Snag It, Flash It, Ship It : Rethinking Factory Flashing

Romain Gantois
Paresh Bhagat

bootlin

Texas Instruments

Introduction to the Speakers

Romain Gantois

Software Engineer, Bootlin

Working as Software Engineer with experience in Linux (kernel, userspace), tools.

Paresh Bhagat

Software Engineer, TI

Working as Software Engineer with experience in Linux (kernel, userspace), hypervisor, tools.

bootlin

Texas Instruments

About us: TI Processors and Open source

THE LINUX FOUNDATION

Real-Time LINUX

LF ENERGY

yocto PROJECT

Decades of contribution and collaboration

Ingrained culture to give back to the community

android open source project

Upstream FIRST!

Focus on long term, sustainable and quality products

OSADL

Zephyr®

Upstream and opensource ecosystem in device architecture

KernelCI

OpenAMP

U-Boot

Open Source

Upstream FIRST mentality!

gstreamer

Table of Contents

- Factory Flashing Methods
- Problem Statement
- U-boot flash writer
- Flow
- Features
- Limitations
- Snagboot
- From Snagboot to Snagfactory
- Using Snagfactory

bootlin

Texas Instruments

Factory Flashing Methods

Before Soldering

Memory Chip is programmed while its standalone.

Requires Specialized hardware like a gang programmer.

Limited flexibility as post production updates are difficult.

Generally faster

After Soldering

Memory chip is programmed after it is soldered to board.

Utilizes standard on board connectors like USB, UART, JTAG.

More flexible as flashing can be done without removing the chip.

Limited by interface speed

Problem Statement

Time-Consuming Flashing

Slow, error-prone processes, delaying production timelines

GUI Centric tools

Tools designed primarily for GUI are not easily automatable.

No Scalability

Legacy tools struggle with high-volume production creating bottlenecks.

Closed Source Tools

Many vendor provided tools are proprietary, difficult to customize and integrate.

Poor Cross Platform

Many existing flashing tools are platform-dependent (e.g., Windows or Linux-only

Vendor Specific

Tightly coupled to a single vendor, creating compatibility issues and requiring different processes.

bootlin

Texas Instruments

U-boot flash writer

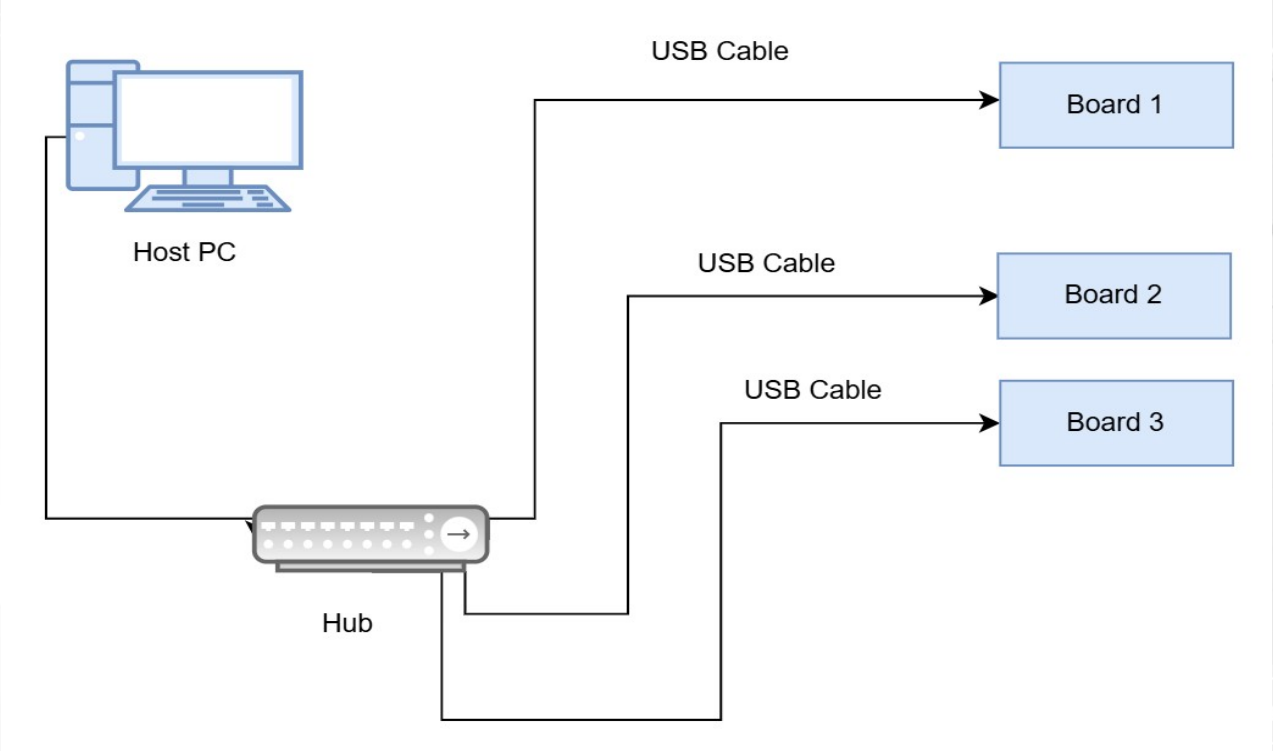
Uses dfu-util - an open source tool for host side implementation of the DFU

