

*Whitepaper*

*Implementing with*

# **INTEL SLIM BOOTLOADER**

*A Lightweight and Secure Bootloader  
for Embedded Systems*



By:

**Rajat Dongre**

R&D Engineer

# Contents

1. Overview	2
2. Introduction	2
3. Features	2
4. Architecture	2
5. Software and Hardware Environment setup	3
6. QEMU	3
7. Implementations of SBL	4
1) Compile time build environment	4
2) SBL Build Steps	4
- Download the SBL source code through Git using the following command	4
- SBL Keys Generation: Steps to be followed to generate SBLKeys	4
- Build SBL	5
- Build Outputs	6
8. Boot Linux with U-Boot on Emulator:	8
a. Build U-Boot and obtain u-boot-dtb.bin	8
b. Prepare Slim Bootloader	8
c. Build Instruction for QEMU target	8
d. Test Linux booting on QEMU target	10
9. Secure Boot	11
10. Troubleshoot	13
11. Conclusion	13
12. Reference	13

# 1. Overview

Slim Bootloader is a lightweight and secure bootloader that is designed to provide a fast, reliable, and secure boot experience on embedded devices. It is based on the UEFI secure boot specification and works with platforms that have UEFI firmware to boot Linux or other operating systems. This white paper provides an in-depth explanation of the Slim Bootloader design document, including the architecture, features, specifications, and implementation details.

# 2. Introduction

Embedded systems are becoming increasingly common in a wide range of applications, from industrial automation and robotics to smart homes and consumer electronics. These systems often require a secure and reliable boot process to ensure that the device is running the intended software and is protected against malicious attacks. A bootloader is an essential component of the boot process, responsible for loading the operating system and initializing the hardware devices. Slim Bootloader is an open-source boot firmware, built from the ground up to be small, secure, and optimized running on Intel x86 architecture.

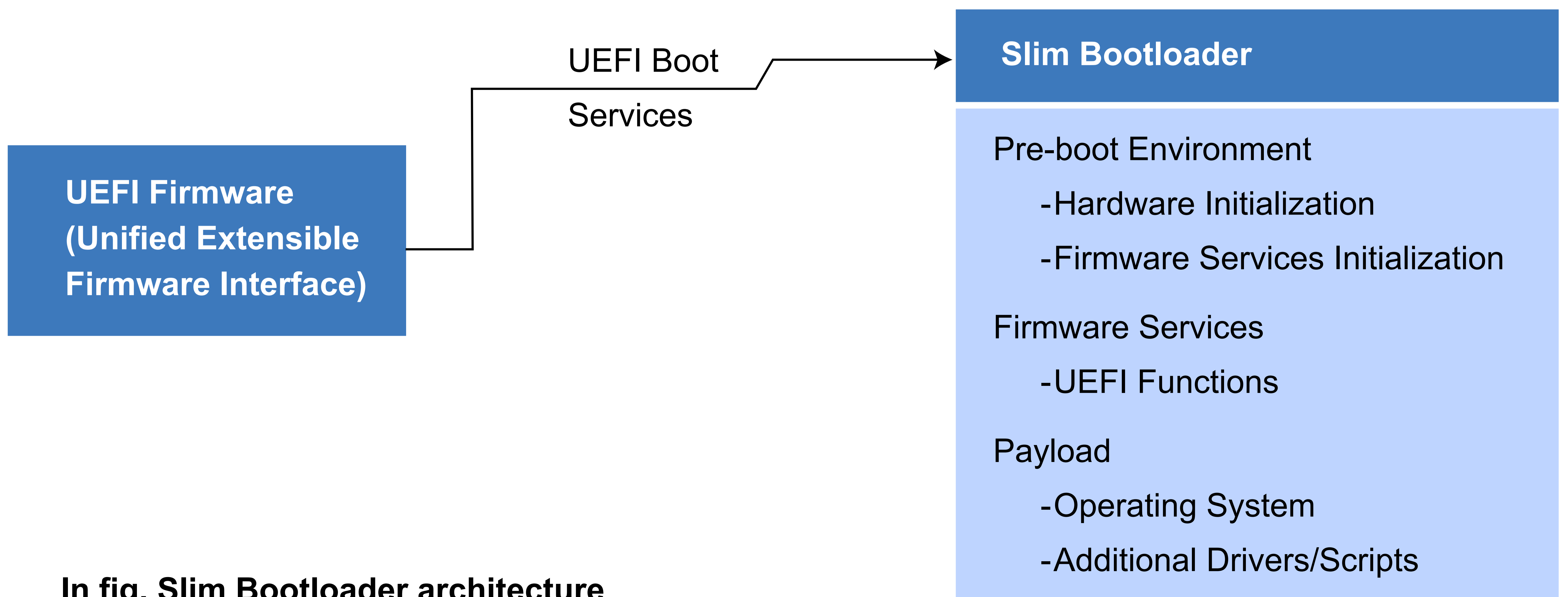
# 3. Features

Slim Bootloader is designed to be:

- 👉 Small
- 👉 Fast
- 👉 Secure
- 👉 Extensible
- 👉 Configurable

# 4. Architecture

Slim Bootloader is based on the UEFI Secure Boot specification, which provides a standard way to ensure the integrity and authenticity of the boot process.



