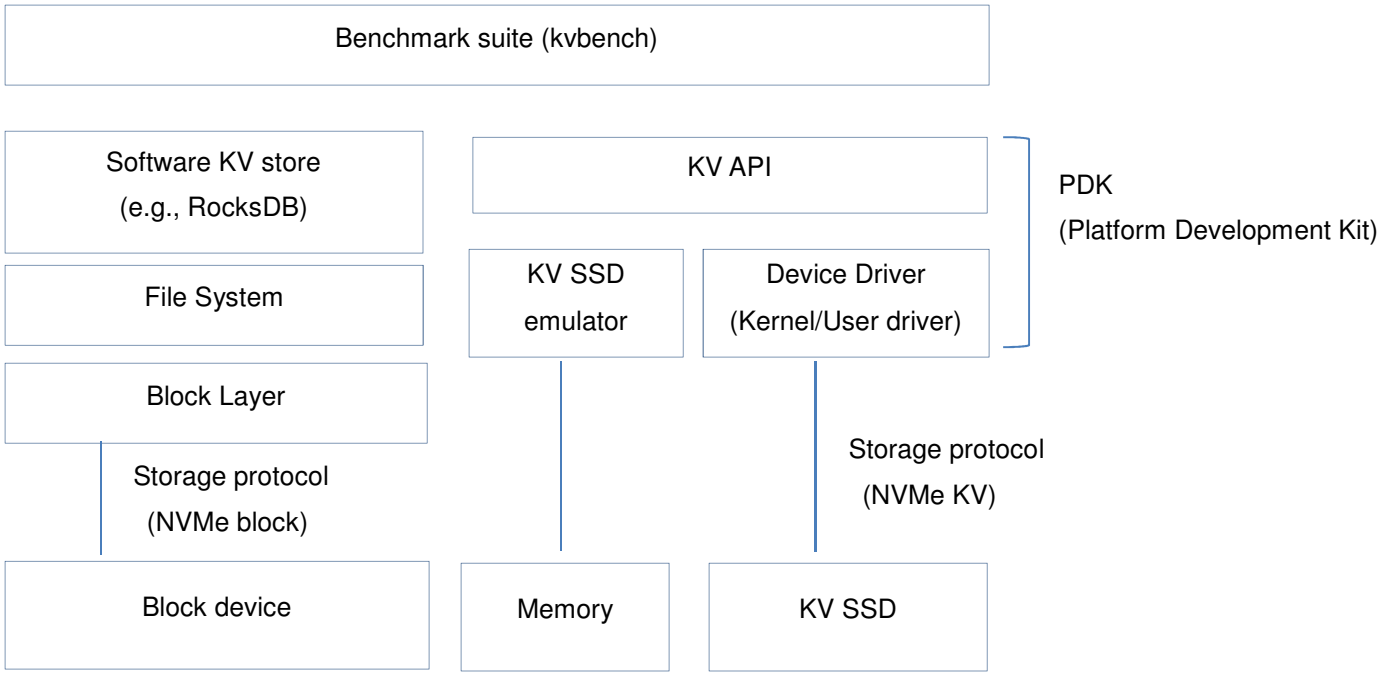
1.0 SCOPE

The `KVSSD` host software package includes the host software that operates with KV SSD. The package includes the API library, emulator, kernel device driver, and performance evaluation suite called kvbench. With the package, users can evaluate multiple application (e.g., RocksDB, Aerospike, etc.) performance on block device in addition to direct key-value stack performance on KV SSD.

Note that the performance of the emulator may not reflect the performance of the actual production product.

2.0 KV SSD SOFTWARE ARCHITECTURE

The KVSSD host software architecture is depicted as follows.



3.0 PACKAGES

The KVSSD host software package contains the following software/document modules.

```
KVSSD
    |- application
    |      |- kvbench : KV benchmark suite
    |
    |- PDK (Platform Development Kit)
    |      |- core
    |      |      |- src
    |      |             |- api : KV API (Application Programming Interface)
    |      |             |- device abstract layer
    |      |                    |- emulator : KV SSD emulator
    |      |                    |- kernel_driver_adapater : kernel driver adapter
    |      |
    |      |- driver : device driver
    |             |- PCIe
    |                    |- kernel driver : kernel device driver
    |                    |- user driver    : user space device driver
    |- spec
          |- KV API : KV API library spec
          |- NVMe : NVMe KV command spec
```

3.1 Platform Development Kit (PDK)

3.1.1 KV API

KV API provides a generic interface for users to communicate with Samsung Key-Value SSDs through different types of device drivers (user space, and kernel space), and a Samsung KeyValue SSD emulator. The main entry is in:

```
/KVSSD/PDK/core
```

3.1.2 Install dependencies

- `KVSSD/PDK/core/tools/install_deps.sh`

3.1.3 Kernel Device Driver

To use the kernel device driver (KDD), the user needs to compile and install NVMe modules for KV SSD, and compile the KV API with the KDD option.