Written in python. Supports both Linux and Windows OS

Supports parallel flashing

Source - processor-sdk/uboot-flash-writer

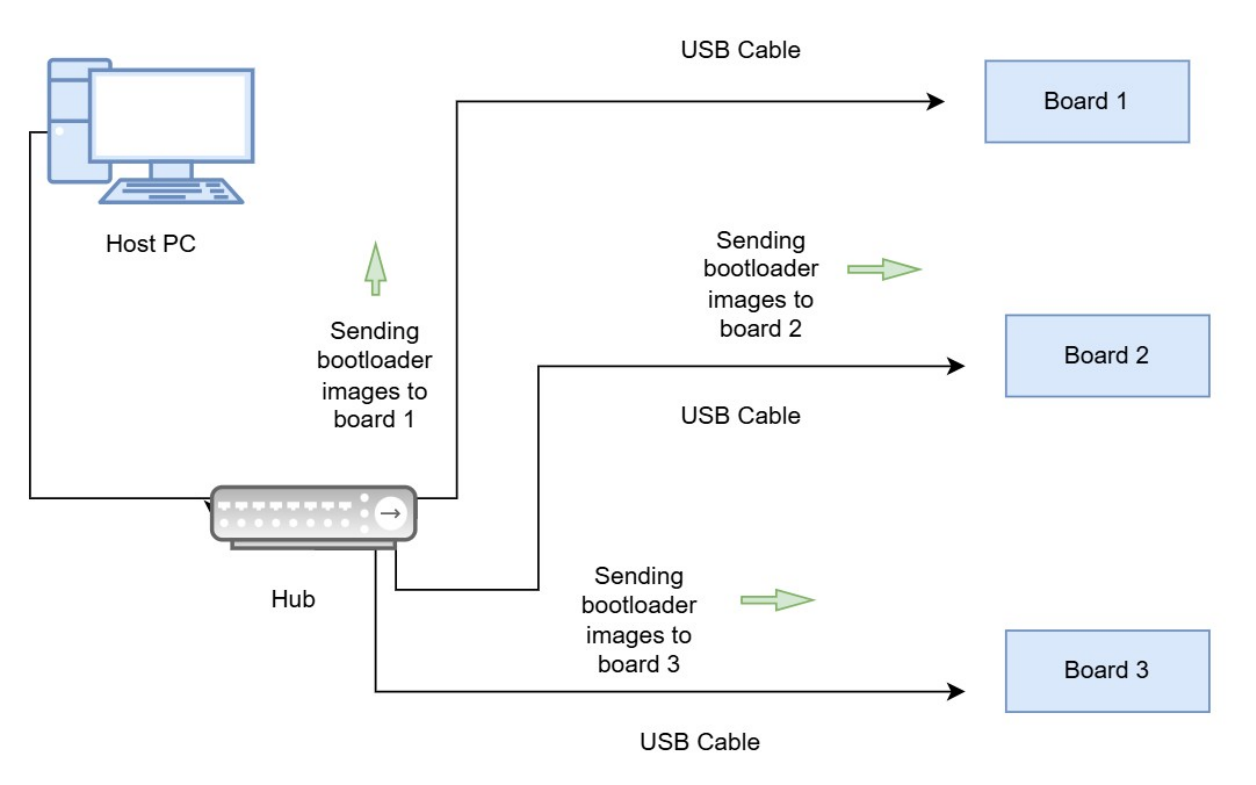
Flow



bootlin

Texas Instruments

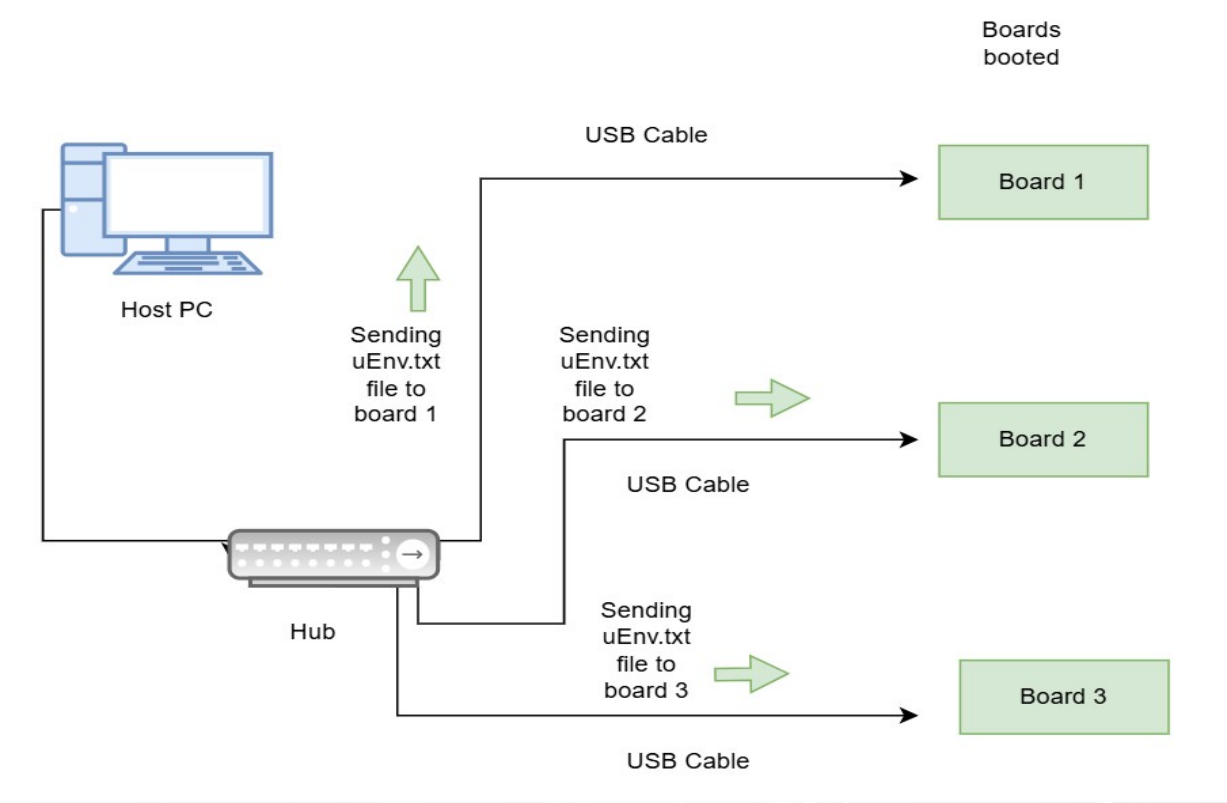
Flow (continued)