In fig. Slim Bootloader architecture

The diagram shows the three main components of the Slim Bootloader:

- 👉 The initial boot services used by Slim Bootloader to initialize the hardware and load the firmware services are provided by the UEFI firmware.
- 👉 The pre-boot environment provides the minimal set of functions required to initialize the hardware devices and load the firmware services. This includes hardware initialization routines such as setting up the memory and processor, as well as initialization of any additional hardware devices required by the payload.
- 👉 The firmware services provide a set of UEFI functions that can be used by the payload to interact with the hardware devices and the operating system. These functions include file system access, device driver loading and configuration, and network communication.
- 👉 The bootloader's primary element, the payload, is responsible for loading the operating system as well as any additional drivers or programs. We can use external payloads like grub, u-boot, etc.

## 5. Software and Hardware Environment setup

The following conditions must be fulfilled to utilize Slim Bootloader:

- 👉 Slim Bootloader source code and its build tools list are mentioned in the section (mention section here: 7.1.b)
- 👉 A hardware platform that supports multi-boot or secondary BIOS.
- 👉 To load the operating system (Linux, Windows, etc.), Osloader(default), or any other UEFI payload such as u-boot (here we are using u-boot).
- 👉 QEMU for testing purposes

## 6. QEMU

- 👉 QEMU is a free and open-source machine emulator and virtualizer. It can run operating systems and programs made for one machine on a different machine.
- 👉 We are using QEMU to emulate hardware and load a Yocto image to test our bootloader.
- 👉 Please go through Qemu documentation for more details: <https://www.qemu.org/docs/master/>
- 👉 We have mentioned details for running QEMU for our bootloader in section 7.2.d

# 7. Implementations of SBL

The slim bootloader implementation steps are as follows:

## 1) Compile time build environment

a) SBL build is supported on both Windows and Linux environments. Here we will be using the Linux environment to explain SBL.

b) Building in Linux:

i. Supported environment: **Ubuntu Linux 18.04 LTS**

ii. Install the following software:

1. GCC 7.3 or above
2. Python 3.6 or above
3. NASM 2.12.02 or above
4. IASL 20190509
5. OpenSSL
6. Git

iii. Build Tools Download - Ubuntu

1. Run the following command to install the required packages through the terminal:

```
$ sudo apt-get install -y build-essential iasl python uuid-dev nasm  
openssl gcc-multilib qemu git
```

## 2) SBL Build Steps

a) Download the SBL source code through Git using the following command:

```
$ git clone https://github.com/slimbootloader/slimbootloader.git  
$ cd slimbootloader
```

```
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader$ git clone https://github.com/slimbootloader/slimbootloader.git  
Cloning into 'slimbootloader' ...  
remote: Enumerating objects: 30634, done.  
remote: Counting objects: 100% (348/348), done.  
remote: Compressing objects: 100% (213/213), done.  
remote: Total 30634 (delta 147), reused 284 (delta 128), pack-reused 30286  
Receiving objects: 100% (30634/30634), 20.67 MiB | 4.07 MiB/s, done.  
Resolving deltas: 100% (20810/20810), done.  
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader$
```

### *Downloading slim-bootloader source code through git*

b) SBL Keys Generation:

SBL keys in Slim Bootloader are used to sign and verify firmware images. This helps to ensure the integrity of the firmware and prevent unauthorized modifications. So, generating SBL keys is a prerequisite before the SBL build

Steps to be followed to generate SBLKeys:

1. Set environment variable for SBL Key directory using the following command:

```
$ export SBL_KEY_DIR=<path to SblKeys directory>
```

Example:

```
$ export SBL_KEY_DIR="/home/ti/SblKeys"
```

You can check above export command is working properly or not using the echo command:

**\$ echo \$SBL\_KEY\_DIR**

```
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader/slimbootloader$ export SBL_KEY_DIR="/home/ti/SblKeys"  
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader/slimbootloader$ echo $SBL_KEY_DIR  
/home/ti/SblKeys  
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader/slimbootloader$
```

### **Set environment variable for SBL key**

2. Use the following command to generate keys:

**\$ python3 \$(SBL\_ROOT)\BootloaderCorePkg\Tools\GenerateKeys.py -k \$SBL\_KEY\_DIR**

Example:

**\$ python3 BootloaderCorePkg/Tools/GenerateKeys.py -k \$SBL\_KEY\_DIR**

### c) Build SBL

- SBL is built using the BuildLoader.py script and uses the following Python command to build:

**\$ python BuildLoader.py <subcommand> <target> <options>**

1. <subcommand> : build or clean

2. <target>: board name (e.g. apl or qemu)

Example of SBL command to build for Qemu emulator:

**\$ python BuildLoader.py build qemu**

👉 The outputs folder will be created if the build is successful. The outputs folder contains build binaries, Stitch\_Components.zip, etc.

```
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader/slimbootloader$ python3 BuildLoader.py build qemu  
Build [qemu] ...  
Checking Toolchain Versions ...  
- /usr/bin/python3: Version 3.8.10 ( ≥ 3.6.0) [PASS]  
- /usr/bin/openssl: Version 1.1.1f ( ≥ 1.1.0g) [PASS]  
- /usr/bin/nasm: Version 2.14.02 ( ≥ 2.12.02) [PASS]  
- /usr/bin/iasl: Version 20190509 ( ≥ 20160318) [PASS]  
- /usr/bin/git: Version 2.40.1 ( ≥ 2.20.0) [PASS]  
- /usr/bin/gcc: Version 9.4.0 ( ≥ 7.3) [PASS]  
... Done!  
  
make: Entering directory '/home/ti/bootloader/slimbootloader/BaseTools/'  
make -C Source/C  
make[1]: Entering directory '/home/ti/bootloader/slimbootloader/BaseTools/Source/C/'  
Attempting to detect HOST_ARCH from 'uname -m': x86_64  
Detected HOST_ARCH of X64 using uname.  
mkdir -p .  
mkdir ./libs  
make -C Common  
make[2]: Entering directory '/home/ti/bootloader/slimbootloader/BaseTools/Source/C/Common/'  
gcc -c -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../Include/X64/ -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation -Wno-restrict -Wno-unused-result -nostdlib -g -O2 BasePeCoff.c -o BasePeCoff.o  
gcc -c -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../Include/X64/ -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation -Wno-restrict -Wno-unused-result -nostdlib -g -O2 BinderFuncs.c -o BinderFuncs.o  
gcc -c -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../Include/X64/ -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation -Wno-restrict -Wno-unused-result -nostdlib -g -O2 CommonLib.c -o CommonLib.o  
gcc -c -I .. -I ../Include/Common -I ../Include/ -I ../Include/IndustryStandard -I ../Common/ -I .. -I . -I ../Include/X64/ -MD -fshort-wchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wno-deprecated-declarations -Wno-stringop-truncation -Wno-restrict -Wno-unused-result -nostdlib -g -O2 Crc32.c -o Crc32.o
```

### **Slimbootloader build command under execution**

```

+-----+
| SG1A | 0x3f0000(0xFFFF0000) | 0x010000 | Uncompressed, TS_A |
+-----+
|                TOP SWAP B                |
+-----+
| SG1A | 0x3e0000(0xFFFE0000) | 0x010000 | Uncompressed, TS_B |
+-----+
|                REDUNDANT A                |
+-----+
| KEYH | 0x3df000(0xFFFD0000) | 0x001000 | Uncompressed, R_A |
| CNFG | 0x3de000(0xFFFE0000) | 0x001000 | Uncompressed, R_A |
| FWUP | 0x3c6000(0xFFFC6000) | 0x018000 | Compressed, R_A |
| SG1B | 0x396000(0xFFF96000) | 0x030000 | Compressed, R_A |
| SG02 | 0x37c000(0xFFF7C000) | 0x01a000 | Compressed, R_A |
| EMTY | 0x360000(0xFFF60000) | 0x01c000 | Uncompressed, R_A |
+-----+
|                REDUNDANT B                |
+-----+
| KEYH | 0x35f000(0xFFF5F000) | 0x001000 | Uncompressed, R_B |
| CNFG | 0x35e000(0xFFF5E000) | 0x001000 | Uncompressed, R_B |
| FWUP | 0x346000(0xFFF46000) | 0x018000 | Compressed, R_B |
| SG1B | 0x316000(0xFFF16000) | 0x030000 | Compressed, R_B |
| SG02 | 0x2fc000(0xFFEFC000) | 0x01a000 | Compressed, R_B |
| EMTY | 0x2e0000(0xFFEE0000) | 0x01c000 | Uncompressed, R_B |
+-----+
|                NON REDUNDANT                |
+-----+
| PTES | 0x2df000(0xFFEDF000) | 0x001000 | Uncompressed, NR |
| IPFW | 0x2cf000(0xFFECF000) | 0x010000 | Uncompressed, NR |
| EPLD | 0x0c2000(0xFFC20000) | 0x20d000 | Uncompressed, NR |
| PYLD | 0x0a2000(0xFFCA2000) | 0x020000 | Compressed, NR |
| VARS | 0x0a0000(0xFFCA0000) | 0x002000 | Uncompressed, NR |
| EMTY | 0x001000(0xFFC01000) | 0x09f000 | Uncompressed, NR |
+-----+
|                NON VOLATILE                |
+-----+
| RSVD | 0x000000(0xFFC00000) | 0x001000 | Uncompressed, NV |
+-----+

```

Done [qemu] !  
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader/slimbootloader\$

**Showing a successful build of SBL for QEMU**

d) Build Outputs:

- i. If the build is successful, the Outputs folder will contain the build binaries. One of the output files will be Stitch\_Components.zip which will be used in the stitching step.
- ii. Boot to Yocto on QEMU Emulator:
- iii. Download the QEMU Yocto Image to the SBL top-level source directory from this link: [yocto.img](#)
- iv. Mount core-image-minimal-genericx86-64.hddimg locally and rename vmlinuz to bzImage:
- v. Linux Users:: Use the commands below:
  1. \$ sudo mkdir mnt
  2. \$ sudo mkdir mnt/yocto
  3. \$ sudo mount -o loop core-image-minimal-genericx86-64.hddimg mnt/yocto
  4. \$ sudo mv /mnt/yocto/vmlinuz mnt/yocto/bzImage
  5. \$ sudo umount mnt/yocto
- vi. Boot new Yocto image (with graphic console).
  1. \$ qemu-system-x86\_64 -machine q35 -m 256 -drive id=mydrive,if=none,-file=/home/ti/bootloader/core-image-minimal-genericx86-64.hddimg,format=raw -device ide-hd,drive=mydrive -serial mon:stdio -boot order=d -pflash Outputs/qemu/SlimBootloader.bin