1. Compile and install NVMe modules for KV SSD.
   - `cd KVSSD/PDK/driver/PCIe/kernel_driver/kernel_v<version>/`
   - `make clean`
   - `make all`
   - `sudo ./re_insmod`
   - `More details: KVSSD/PDK/driver/PCIe/kernel_driver/README`

2. Compile KV API with KDD option
   - `cd KVSSD/PDK/core`
   - `mkdir build && cd build`
   - `cmake -DWITH_KDD=ON ../`
   - `make -j24`
   - `kvapi library and test binaries are at: build/`
   - `More details: KVSSD/PDK/core/README`

3. Sample code test
   - `./sample_test -d device_path [-n num_ios] [-q queue_depth] [-o op_type] [-k klen] [-v vlen] [-p is_polling]`
   - `Write 1000 key-value pairs of key size 16-byte and value size 4096-byte to /dev/nvme0n1 with queue depth 64 in polling mode`

   ```
   ./sample_test -d /dev/nvme0n1 -n 1000 -q 64 -o 1 -k 16 -v 4096 -p 1
   ```

4. Limitations
   - There are limitations on kernel versions supported.
     i. kernel_v3.10
     ii. kernel_v4.9.5

iii. kernel_v4.13

iv. kernel_v4.15

3.1.4 User Space Device Driver

To use the user space device driver (UDD), the user needs to compile the user driver, and compile the KV API with the UDD option. Samsung KV SSD user space driver is available at:

https://github.com/OpenMPDK/uNVMe/tree/master/driver

There is a pre-built udd driver in `KVSSD/PDK/core/lib/libkvnvmedd.a` with `gcc version 5.4.0.`
The user can skip step 1 if using the same GCC version.

1. Compile SPDK UDD.

```
- git clone https://github.com/OpenMPDK/uNVMe.git
- cd uNVMe
- make intel
- cd uNVMe/driver/core
- make clean && make
- cp uNVMe/driver/core/libkvnvmedd.a KVSSD/PDK/core/lib
```

2. Compile KV API with user space driver option

```
- cd KVSSD/PDK/core
- mkdir build && cd build
- cmake -DWITH_SPDK=ON ../
- make -j24
- kvapi library and test binaries are at: build/
- More details: KVSSD/PDK/core/README
```

3. Sample code test

```
- Setup spdk environment before running spdk driver tests
      o  /KVSSD/PDK/core/tools/setup.sh

- ./sample_test -d device_path [-n num_ios] [-q queue_depth] [-o op_type] [-k klen] [-
   v vlen] [-p is_polling]

- Write 1000 key-value pairs of key size 16-byte and value size 4096-byte to
   0000:06:00.0 with queue depth 64 in polling mode

  ./sample_test -d 0000:06:00.0 -n 1000 -q 64 -o 1 -k 16 -v 4096 -p 1
```

4. Limitations
    - UDD only works in interrupt mode

3.1.5 Emulator

To use the in-memory key-value SSD emulator, the user needs to compile the KV API with the EMU option.

1. Compile KV API with EMU option

   - cd KVSSD/PDK/core
   - mkdir build && cd build
   - cmake -DWITH_EMU=ON ../
   - make -j24
   - kvapi library and test binaries are at: build/

   * Emulator configuration details: refer to the Appendix for kvssd_emul.conf or KVSSD/PDK/core/README

2. Sample code test

   - ./sample_test -d device_path [-n num_ios] [-q queue_depth] [-o op_type] [-k klen] [-v vlen] [-p is_polling]

   - Write 1000 key-value pairs of key size 16-byte and value size 4096-byte to /dev/kvemul with queue depth 64 in polling mode

     ./sample_test -d /dev/kvemul -n 1000 -q 64 -o 1 -k 16 -v 4096 -p 1

3. Limitations

   - The emulator only works in polling mode. Interrupt mode will be supported soon.

3.2 KV SSD benchmark suite

KVbench is a benchmark suite for embedded key-value storage engines and based on a sophisticated workload generation which is more realistic than performing a bunch of read/write operations. It is based on ForestDB-benchmark tool, with an extension of KV SSD API support.