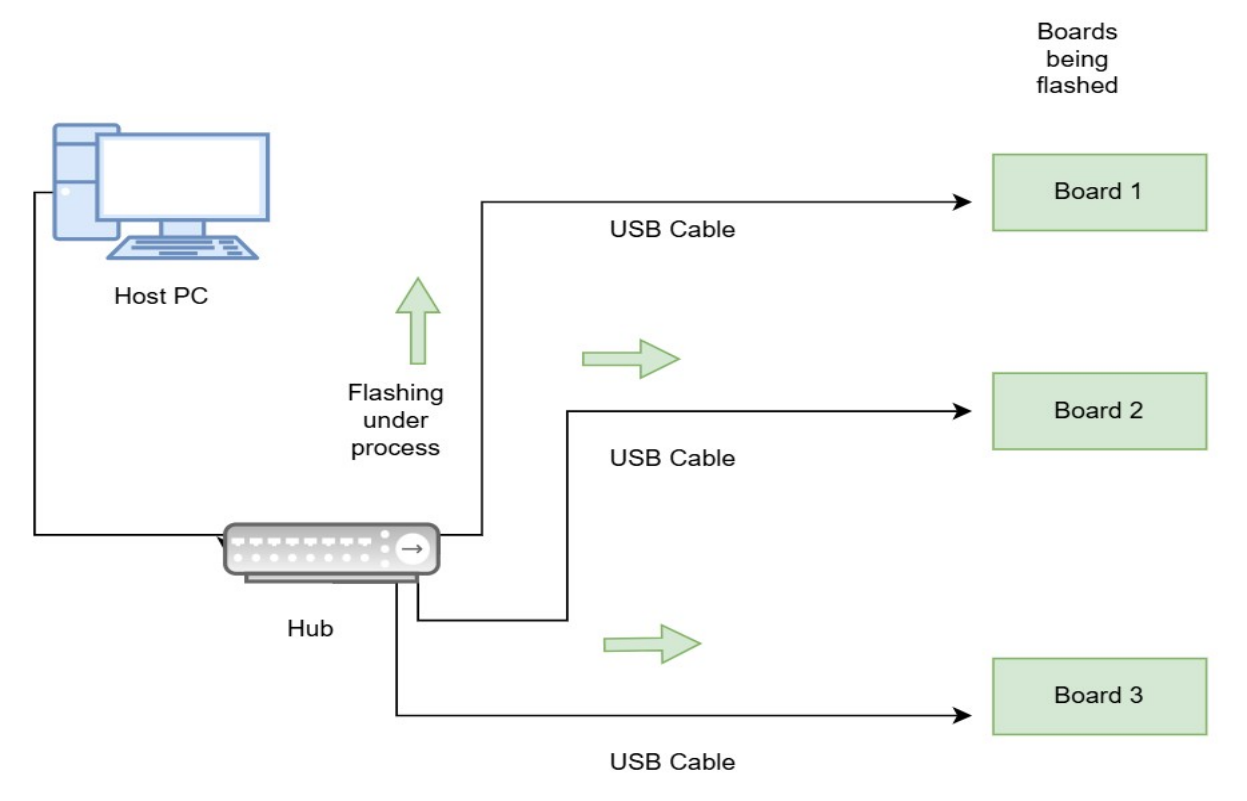
bootlin

Texas Instruments

Flow (continued)



Flow (continued)



bootlin

Texas Instruments

Features

- Factory Flashing
- Error Reporting
- Supports both Linux and Windows
- Multiple boot media