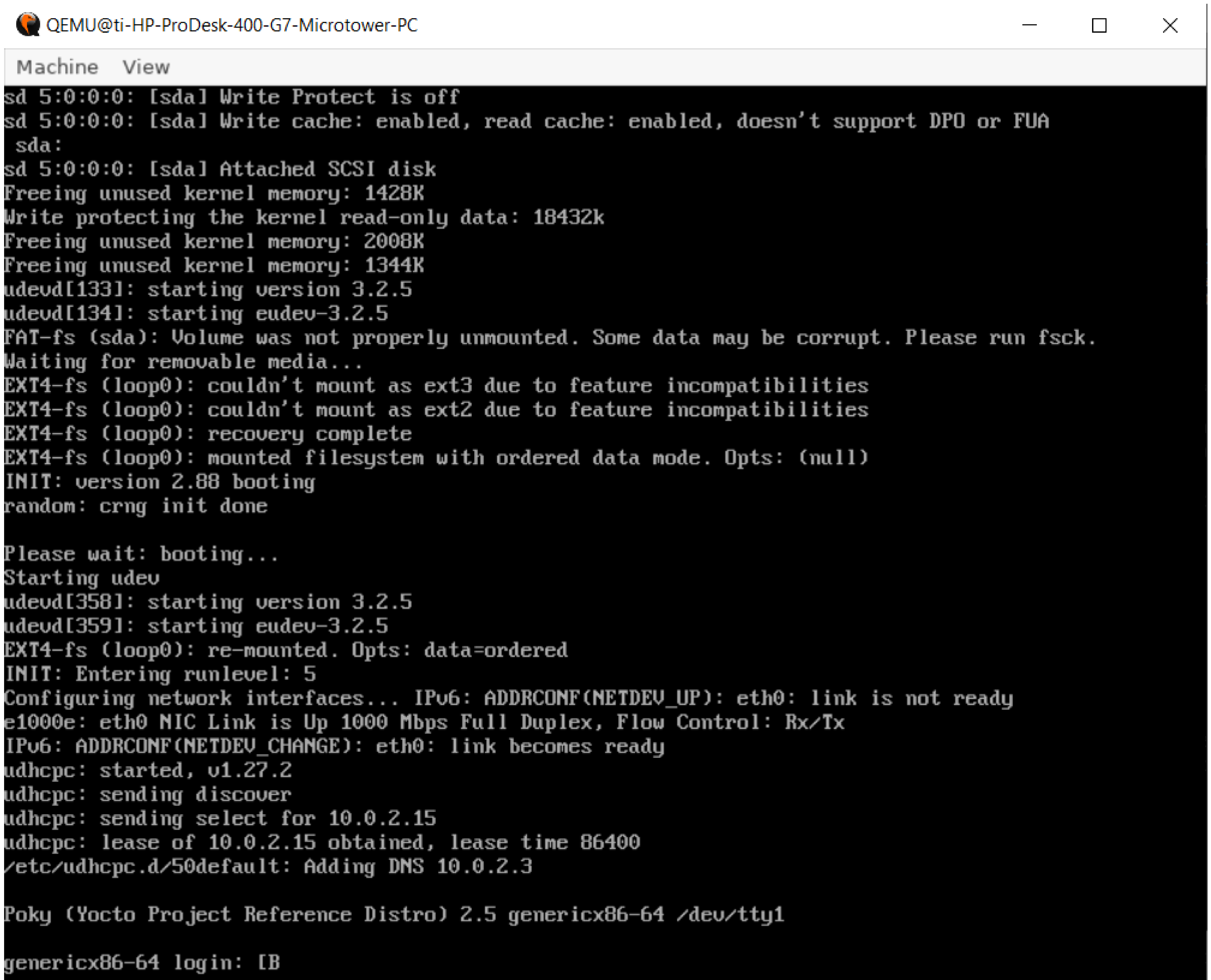
## 2. Output:

```
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/bootloader/slimbootloader$ qemu-system-x86_64 -machine q35 -m 256 -drive id=mydrive,if=none,file=/home/ti/bootloader/core-image-minimal-genericx86-64.hddimg,format=raw -device ide-hd,drive=mydrive -serial mon:stdio -boot order=d -pflash Outputs/qemu/SlimBootloader.bin
WARNING: Image format was not specified for 'Outputs/qemu/SlimBootloader.bin' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

===== Intel Slim Bootloader STAGE1A =====
SBID: SB_QEMU
ISVN: 001
IVER: 001.000.001.001.17782
SVER: D18BF478C0CB6936
FDBG: BLD(D IA32) FSP(R)
FSPV: ID($QEMFSP$) REV(00001000)
CPUV: ID(663) UCODE(0)
Loader global data @ 0x00001D10
Run STAGE1A @ 0x00070000
Load STAGE1B @ 0x00040000
HASH verification for usage (0x00000001) with Hash Alg (0x2): Success

===== Intel Slim Bootloader STAGE1B =====
Host Bridge Device ID:0x29C0
Board ID:0x1 - Loading QEMU!
QEMU Flash: Attempting flash detection at FFC00000
QemuFlashDetected => FD behaves as FLASH
QemuFlashDetected => Yes
SpiInstance = 0000E470
Variable region: 0xFFCA0000:0x2000
Loading Component KEYH: HS_
Registering container KEYH
HASH verification for usage (0x00000100) with Hash Alg (0x2): Success
SignType (0x2) SignSize (0x180) SignHashAlg (0x2)
RSA verification for usage (0x00000100): Success
HASH verification for usage (0x00000000) with Hash Alg (0x2): Success
Append public key hash into store: Success
Load EXT CFG Data @ 0x0000EB58:0x015C ... Success
HASH verification for usage (0x00000200) with Hash Alg (0x2): Success
SignType (0x2) SignSize (0x180) SignHashAlg (0x2)
```