KVbench supports following four types of key-value engines:

- RocksDB (Linux filesystem)
- RocksDB (SPDK)
- Samsung's KV SSD direct access
- Aerospike

More details are available at: KVSSD/application/kvbench/README

3.2.1 RocksDB on Linux filesystem

1. Build RocksDB from source code
   - Download rocksdb source code from https://github.com/facebook/rocksdb
   - cd rocksdb
   - make static_lib
   * RocksDB tested in version of v5.0.2, v5.6.1

2. Build rocksdb_bench
   - cd kvbench
   - mkdir build_rxdb && cd build_rxdb
   - cmake -DCMAKE_INCLUDE_PATH=<YOUR ROCKSDB HEADER FILE DIR> -DCMAKE_LIBRARY_PATH=<YOUR ROCKSDB LIB FILE DIR> ../
   - make rockdb_bench

3.2.2 RocksDB on SPDK

1. A working version of source code for intel SPDK and RocksDB (SPDK) is included in the kvbench package

2. Build SPDK library

   - `cd kvbench/spdk/spdk`
   - `./configure`
   - `make`
   - `cd ../`

3. RocksDB on SPDK

   - `cd kvbench/spdk/rocksdb`
   - `make static_lib`
   - `cd ../../`

4. build blobfs_rocksdb_bench
   - `cd kvbench`
   - `mkdir build_spdk && cd build_spdk`
   - `cmake ../`
   - `make blobfs_rocksdb_bench`

3.2.3 KV Stack on KV SSD (Direct operation to KV SSD)

1. Download and build the kvapi library

   - Refer to section 3.1

2. Build kv_bench

```
- cd kvbench
- mkdir build_kv && cd build_kv
- cmake -DCMAKE_INCLUDE_PATH=/KVSSD/PDK/core/include -
    DCMAKE_LIBRARY_PATH=/KVSSD/PDK/core/build ../
- make kv_bench
```

3.2.4 Aerospike

1. Download & install aerospike server

   https://www.aerospike.com/docs/operations/install/linux/ubuntu

2. build as_bench
   - cd kvbench
   - mkdir build_as && cd build_as
   - cmake ../
   - make as_bench

3.2.5 Benchmark run

Run kv_bench as an example.

1. Create & modify the cpu config file for the first time

   - LD_LIBRARY_PATH= ./kv_bench -c # This will generate default cpu.txt file

   - Modify cpu.txt for (nodeid, coreid, deviceid) mapping if needed

   - This cpu.txt only needs to be generated once and can be used for all tests on the same system. User can copy it to other 'build' directories where the executable files reside (e.g. build_as for as_bench)

2. Modify bench_config.ini for workloads (refer to the Appendix for bench_config.ini)

3. Setup environment

   - RocksDB on Linux filesystem: file system needs to be created and mounted.
   - RocksDB on SPDK: spdk environment needs to be set up.
   - KV Stack on KV SSD: driver environment needs to be set up, refer to section 3.1 for details.
   - Aerospike: aerospike service needs to be started.

   * More configuration details: KVSSD/application/kvbench/README

4. Run benchmark

   - LD_LIBRARY_PATH=/KVSSD/PDK/core/build ./kv_bench -f bench_config.ini

APPENDIX

bench_config.ini

This section describes the KV benchmark suite key configuration parameters (bench_config.ini). For more details, please refer to:

