Rev. 0.8.0, July 2019

Samsung® KVSSD Quick Start Guide

Revision History

Revision No.	History	Draft Date	Remark
0.5	Initial Revision	Aug. 8, 2018	Preliminary
0.6.0	Refined revision	Nov. 2, 2018	
0.8.0	Update the supported KDD and UDD-related information	Jul. 8, 2019	

Table Of Contents

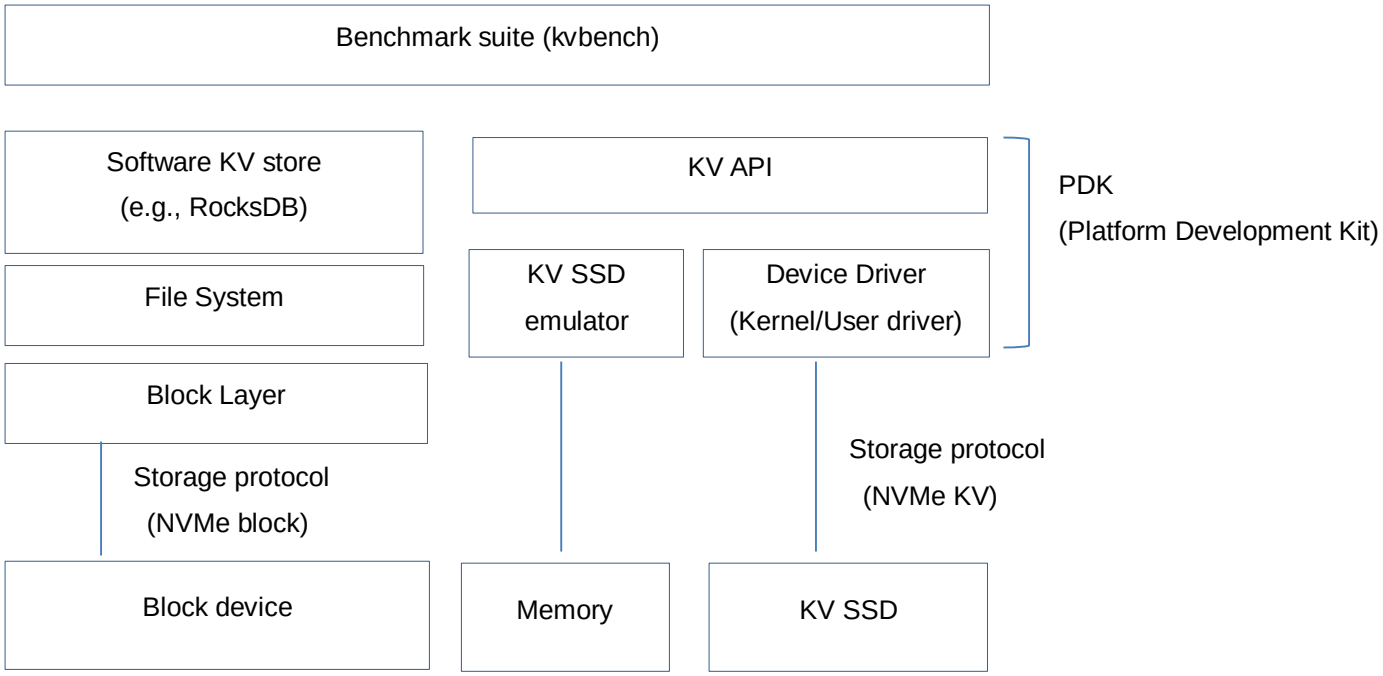
1.0 SCOPE

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

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`
- `gcc version must be equal to or newer than v5.0`

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.

```
1) cd KVSSD/PDK/driver/PCIe/kernel_driver/kernel_v<version>/
2) make clean
3) make all
4) sudo ./re_insmod.sh
5) More details: KVSSD/PDK/driver/PCIe/kernel_driver/README
```

2. Compile KV API with KDD option

```
1) cd KVSSD/PDK/core
2) mkdir build && cd build
   • If build directory already existed, all files in the directory should be deleted.
3) cmake -DWITH_KDD=ON ../
4) make -j24
5) kvapi library(libkvapi.so) and test binaries(sample_code_async and sample_code_sync)
   are at: build/
6) More details: KVSSD/PDK/core/README
```