### ***QEMU command to run Yocto image through slimbootloader under execution***



```
QEMU@ti-HP-ProDesk-400-G7-Microtower-PC
Machine View
sd 5:0:0:0: [sda] Write Protect is off
sd 5:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
sda:
sd 5:0:0:0: [sda] Attached SCSI disk
Freeing unused kernel memory: 1428K
Write protecting the kernel read-only data: 18432k
Freeing unused kernel memory: 2008K
Freeing unused kernel memory: 1344K
udevd[133]: starting version 3.2.5
udevd[134]: starting eudev-3.2.5
FAT-fs (sda): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
Waiting for removable media...
EXT4-fs (loop0): couldn't mount as ext3 due to feature incompatibilities
EXT4-fs (loop0): couldn't mount as ext2 due to feature incompatibilities
EXT4-fs (loop0): recovery complete
EXT4-fs (loop0): mounted filesystem with ordered data mode. Opts: (null)
INIT: version 2.88 booting
random: crng init done

Please wait: booting...
Starting udev
udevd[358]: starting version 3.2.5
udevd[359]: starting eudev-3.2.5
EXT4-fs (loop0): re-mounted. Opts: data=ordered
INIT: Entering runlevel: 5
Configuring network interfaces... IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
e1000e: eth0 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: Rx/Tx
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpd: started, v1.27.2
udhcpd: sending discover
udhcpd: sending select for 10.0.2.15
udhcpd: lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpd.d/50default: Adding DNS 10.0.2.3

Poky (Yocto Project Reference Distro) 2.5 genericx86-64 /dev/tty1
genericx86-64 login: [B
```

***After executing the QEMU command to run the Yocto image through***



```

Poky (Yocto Project Reference Distro) 2.5 genericx86-64 /dev/tty1

genericx86-64 login: root
root@genericx86-64:~# ps 1
ps: invalid option -- '1'
BusyBox v1.27.2 (2018-04-23 17:32:04 UTC) multi-call binary.

Usage: ps
root@genericx86-64:~# ls
root@genericx86-64:~# pwd
/home/root
root@genericx86-64:~# _

```

*Yocto image is successfully running.*

## 8. Boot Linux with U-Boot on Emulator

The following steps are required to integrate U-Boot as a payload with Slim Bootloader

### Build Instructions for U-Boot:

1. Build U-Boot and obtain u-boot-dtb.bin:

- a) \$ git clone <https://gitlab.denx.de/u-boot/u-boot.git> && cd u-boot
- b) \$ git checkout -b test c2addf9fc171de55d99fcccd7e5622894f74fe18
- c) \$ make distclean
- d) \$ make slimbootloader\_defconfig
- e) \$ make all

2. Prepare Slim Bootloader

a) Create payloadBins directory in payloadPkg:

```
$ mkdir -p <Slim Bootloader Dir>/PayloadPkg/PayloadBins/
```

b) Copy u-boot-dtb.bin to the PayloadBins directory

```
$ cp <U-Boot Dir>/u-boot-dtb.bin <Slim Bootloader Dir>/PayloadPkg/Payload-
Bins/u-boot-dtb.bin
```

3. Build Instruction for QEMU target

Slim Bootloader supports multiple payloads, and a board of Slim Bootloader detects its target payload by PayloadId inboard configuration. The payload can be any 4-byte value.

a) Update payload. Let's use 'U-BT' as an example.

```
$ vi Platform/QemuBoardPkg/CfgData/CfgDataExt_Brd1.dlt
```

Changes:

```
-GEN_CFG_DATA.PayloadId      | 'AUTO'
+GEN_CFG_DATA.PayloadId      | 'U-BT'
```

b) Update payload textbase. PAYLOAD\_EXE\_BASE must be the same as U-Boot

CONFIG\_SYS\_TEXT\_BASE in board/intel/slimbootloader/Kconfig. PAYLOAD\_LOAD\_HIGH must be 0:

```
$ vi Platform/QemuBoardPkg/BoardConfig.py
```

Changes:

+self.PAYLOAD\_LOAD\_HIGH = 0

+self.PAYLOAD\_EXE\_BASE = 0x00100000

c) Build QEMU target. Make sure u-boot-dtb.bin and U-BT PayloadId are in the build command.

The output is Outputs/qemu/SlimBootloader.bin:

**\$ python3 BuildLoader.py build qemu -p "OsLoader.efi:LLDR:Lz4;u-boot-dtb.bin:U-BT:Lzma"**

```
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/Slim_Bootloader/slimbootloader$ python3 BuildLoader.py build qemu -p "OsLoader.efi:LLDR:Lz4;u-boot-dtb.bin:U-BT:Lzma"
Build [qemu] ...
Checking Toolchain Versions ...
- /usr/bin/python3: Version 3.8.10 (≥ 3.6.0) [PASS]
- /usr/bin/openssl: Version 1.1.1f (≥ 1.1.0g) [PASS]
- /usr/bin/nasm: Version 2.14.02 (≥ 2.12.02) [PASS]
- /usr/bin/iasl: Version 20190509 (≥ 20160318) [PASS]
- /usr/bin/git: Version 2.40.1 (≥ 2.20.0) [PASS]
- /usr/bin/gcc: Version 9.4.0 (≥ 7.3) [PASS]
... Done!

SBL_KEY_DIR is set to /home/ti/SblKeys !!
Create FSP component file '/home/ti/Slim_Bootloader/slimbootloader/Build/BootloaderCorePkg/DEBUG_GCC5/FV/FSP_S.bin'
Create FSP component file '/home/ti/Slim_Bootloader/slimbootloader/Build/BootloaderCorePkg/DEBUG_GCC5/FV/FSP_M.bin'
Create FSP component file '/home/ti/Slim_Bootloader/slimbootloader/Build/BootloaderCorePkg/DEBUG_GCC5/FV/FSP_T.bin'
Key used for Signing /home/ti/SblKeys/MasterTestKey_Priv_RSA3072.pem !!
Container 'KEYH' was created successfully at:
/home/ti/Slim_Bootloader/slimbootloader/Build/BootloaderCorePkg/DEBUG_GCC5/FV/KEYHASH.bin
Rebase FSP-T from 0xFFFFF000 to 0xFFFF0000:
Patched 27 entries in 1 TE/PE32 images.
Patched 1 entries using FSP patch table.
Rebase FSP-M from 0xFFFD0000 to 0x0004E000:
Patched 278 entries in 4 TE/PE32 images.
Patched 1 entries using FSP patch table.
Rebase FSP-S from 0xFFC80000 to 0x0104B000:
Patched 49 entries in 2 TE/PE32 images.
Patched 1 entries using FSP patch table.
Create FSP component file '/home/ti/Slim_Bootloader/slimbootloader/Build/BootloaderCorePkg/DEBUG_GCC5/FV/FSP_S.bin'
Create FSP component file '/home/ti/Slim_Bootloader/slimbootloader/Build/BootloaderCorePkg/DEBUG_GCC5/FV/FSP_M.bin'
Create FSP component file '/home/ti/Slim_Bootloader/slimbootloader/Build/BootloaderCorePkg/DEBUG_GCC5/FV/FSP_T.bin'
1 config binary files were merged successfully!
4 config binary files were merged successfully!
Key used for Signing /home/ti/SblKeys/ConfigTestKey_Priv_RSA3072.pem !!
Config file was signed successfully!
ASM Bin/ResetVector.ia32.raw
FIXUP Bin/ResetVector.ia32.raw
Build environment: Linux-5.15.0-69-generic-x86_64-with-glibc2.29
Build start time: 00:18:07, May.14 2023
```

### Slimbootloader build command under execution using u-boot payload

SG1A	0x3f0000(0xFFFF0000)	0x010000	Uncompressed, TS_A
TOP SWAP B			
SG1A	0x3e0000(0xFFFE0000)	0x010000	Uncompressed, TS_B
REDUNDANT A			
KEYH	0x3df000(0xFFFD0000)	0x001000	Uncompressed, R_A
CNFG	0x3de000(0xFFFD0000)	0x001000	Uncompressed, R_A
FWUP	0x3c6000(0xFFC60000)	0x018000	Compressed, R_A
SG1B	0x396000(0xFF960000)	0x030000	Compressed, R_A
SG02	0x37c000(0xFF7C0000)	0x01a000	Compressed, R_A
EMTY	0x360000(0xFF600000)	0x01c000	Uncompressed, R_A
REDUNDANT B			
KEYH	0x35f000(0xFF5F0000)	0x001000	Uncompressed, R_B
CNFG	0x35e000(0xFF5E0000)	0x001000	Uncompressed, R_B
FWUP	0x346000(0xFF460000)	0x018000	Compressed, R_B
SG1B	0x316000(0xFF160000)	0x030000	Compressed, R_B
SG02	0x2fc000(0xFFEFC000)	0x01a000	Compressed, R_B
EMTY	0x2e0000(0xFFE00000)	0x01c000	Uncompressed, R_B
NON REDUNDANT			
PTES	0x2df000(0xFFED0000)	0x001000	Uncompressed, NR
IPFW	0x2cf000(0xFFEFC000)	0x010000	Uncompressed, NR
EPLD	0x0c2000(0xFFC20000)	0x20d000	Uncompressed, NR
PYLD	0x0a2000(0xFFCA2000)	0x020000	Compressed, NR
VARS	0x0a0000(0xFFCA0000)	0x002000	Uncompressed, NR
EMTY	0x001000(0xFFC01000)	0x09f000	Uncompressed, NR
NON VOLATILE			
RSVD	0x000000(0xFFC00000)	0x001000	Uncompressed, NV

Done [qemu] !  
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/Slim\_Bootloader/slimbootloader\$

**Showing a successful build of SBL for QEMU**

d) Launch Slim Bootloader on QEMU. You should reach at U-Boot serial console:

```
$ qemu-system-x86_64 -machine q35 -nographic -serial mon:stdio -pflash Outputs/qemu/SlimBootloader.bin
```

4. Test Linux booting on QEMU target:

Let's use LeafHill (APL) yocto image for testing:

a) Prepare yocto image hard disk image:

1. \$ wget [http://downloads.yoctoproject.org/releases/yocto-to/yocto-2.0/machines/leafhill/leafhill-4.0-jethro-2.0.tar.bz2](http://downloads.yoctoproject.org/releases/yocto/yocto-2.0/machines/leafhill/leafhill-4.0-jethro-2.0.tar.bz2)
2. \$ tar -xvf leafhill-4.0-jethro-2.0.tar.bz2
3. \$ ls -l leafhill-4.0-jethro-2.0/binary/core-image-sato-intel-corei7-64.hddimg