Limitations

Needs U-boot Support

No GUI

Speed

Snagboot

Snagboot in 2023: a CLI tool for Linux

Snagrecover

Targets USB-recovery mode

SoC-specific

Downloads and runs U-Boot on the target

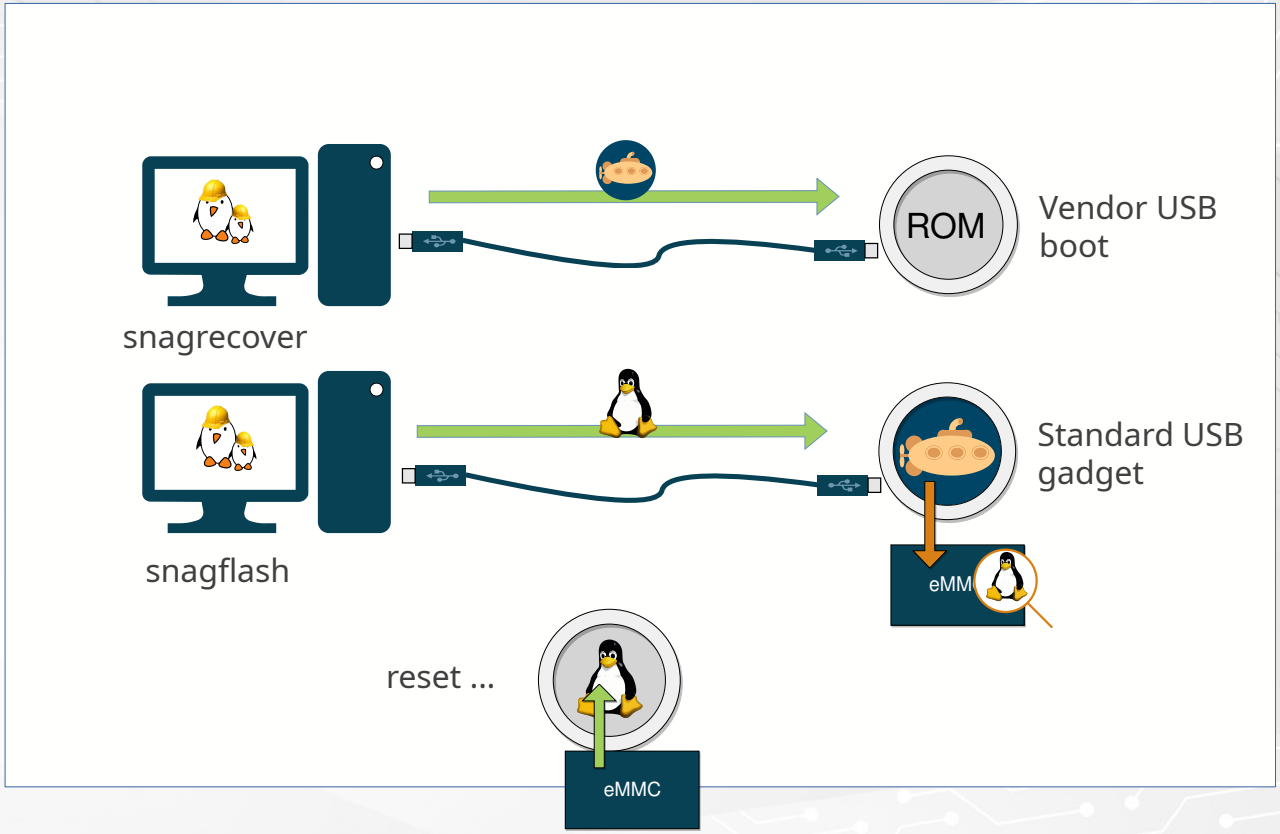
+

Snagflash

Targets a standardized USB gadget

Fastboot, DFU or UMS

Writes data to non-volatile storage



bootlin

Texas Instruments

Current support range

NXP i.MX	Microchip SAMA5	TI AM335, AM6x	ST STM32MP
Allwinner SUNXI	Intel KeemBay	Broadcom BCM2711 /2712	Xilinx ZynqMP

bootlin

Texas Instruments

Firmware Configuration Files

```
tiboot3:

 path: /path/to/tiboot3.bin

tispl:

 path: /path/to/tispl.bin

u-boot:

 path: /path/to/u-boot.img

                    example: AM62x
```

bootlin

Texas Instruments

Snagflash Protocols

DFU:

```
snagflash -P dfu -p 0483:df11 -D 0:binaries/u-boot.stm32
```

UMS:

```
snagflash -P ums -s binaries/u-boot.stm32 -b /dev/sdb1
```

Fastboot:

```
snagflash -P fastboot -p ... -f download:boot.img -f flash:0:1
```

Fastboot-Uboot:

```
snagflash -P fastboot-uboot -p 0483:0afb -I flash.cmd
```

https://github.com/bootlin/snagboot/blob/main/docs/snagflash.md

From Snagboot to Snagfactory

bootlin

Texas Instruments

Building upon Snagboot

Snagboot

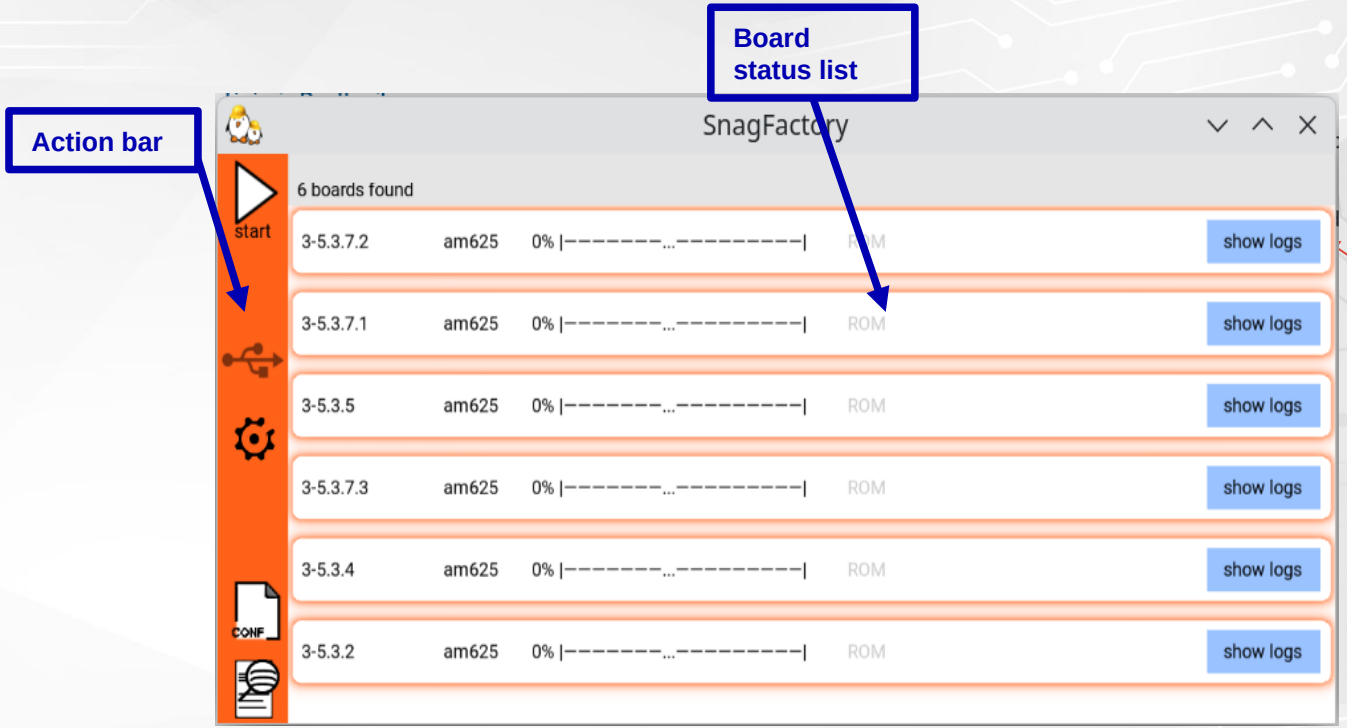
Recovery tool
Flashing tool
CLI and Linux-oriented

+ Windows 10/11 support

+ Graphical User Interface

+ Parallelization of tasks

User Interface



bootlin

Texas Instruments

Development milestones

v1.0

Initial Snagboot release

May 23, 2023

Supported 6 SoC families, DFU UMS and Fastboot

v2.0

Public Snagfactory release

Nov 20, 2024

Windows support, Snagfactory tool, AM6x support, Fastboot-Uboot protocol

v1.3

Last v1 release

Feb 24, 2024

Mostly bug fixes and small improvements

v2.4

Latest Snagboot release

Jul 21, 2025

ZynqMP, STM32MP2, Intel KeemBay support. Various bug fixes and improvements

Using Snagfactory

bootlin

Texas Instruments

Installation on Linux

Dependencies

python >= 3.9, pip, ensurepip, libusb

PyPi package

pip install snagboot[gui]