3. Sample code test

```
• Users must run sample codes in root privileges (users should be root or use sudo to
  run sample codes).
• ./sample_code_sync –h to get usage
• sudo ./sample_code_sync -d device_path [-n num_ios] [-o op_type] [-k klen] [-v vlen]
  [-t threads]
• sudo ./sample_code_async -d device_path [-n num_ios] [-q queue_depth] [-o op_type]
  [-k klen] [-v vlen]
• Write 1000 key-value pairs of key size 16-byte and value size 4096-byte to
  /dev/nvme0n1 with queue depth 64
```

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

4. Limitations
    - Following Linux Kernel are supported.
        i. CentOS 7.2 Kernel_v3.10.0-327
        ii. CentOS 7.4 Kernel_v3.10.0-693
        iii. Ubuntu 16.04 Kernel_v4.4.0-98
        iv. Ubuntu 16.04 Kernel_v4.4.0-141
        v. Ubuntu 16.04 Kernel_v4.9.5
        vi. Ubuntu 16.04 Kernel_v4.13
        vii. Ubuntu 16.04 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/driver

There is a pre-built udd driver in `KVSSD/PDK/core/lib/libkvnvmedd.a (v18.11)` with `gcc version 5.4.0.` The user can skip step 1 if using the same GCC version.

1. Compile SPDK UDD

```
1) git clone https://github.com/OpenMPDK/uNVMe.git
2) cd uNVMe
3) sudo ./script/pkgdep.sh
4) ./make.sh intel
5) ./make.sh driver
6) cp driver/core/libkvnvmedd.a KVSSD/PDK/core/lib
```

2. Compile KV API with user space driver option

```
1) cd KVSSD/PDK/core
2) mkdir build && cd build
   • If build directory already exist, need delete all of the file in this directory
3) cmake -DWITH_SPDK=ON ../
4) 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
  sudo KVSSD/PDK/core/tools/setup.sh

• sudo ./sample_code_sync -d device_path [-n num_ios] [-o op_type] [-k klen] [-v
  vlen] [-t threads]

• sudo ./sample_code_async -d device_path [-n num_ios] [-q queue_depth] [-o
  op_type] [-k klen] [-v vlen]

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

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

4. Limitations
   1) UDD does not work in mixed (SYNC & ASYNC) mode. Either Sync or Async is setup globally during device initialization

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. KVSSD emulator is only used for functional testing purpose, and does not have any performance indication on real KV SSD device.

1. Compile KV API with EMU option

```
1) cd KVSSD/PDK/core
2) mkdir build && cd build
3) cmake -DWITH_EMU=ON ../
4) 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

   - sudo ./sample_code_sync -d device_path [-n num_ios] [-o op_type] [-k klen] [-v vlen] [-t threads]
   - sudo ./sample_code_async -d device_path [-n num_ios] [-q queue_depth] [-o op_type] [-k klen] [-v vlen]

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

```
sudo ./sample_code_async -d /dev/kvemul -n 1000 -q 64 -o 1 -k 16 -v 4096
```

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)
- 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

```
1) Download rocksdb source code from https://github.com/facebook/rocksdb
2) cd rocksdb
3) make static_lib
* RocksDB was tested in version of v5.0.2, v5.6.1
```

2. Build rocksdb_bench

```
1) cd KVSSD/application/kvbench
2) mkdir build_rxdb && cd build_rxdb
3) cmake -DCMAKE_INCLUDE_PATH=<YOUR ROCKSDB HEADER FILE DIR> -
   DCMAKE_LIBRARY_PATH=<YOUR ROCKSDB LIB FILE DIR> ../
4) make rocksdb_bench
```

3.2.2 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

```
1) Install dependencies
   $(CentOS) sudo yum install snappy-devel libev-devel bzip2-devel lz4-devel
   gflags-devel jemalloc-devel

   $(Ubuntu) sudo apt-get install libsnappy-dev libev-dev libbz2-dev liblz4-dev
   libzstd-dev libjemalloc-dev libnuma-dev libgflags-dev

   Download and Install zstd https://github.com/facebook/zstd

2) Goto kvbench directory (KVSSD/application)
   cd kvbench
3) mkdir build_kv && cd build_kv
4) cmake -DCMAKE_INCLUDE_PATH=KVSSD/PDK/core/include -
   DCMAKE_LIBRARY_PATH=KVSSD/PDK/core/build/libkvapi.so ../
   * include and library directory path must be updated according to your directory
   structure. Directory path should be absolute path. Relative path would not work.
5) make kv_bench
```

3.2.3 Aerospike

1. Download & install aerospike server

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

2. Start aerospike service
   ```
   sudo service aerospike start
   ```

3. Build as_bench
   ```
   1) cd KVSSD/application/kvbench
   2) mkdir build_as && cd build_as
   3) cmake ../
   4) make as_bench
   ```

3.2.4 Benchmark run

Run kv_bench as an example.