b) Launch Slim Bootloader on QEMU with disk image:

```
$ qemu-system-x86_64 -machine q35 -nographic -serial mon:stdio -pflash Outputs/qemu/SlimBootloader.bin -usb -device qemu-xhci,id=xhci,bus=pcie.0,addr=4 -device usb-storage,bus=xhci.0,drive=mydrive,serial=foo -drive id=mydrive,if=none,-file=/home/ti/Slim_Bootloader/leafhill-4.0-jethro-2.0/binary/core-image-sato-intel-corei7-64.hddimg,format=raw
```

```
ti@ti-HP-ProDesk-400-G7-Microtower-PC:~/Slim_Bootloader/slimbootloader$ qemu-system-x86_64 -machine q35 -nographic -serial mon:stdio -pflash Outputs/qemu/SlimBootloader.bin -usb -device qemu-xhci,id=xhci,bus=pcie.0,addr=4 -device usb-storage,bus=xhci.0,drive=mydrive,serial=foo -drive id=mydrive,if=none,file=/home/ti/Slim_Bootloader/leafhill-4.0-jethro-2.0/binary/core-image-sato-intel-corei7-64.hddimg,format=raw
WARNING: Image format was not specified for 'Outputs/qemu/SlimBootloader.bin' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

===== Intel Slim Bootloader STAGE1A =====
SBID: SB_QEMU
ISVN: 001
IVER: 001.000.001.001.17784
SVER: 96F72C39B86F9B92-dirty
FDBG: BLD(D IA32) FSP(R)
FSPV: ID($QEMFSP$) REV(00001000)
CPUV: ID(663) UCODE(0)
Loader global data @ 0x00001D10
Run STAGE1A @ 0x00070000
Load STAGE1B @ 0x00040000
HASH verification for usage (0x00000001) with Hash Alg (0x2): Success

===== Intel Slim Bootloader STAGE1B =====
Host Bridge Device ID:0x29C0
Board ID:0x1 - Loading QEMU!
QEMU Flash: Attempting flash detection at FFC00000
QemuFlashDetected => FD behaves as FLASH
QemuFlashDetected => Yes
SpiInstance = 0000E470
Variable region: 0xFFCA0000:0x2000
SPI WRITE: FFCA0010 00000004
SPI WRITE: FFCA0011 00000001
SPI WRITE: FFCA0014 00000008
SPI WRITE: FFCA001C 00000004
SPI WRITE: FFCA0011 00000001
Loading Component KEYH: HS_
Registering container KEYH
HASH verification for usage (0x00000100) with Hash Alg (0x2): Success
SignType (0x2) SignSize (0x180) SignHashAlg (0x2)
RSA verification for usage (0x00000100): Success
HASH verification for usage (0x00000000) with Hash Alg (0x2): Success
```

***QEMU command to run Yocto image through slimbootloader under execution***

c) Update boot environment values on the shell:

- => setenv bootfile vmlinuz
- => setenv bootdev usb
- => boot

```

Hit any key to stop autoboot: 0
=> setenv bootfile vmlinuz
=> setenv bootdev usb
=> boot
starting USB ...
Bus xhci_pci: Register 8001040 NbrPorts 8
Starting the controller
USB XHCI 1.00
Bus ehci_pci: USB EHCI 1.00
scanning bus xhci_pci for devices... 2 USB Device(s) found
scanning bus ehci_pci for devices... 1 USB Device(s) found
scanning usb for storage devices... 1 Storage Device(s) found
1: Hub, USB Revision 3.0
- U-Boot XHCI Host Controller
- Class: Hub
- PacketSize: 512 Configurations: 1
- Vendor: 0x0000 Product 0x0000 Version 1.0
Configuration: 1
- Interfaces: 1 Self Powered 0mA
Interface: 0
- Alternate Setting 0, Endpoints: 1
- Class Hub
- Endpoint 1 In Interrupt MaxPacket 8 Interval 255ms

2: Mass Storage, USB Revision 3.0
- QEMU QEMU USB HARDDRIVE foo
- Class: (from Interface) Mass Storage
- PacketSize: 512 Configurations: 1
- Vendor: 0x46f4 Product 0x0001 Version 0.0
Configuration: 1
- Interfaces: 1 Self Powered 0mA
- String: "Super speed config (usb 3.0)"
Interface: 0
- Alternate Setting 0, Endpoints: 2
- Class Mass Storage, Transp. SCSI, Bulk only

```

### ***Updating boot environment values on the shell***

```

intel-corei7-64 login: root
This image contains a time limited kernel and will reboot the machine
automatically in 10 days. Do not include this image in a product.

Use the image for evaluation purposes only.

Please see http://www.yoctoproject.org/tlk for instructions on how to
eliminate the timeout.
root@intel-corei7-64:~# ls
rajat test
root@intel-corei7-64:~# mkdir test1
root@intel-corei7-64:~# ls
rajat test test1
root@intel-corei7-64:~# █