From source

./install.sh --with-gui

Installation on Windows

Dependencies

python >= 3.9, pip, ensurepip, libusb, zadig

PyPi package

pip install snagboot[gui]

From source

pip install .[gui]

Binary installer

snagboot_installer_win64.exe

USB access on Linux

USB recovery tools in general require read/write access to USB devices exposed by the target

Typical solution: per VID:PID udev rules

ROM-exposed VID:PID pairs are provided in snagboot:

snagrecover –udev

Other ones should be added on a per-device basis

USB access on Windows

On windows, unrecognized USB devices aren’t usually directly accessible

Typical solution: binding libusb-compatible drivers to specific VID:PID pairs

The open-source Zadig tool can be used for this

https://zadig.akeo.ie/

Documentation entry points

- Snagboot README
  - Installation instructions + basic usage
- Snagfactory introduction
  - Tour of the Snagfactory GUI and working principles
- Snagfactory configuration
  - Reference for writing YAML configuration files for Snagfactory

Configuration file

```
boards:
  "0451:6165": am625
  "03fd:0050": zynqmp
soc-models:
  am625-firmware:
  am625-tasks:
```

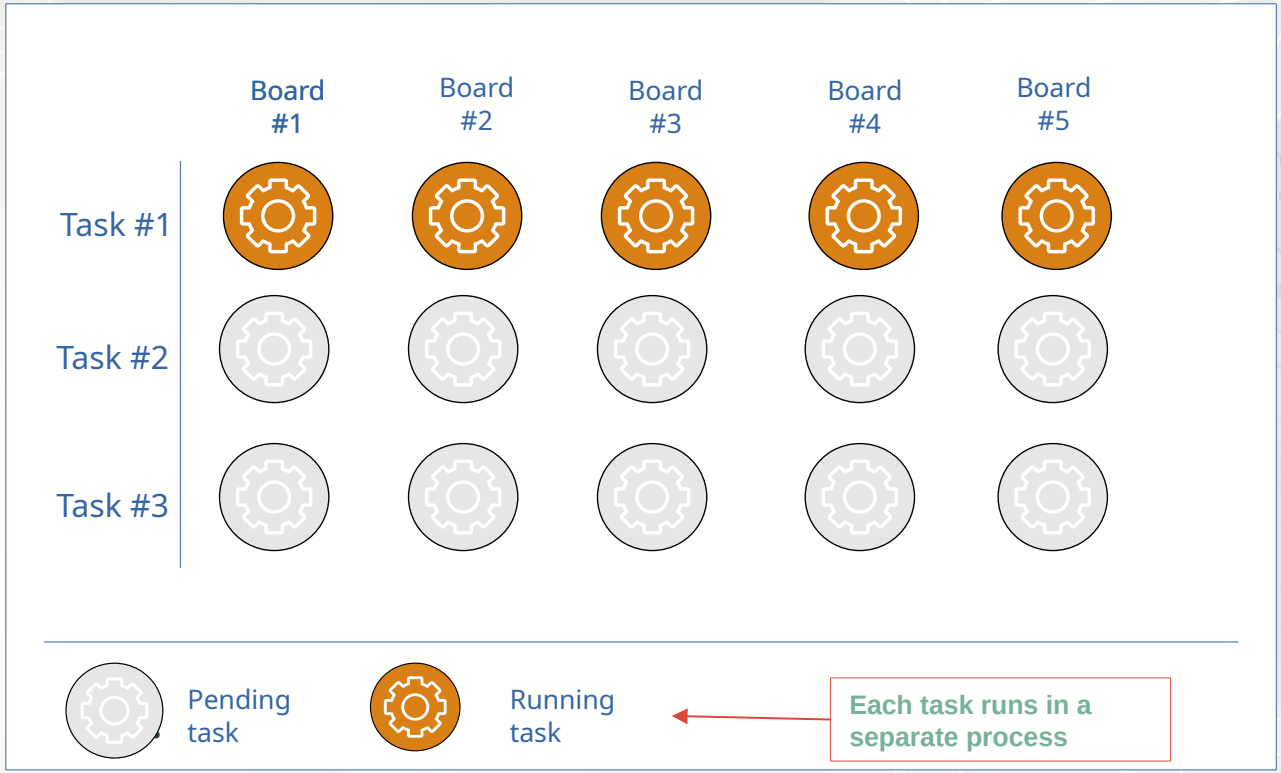
```
am625-firmware :
  tiboot3:
    path: ...
  tispl:
    path: ...
  u-boot:
    path: ...
```

```
am625-tasks :
  - target-device: mmc0
  - task: gpt
    args : …
  - task : flash
    args : ...
```