* assuming the current directory is /KVSSD/application/kvbench/build_kv

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

   - LD_LIBRARY_PATH=/KVSSD/PDK/core/build ./kv_bench -c # This will generate default cpu.txt file

   * LD_LIBRARY_PATH must be updated according to your directory structure. Directory path should be absolute path. Relative path would not work.

   - 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.
   - 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
     - LD_LIBRARY_PATH must be updated according to your directory structure. Directory path should be absolute path. Relative path would not work.
     - When running kv_bench on the KV emulator, the emul_configfile field must be updated. The path should be absolute path.

5. Benchmark results

   Performance measurement result files are in ./logs directory.

   - KVS-ops.txt: result summary that is same as those printed to the screen, including configuration parameters, total run time, average throughput, tail latency, etc.
   - Insertion phase:
     i. KVS-insert.latency.csv: latency measured during ‘insertion’ phase. IO latency is measured in a sampling rate defined in [latency_monitory] ‘rate’ section (in the unit of Hertz). Result .csv file shows

the latency percentile (1% - 99%) for each type of operations. e.g. a line ‘50,24,20,5’ in the file indicates 50% percentile latency for write, read & delete is 24us, 20us, and 5us respectively.

ii. KVS-insert.ops.csv: throughput measured during ‘insertion’ phase. Throughput is measured in a time interval defined in [latency_monitor]’print_term_ms’ section (in the unit of millisecond). Result .csv file shows the throughput over time. e.g. a line ’30,160,80,5000’ in the file indicates at runtime of 30 second, the overall average throughput is 160ops/sec, instant throughput during the last ‘print_term_ms’ period is 80op/sec, and total operations finished is 5000.

- Benchmark phase:
  i. KVS-run.latnecy.csv: similar to KVS-insert.latency.csv
  ii. KVS-run.ops.csv: similar to KVS-insert.ops.csv
- Limitations:
  i. Direct operation to KV SSD does not capture IOs (disk bytes written) per process. This stats will be updated in future release.

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     # number of records. insert 100 KV pairs during `load`

[system]
key_pool_size = 128     # number of units to create for key mempool, it should be larger than queue_depth
key_pool_unit = 16       # size of units per key mempool; this should match the key length; If testing with various key lengths, this unit should be equal to or larger than the maximum key size;
key_pool_alignment = 4096  # memory will be aligned in this unit
value_pool_size = 128     # number of units to create for value mempool, it should be larger than queue_depth
value_pool_unit = 4096     # size of units per value mempool; this should match the value length; If testing with various value lengths, this unit should be equal to or larger than the maximum key size;
value_pool_alignment = 4096   # memory will be aligned in this unit
device_path = /dev/nvme0n1    # device path for block devices

[kvs]
device_path = /dev/nvme0n1 # device path for kv ssd. When using KV SSD, 'device_path' under [system] & [kvs] should both be set properly.
emul_configfile = kvssd_emul.conf
# This file resides in PDK/core/. When running kv_bench on emulator, the full path should be
 set correctly.
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 1 for device 0, core 3 for device 1, and core 5 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 is only supported under async mode.
iterator_mode = key    # [DEFAULT] iterator command gets only key entries without values
           = value  # iterator command gets both key and value pairs

[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, it will have dedicated readers/writers to run without ratio 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 (total ratio(e.g., 1000) is bigger than 100), each DB will have 1 reader, 2 writers, each thread running its own operations for block device. Total 1 thread (one write thread only) should run IO in each KV SSD.

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

[body_length]       # value size
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 defined in value_size_ratio
fixed_size  = 4096          # value size in fixed length
value_size = 512,2048,4096  # various value size in bytes
value_size_ratio = 10:50:40  # variable value size of 512, 2048 and 4096 byte with ratio of
```

```
10:50:40; Support maximum 5 variable 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 must run under ‘nops’ mode.
batch_distribution = uniform # key space distribution: uniform; zipfian;
read_write_insert_delete = 50:50:0:0  # operation type ratios for read/write/insert/delete. If 'insert' ratio is larger than 0, 'nops' instead of 'duration' must be used 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.

* No IOPS modeling is supported in the current version.

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 = 100GB

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

    # 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 = false

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