```

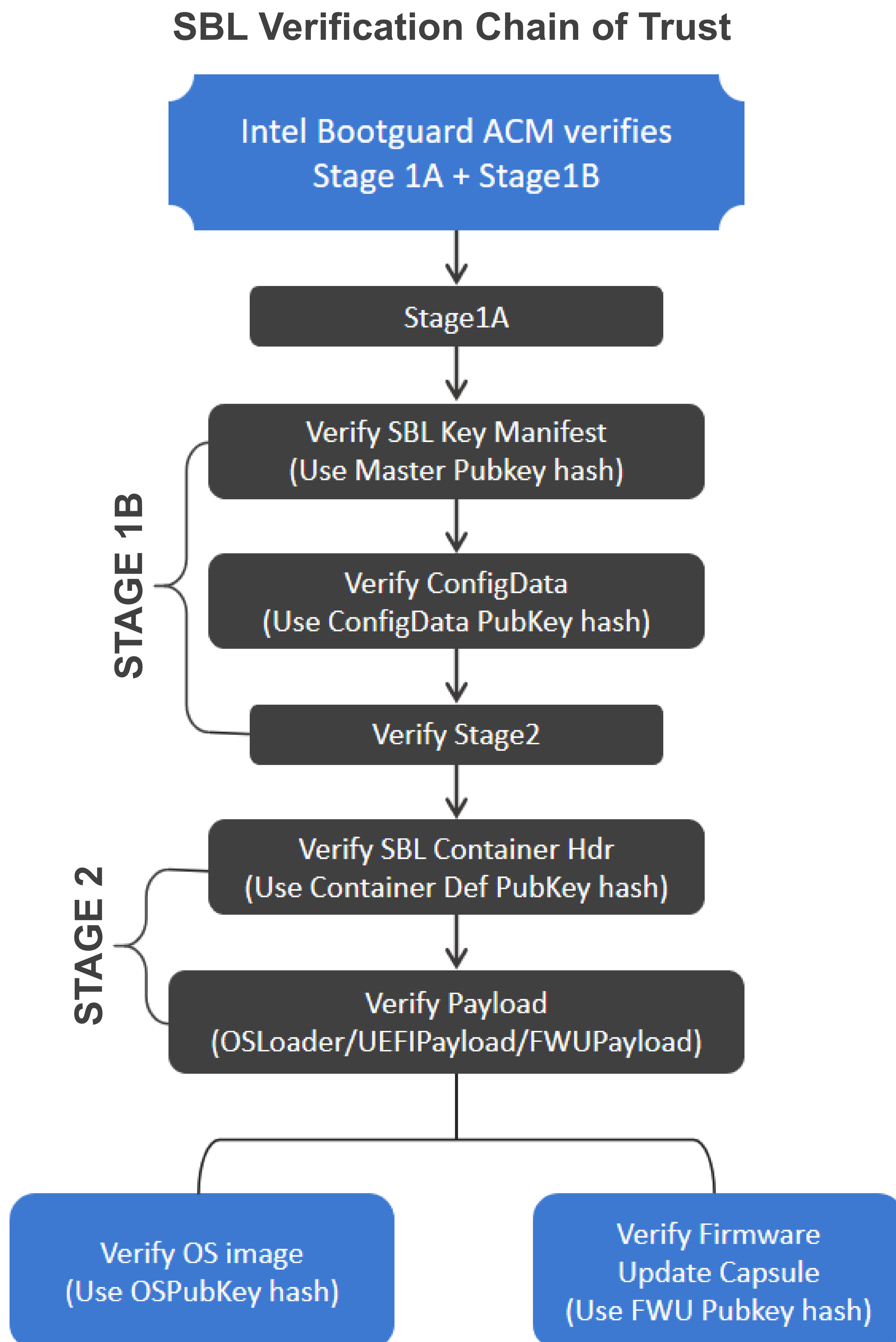
### ***Yocto image is successfully running***

## **9. Secure Boot**

- ✔ Secure Boot is a security feature that ensures the integrity and authenticity of the bootloader and software components during the boot process.
- ✔ It relies on digital signatures to verify the authenticity of the loaded components.
- ✔ Secure Boot establishes a chain of trust by verifying the signatures of each component against a set of trusted public keys
- ✔ Slim Bootloader supports verified boot as part of its secure boot feature. Verification uses either of the following two approaches.
  1. Hash verification
  2. Signature verification

✔ Verified Boot FLOW:

- 👉 The initial Root of Trust (RoT) provides the anchor of trust for the platform and is typically rooted in hardware.
- 👉 The chain of trust is maintained by cryptographically verifying each subsequent component before it is executed.
- 👉 If the verification of a component fails, the boot process will be halted.
- 👉 Verified boot ensures all executed code comes from a trusted source SBL supports verified boot.
- 👉 The below picture depicts how SBL maintains a chain of trust as the platform boots across various stages:



In fig. Verified boot flow

## 10. Troubleshoot

1. In uboot, after running the command:

```
make slimbootloader_defconfig
```

You could get the following error:

```
/bin/sh: 1: bison: not found make[1]: *** [scripts/Makefile.lib:222: scripts/kconfig/z-  
conf.tab.c] Error 127  
make: *** [Makefile:565: slimbootloader_defconfig] Error 2
```

Solution:

- 1) Manual install of bison using the following command:

```
sudo apt-get install bison
```

- 2) Also, install Felix using the following command:

```
sudo apt-get install flex
```

## 11. Conclusion

This document provides a detailed understanding of the steps involved in successfully implementing Slim Bootloader on embedded devices.

## 12. References

1. <https://slimbootloader.github.io/introduction/index.html>
2. <https://u-boot.readthedocs.io/en/latest/>

# Thank You!



## **Gurugram (Headquarter)**

806, 8th Floor  
BPTP Park Centra Sector-30,  
NH-8 Gurgaon - 122001  
Haryana (India)

[info@logic-fruit.com](mailto:info@logic-fruit.com)

+91-0124 4643950



## **Bengaluru (R&D House)**

Sy. No 118, 3rd Floor,  
Gayathri Lakefront,  
Outer Ring Road, Hebbal,  
Bangalore - 560 024

[sales@logic-fruit.com](mailto:sales@logic-fruit.com)

+91 80-69019700/01



## **United States (Sales Office)**

Logic Fruit Technologies  
INC 691 S Milpitas Blvd  
Ste 217 (Room 9) Milpitas  
CA 95035

[info@logic-fruit.com](mailto:info@logic-fruit.com)

+1-408 338 9743