bootlin

Texas Instruments

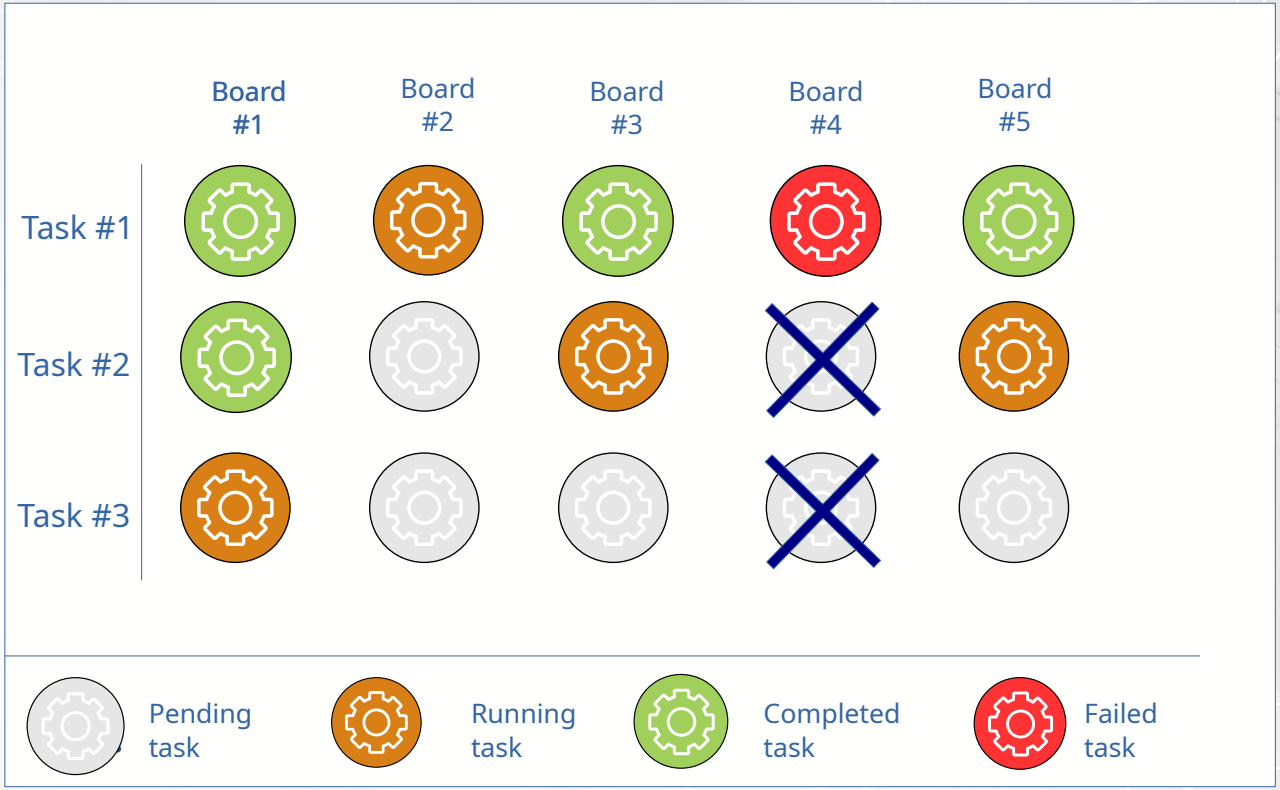
Task Pipelines



bootlin

Texas Instruments

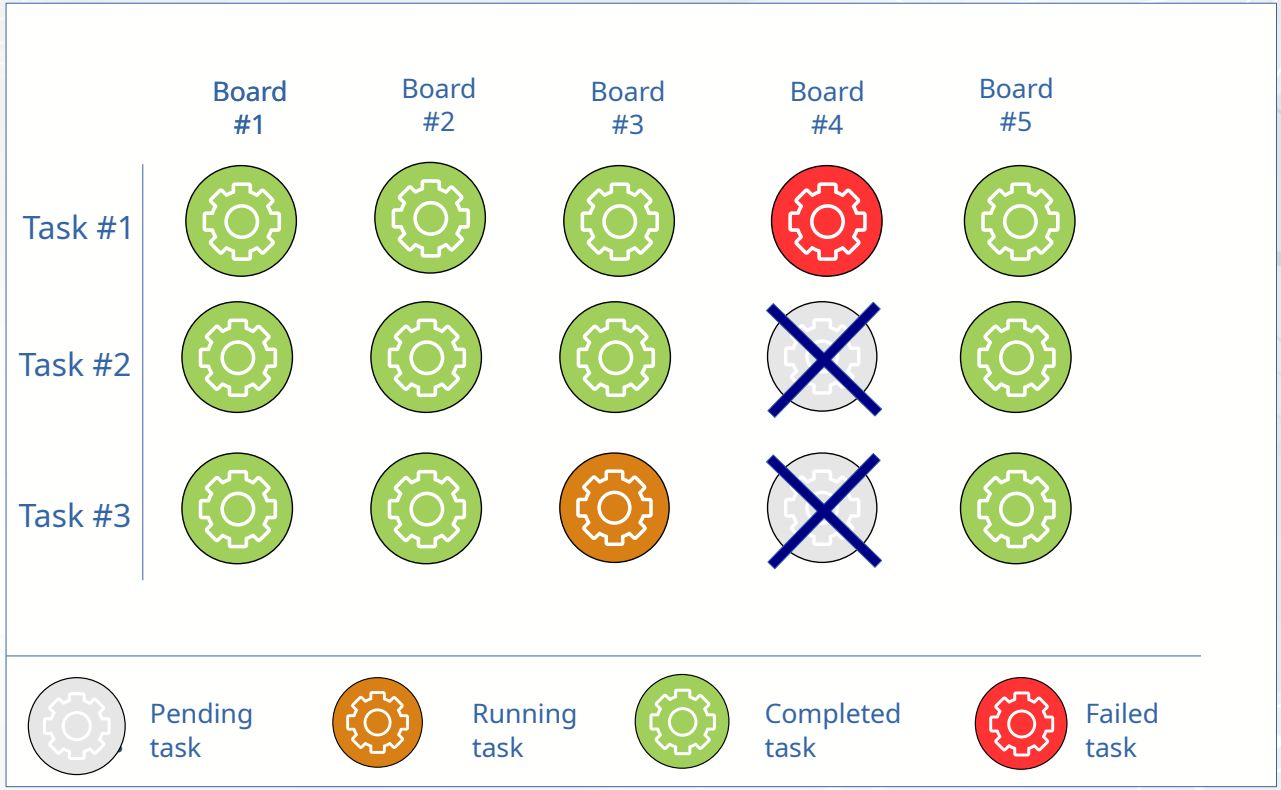
Task Pipelines



bootlin

Texas Instruments

Task Pipelines



bootlin

Texas Instruments

Tasks

Flash

part: entire device
OR GPT partition
OR MTD partition
OR eMMC hw partition
image: /path/to/file
image-offset: optional

Tasks

Flash

- Supports BMAP
- Supports larger-than-RAM files
- Requires U-Boot Fastboot buffer address