KVSSD/application/kvbench/README

```
=
[document]
ndocs = 100     # insert 100 kv pairs during `load`

[system]
key_pool_size = 128      # number of units to create for key mempool, should be larger than queue_depth
key_pool_unit = 16       # size of units per key mempool; this should match the key size
key_pool_alignment = 4096  # memory will be aligned in this unit
value_pool_size = 128     # same as above
value_pool_unit = 4096    # same as above
value_pool_alignment = 4096   # same as above
device_path = /dev/nvme0n1    # device path for block devices

[kvs]
device_path = /dev/kvemul # device path for kv ssd. When using KV SSD, 'device_path' under [system] & [kvs] should both be set properly.
emul_configfile = /tmp/kvemul.conf   # path to the emulator configuration file
queue_depth = 64   # queue depth when using ASYN IO
core_ids = 1,3,5  # core ids for submission queue when using spdk driver, one core per device. In this case, core 0 for device 0, core 2 for device 1, and core 4 for device 2. This core ids should match the configuration in cpu.txt. For kernel driver this could be ignored.
cq_thread_ids = 2,4,6   # core ids for completion queue when using spdk driver. For kernel driver this could be ignored.
write_mode = async    # sync/async IO mode for kv/aerospike, sync mode for rocksdb
with_iterator = true   # running iterator mixed with read/write workloads
          = false  # no iterator
          = alone  # running iterator alone without other workloads until end of the iteration
```

```
iterator_mode = key    # [DEFAULT] iterator command gets only key entries without values
           = value  # iterator command gets key and value pairs
is_polling = true   # kv ssd ASYNC IO completion handled in polling mode
        = false  # kv ssd ASYNc IO completion handled in interruption mode

[population]
nthreads = 1   # number of client threads each device have during `load`, set 1 for KV SSD
seq_fill = true  # sequential insertion; false: random insertion

[threads]
readers = 1
writers = 2  # If [operation] read_write_insert_delete total ratio is equal to 100, each thread
will run mixed workload based on ratio control; Otherwise, will have dedicated readers/writers
to   run   without   raio   control.   Also   see   below   [opertion].   For   example,   if
read_write_insert_delete = 50:40:10:0. Each DB will have total (readers + writers) 3 threads,
each   thread   runs   mixed   workload   of   50%   read,   40%   update,   10%   insert.   If
read_write_insert_delete = 0:1000:0:0, each DB will have 1 reader, 2 writers, each thread
running its own operations. Only use total 1 thread for each KV SSD.

[key_length]
distribution = fixed   # key size in fixed length
fixed_size = 16

[body_length]
distribution   = uniform, normal   # as defined
           = fixed               # fixed value size as defined below 'fixed_size'
           = ratio         # variable value size as defined below 'value_size' with each having
a ratio as def fixed_size
fixed_size  = 4096          # value size in fixed length
value_size = 512,2048,4096
value_size_ratio = 10:50:40  # variable value size of 512, 2048 and 4096 byte with ratio of
10:50:40; Support maximum 5 varible lengths.

[operation]
duration = 5    # run benchmark for 5 seconds after insertion
#nops = 1000   # run benchmark for total 1000 operations after insertion, kvbench will run
under either 'duration' or 'nops' mode. If insertion ratio (see read_write_insert_delete below)
is larger than 0, kvbench should run under ‘nops’ mode.
read_write_insert_delete = 50:50:0:0  # operation type ratios for read/write/insert/delete.
```

```
If 'insert' ratio is larger than 0, set 'nops' instead of 'duration' for benchmark test.
```

kvssd_emul.conf

`kvssd_emul.conf` has two sections for emulator configuration.

The first section is the general section. It contains `capacity, polling, keylen_fixed, and use_iops_model`. You can use `capacity` to specify the max capacity of KVSSD emulator, once the capacity is reached, the emulator will return a capacity full error. `polling` is used to overwrite the device initialization setting of field `is_polling` in structure `kv_device_init_t`, which is used by `kv_initialize_device()`. `keylen_fixed` is used to indicate if a key length field should be included for iteration output buffer. If `keylen_fixed` is set to be "`true`", then the key length field is not included assuming the API caller will know the length of key in iteration output buffer. Otherwise, the key length field is included in the iteration output buffer, preceding the value of each key. `use_iops_model` is used to enable or disable IOPS modeling within the KVSSD emulator. When it's set to false, KVSSD emulator will bypass IOPS modeling and perform faster than a real device.

`iops_modeling` section should be treated as a read only section, end users shouldn't modify this section without instructions from Samsung.

Please see a sample KVSSD emulator configuration file below:

```
## default configuration options
[ general ]
    # capacity per device, only use GB or MB
    # default capacity is unlimited if not specified here
    # capacity = 7GB

    # use device in polling mode or interrupt mode
    # false means using interrupt mode
    # if specified, this will overwrite initialization setting through code.
    # polling = false
    polling = true

    # fixed key length
    # if your keys are not fixed in length
    # please change it to be false, this will only affect iterator key output
    keylen_fixed = true

    # use IOPS model, by default it is set to be true
    use_iops_model = true
```

```
# PLEASE DON'T CHANGE THESE PARAMETERS UNLESS INSTRUCTED
# IOPS model parameters
# 3 degree polynomial linear model feature coefficient list
# first parameter is the intercept, the rest are listed from lower terms to higher terms
[ iops_model ]
   parameters  =  93.55536664,  9.73333650e-03,    -7.48332780e-04,    -1.02151073e+01,    -
3.25414892e+01,  4.27551652e+01,   -1.37097247e-03,    5.35734352e-04,    6.63934171e-05,  -
5.98786540e-03,     6.59822790e+01,    -9.08029926e+01,     1.46061898e+01,  1.26562873e+01,
4.56136757e+01,    -1.74634825e+01,     3.45057316e-12,  1.37099315e-03,     1.37105773e-03,
1.37092722e-03,   -7.66785198e-03,  1.28200988e-02,   -4.76358522e-03,   -5.57416593e-03,   -
4.75173882e-03,  3.58063105e-03,   -9.87217038e+00,    6.72385443e+01,    8.61728612e+00,  -
1.39270618e+02,   -1.87756101e+01,    2.47645398e+01,    7.51414174e+01,  7.67858478e+01,   -
1.23967843e+01,  -2.98320336e+01
```