Tasks

GPT

- name: partition name
  size: partition size
  start: partition offset
  bootable: set GPT “bootable” flag
  uuid: GPT UUID
  type: GPT type UUID
  image: optional file to flash

Tasks

Run

- "snagflash fastboot command"

Main use case: "oem_run: <U-Boot shell command>"

Tasks

Define non-persistent MTD partitions

mtd-parts

reset

Send reset command and recover device

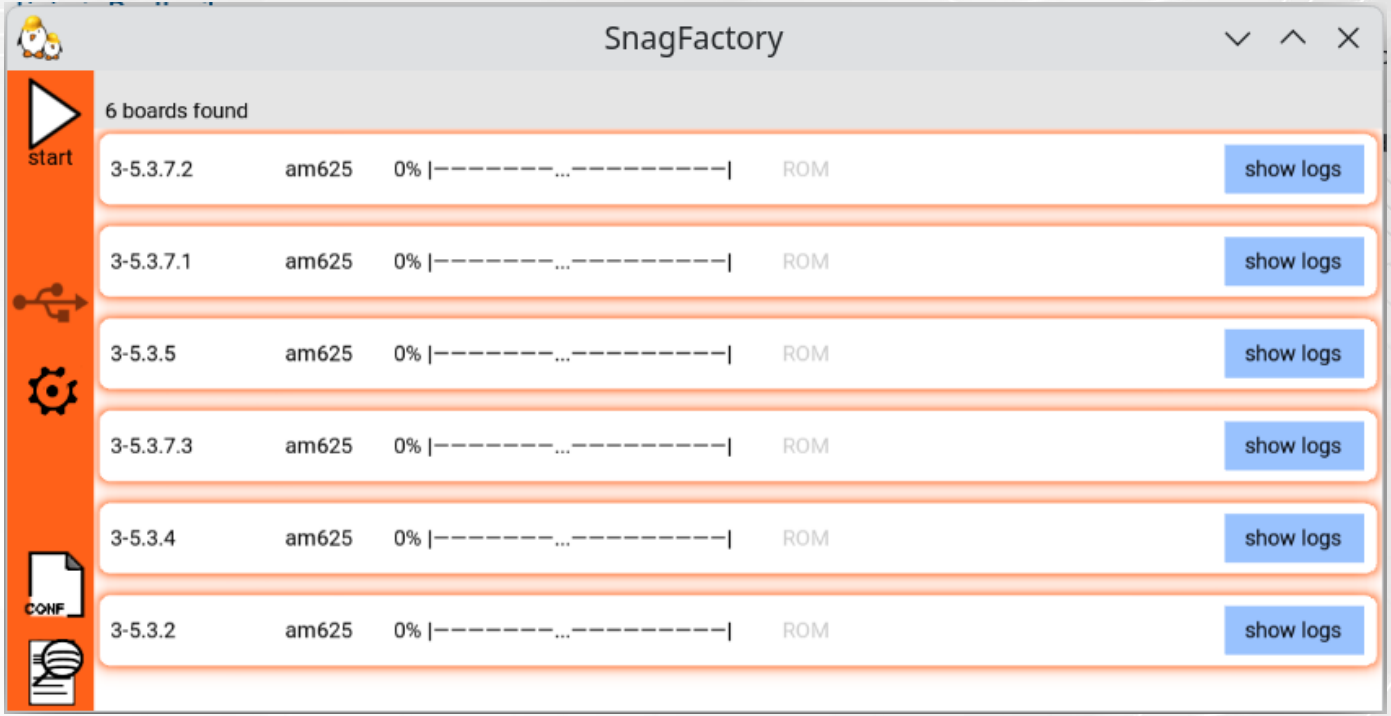
Pause task pipeline and request operator action

prompt-operator

emmc-hwpart

Permanently write eMMC hardware partition layout

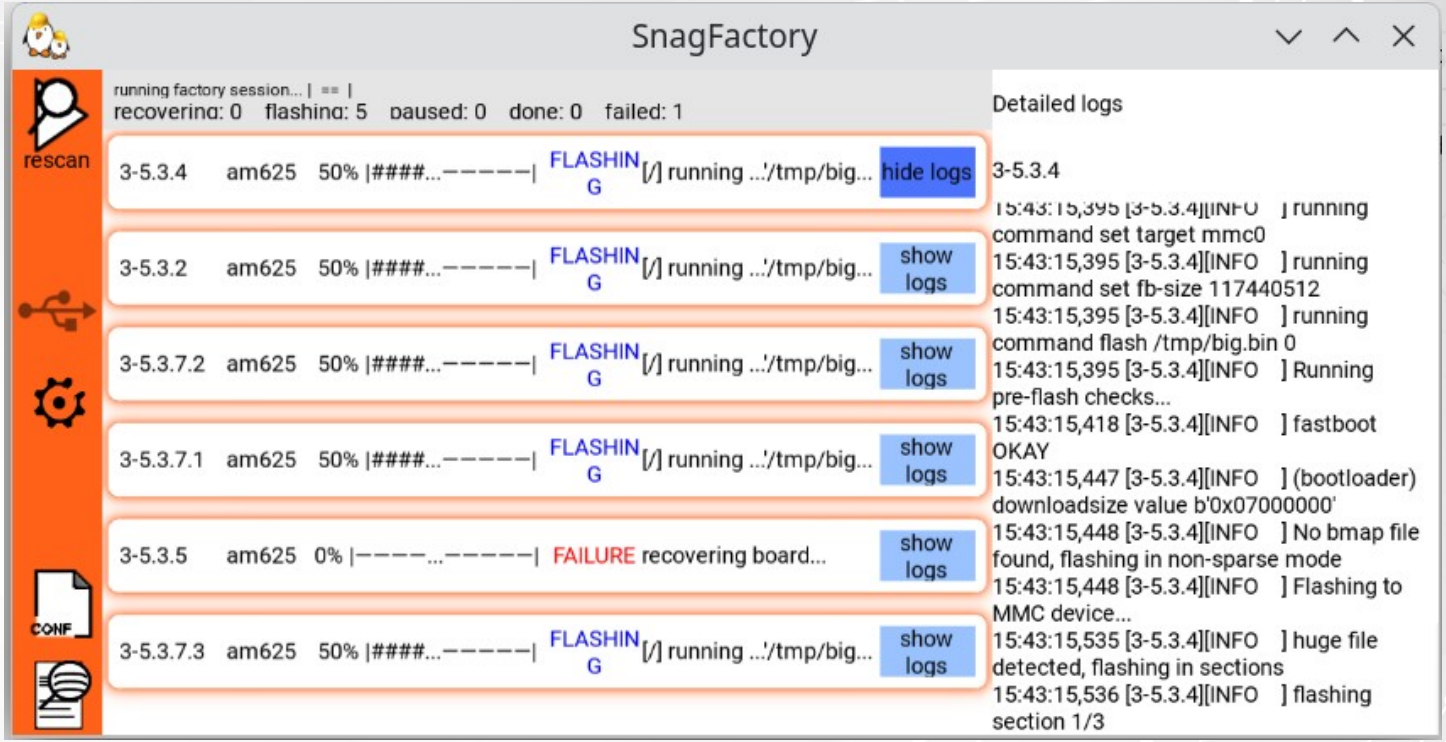
Scanning phase



bootlin

Texas Instruments

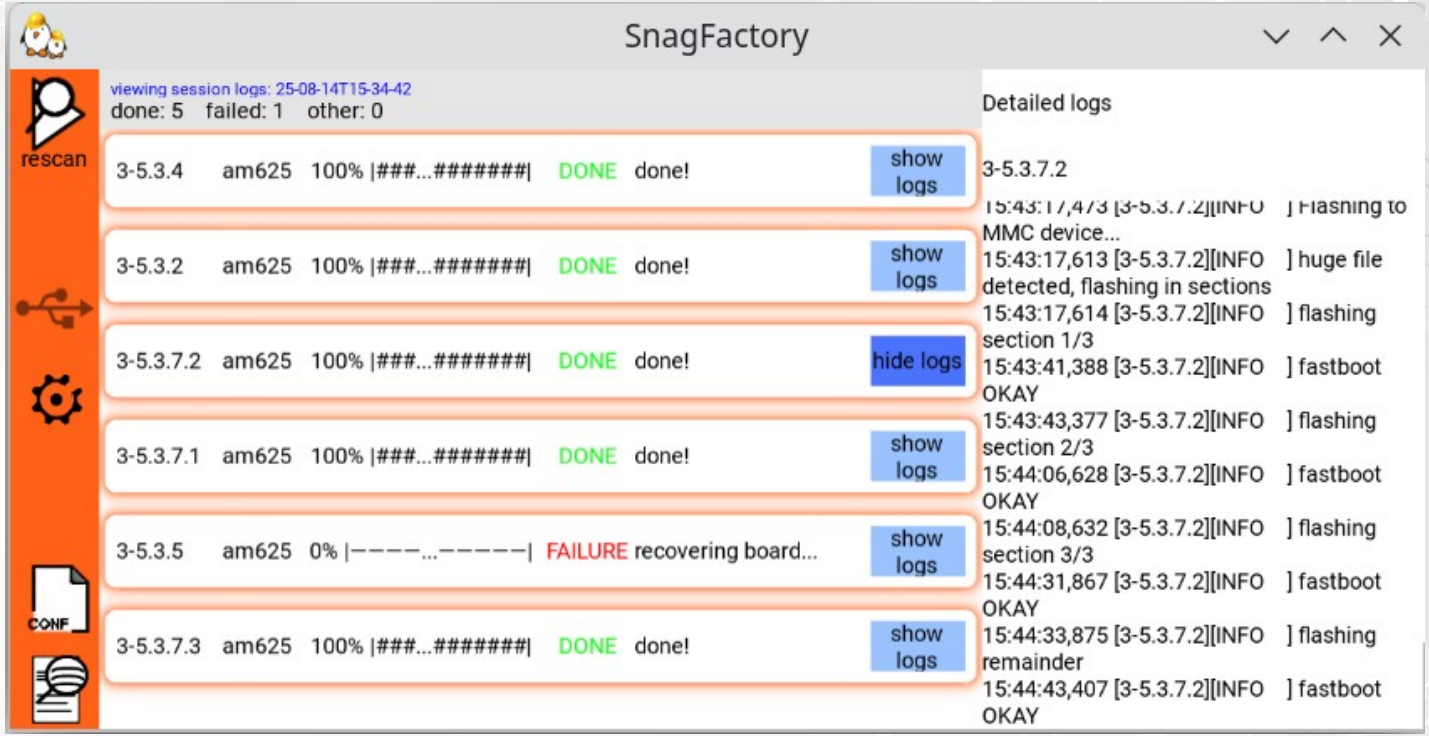
Factory flashing phase



bootlin

Texas Instruments

Log view phase



bootlin

Texas Instruments

Log file

```
<session timestamp>
summary: 5 done 1 failed
config: <config used>
results:
0451:6165 at 3-5.3.4: DONE
0451:6165 at 3-5.3.2: DONE
0451:6165 at 3-5.3.7.2: DONE
FACTORY LOG:
BOARD LOG 3-5.3.4:
BOARD LOG 3-5.3.2:
```

```
FACTORY LOG:
<ts> Start
<ts> 3-5.3.4 starting recovery task
<ts> 3-5.3.4 phase: BoardPhase.ROM -> BoardPhase.RECOVERING
<ts> 3-5.3.2 starting recovery task
```

```
BOARD LOG:
<ts> Installing firmware tiboot3
<ts> Searching for partition id…
<ts> Found DFU Functional descriptor: wTransferSize = 512
```

Development goals

- Expand Snagboot's support base
- Improve error reporting
- Improve documentation
- Get community feedback
- Introduce new factory flashing tasks

Thank you for listening!

bootlin

Texas Instruments

Q&A

- Contact Information:
  - Paresh Bhagat <p-bhagat@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.