Secure Boot ROM

UNDER CONSTRUCTION

- Overview
- Requirements
  - General
  - Update
  - Second Level Boot
  - Cryptographic Keys
- Architecture
  - Modules
    - Features
    - Platform Phase Management
    - Patch Initialization UNDER STUDY
    - Secure Boot ROM Management
    - Key Management
    - Second Level Boot Verification
    - Secure Protocol
    - Debug Authentication Interface Management UNDER STUDY
  - Module Structure
    - Public
    - Internal Procedures
- Secure Patch Management
- Parameters Mapping
- Life Cycle Management
  - Special Case - RMA Mode
- Security Keys
  - SiFive Test Key
  - SiFive Signing Key
  - Customer Signing Key
  - Customer Update Key
  - CSK Management
    - CSK Descriptor
    - Usecases
      - CSK Signature Check
      - CSK Slot Management

Overview

This section introduces concepts/features/behaviors of Secure Bootloader. The term "*Secure Bootloader*" covers the SBR located in ROM or in OTP, depending on platform hardware specifications.

CAUTION Note that "*non-Secure Bootloader*" shall be:

- either based on "*secure*" one with deactivated parts to keep coherency among platforms.
- either regular "*Secure Bootloader*" with test keys only.

DISCLAIMER

Current specifications of Secure Bootloader are for mono-core platforms. Multi-cores platforms induce more complex concerns/features that are not addressed here.

Requirements

This paragraph gathers all features/concepts that SBC shall cover/implement.

General

1. Purpose of SBR is to provide a trusted startup for platform.
   SBL-19 - [SBR][GENERAL] Trusted Startup Platform
   OPEN
2. When coming from POR, SBR must not be overcome.
   SBL-20 - [SBR][GENERAL] No overcome.
   OPEN
3. When resuming from low-power mode, SBR shall be overcome to speed up "*Resume Time*""
   ... **SBL-3** - [SBC] Power modes must be managed
   OPEN **TBD by customer???**
4. SBR must perform code check as soon as possible.
   a. if security check fails then SBR must stop code execution by going into "*shutdown*" mode.
   SBL-21 - [SBR][GENERAL] Code selfcheck
   OPEN
5. SBR shall run self-tests for cryptographic algorithms it's going to use.
   a. if security check fails then SBR must stop code execution by going into "*shutdown*" mode.
   SBL-22 - [SBR][GENERAL] Cryptographic library selfchecks
   OPEN
6. Any security related error leads SBR to "*shutdown mode*".
   SBL-23 - [SBR][GENERAL] Security error
   OPEN
7. '**MSEL**' pins shall indicate SBR what is expected boot mode.
   SBL-24 - [SBR][GENERAL] MSEL mode
   OPEN
   a. Either "*generic/normal mode*".
   b. Either "*RMA mode*".
   c. Either "*Watchdog mode*".
8. SBR restores platform to its previous state. It is to say SBR shall store contexts of all IPs it's going to use and restore them after use.
   SBL-25 - [SBR][GENERAL] Restore context
   OPEN
9. SBR shall be able to start SLB stored in external memory such as eMMC, flash SPI.
   SBL-26 - [SBR][GENERAL] SLB boot source
   OPEN
10. SBR shall have be able to use additional code$_{(drivers, algorithms, ...)}$ if trusted.

SBL-27 - [SBR][GENERAL] Additional code
OPEN

11. SBR shall have different states - Life Cycle Phase - which induce different behaviors.

SBL-28 - [SBR][GENERAL] Life Cycle feature
OPEN

    a. "*hardware test mode*".
    b. "*user mode*".
    c. EOL.
    d. "*unknown*".

12. **SBL-29** - [SBR][GENERAL] Watchdog process
OPEN Watchdog feature, if present, shall not interfere with SBR process ... **TBD**
13. CUK shall be programmed into platform to supplant SSK.

SBL-30 - [SBR][GENERAL] Customer Update Key
OPEN

Update

1. At SBR level, a secure update protocol$_{(SUP)}$ must be provided to program internal/external memories. **SBL-36** - [SBR][UPDATE] Secure update Protocol
OPEN
2. Secure update protocol may be triggered by external stimulus.

SBL-37 - [SBR][UPDATE] SUP stimulus
OPEN

3. If an error occurs during update protocol$_{(except\ packet\ signature\ check)}$, SBR shall go to SCL process. **SBL-38** - [SBR][UPDATE] SUP casual error behavior
OPEN
4. If packet signature check fails, then SBR must exit SBR management and shall go to SBR process. **SBL-39** - [SBR][UPDATE] SUP packet signature check failure
OPEN
5. After SUP communication session, SBR shall launch SFL.

SBL-40 - [SBR][UPDATE] SLB launch after update protocol
OPEN

6. Any non security related error$_{(except\ in\ SUP\ context)}$ leads to turn SBR into "*download mode*". This reset induces a "*download mode*" identical to previous one.
7. 'open source' community will push to have end-user ability to update/change/reflash secure boot keys - troy
    a. ship fully unlocked jtag enabled 'debug' chips? customer's responsibility
    b. customer's public key is loaded by SiFive, charge for loading customer's keys (value added service)
8. SUP shall support UART, USB, SPI or Secure JTAG.

SBL-41 - [SBR][UPDATE] SUP communication bus
OPEN

9. SUP communication bus choice is made through parameter stored in OTP.

SBL-42 - [SBR][UPDATE] SUP bus choice
OPEN

    a. UART is the default, mandatory interface.

10. Stimulus to activate SUP shall be settable in OTP.

SBL-43 - [SBR][UPDATE] SUP stimulus setting
OPEN

Second Level Boot

1. SBR is able to search for SLB/SFL's Secure Header directly from base address given in OTP. **SBL-31** - [SBR][SLB] SLB secure header
OPEN
2. SBR is able to parse and treat GPT structure in order to access SFL.

SBL-32 - [SBR][SLB] GPT management
OPEN

- Security Key Choice
- Secure Update Protocol
- Second Level Boot
- Usecases
    - Nominal Boot - no update protocol - Valid SLB
    - Nominal Boot - update protocol - Valid SLB
    - Nominal Boot - No update protocol - Invalid SLB
    - Nominal Boot - Update protocols errors - N/A SLB

    a. SBR shall introduce new GUID reference for its own use.
    b. strawman proposal for GUIDs https://github.com/tmagik/gptfdisk/commit/d831c2c88bedc11f85338d90d5a914eff8948c6a
3. Any software to be run after SBR must have its header$_{(application\ format)}$ checked successfully by this latter. **SBL-33** - [SBR][SLB] SLB format check OPEN
4. Any software to be run after SBR must have its digital signature checked successfully by this latter. **SBL-34** - [SBR][SLB] Digital signature check OPEN
5. SLB/SFL's header shall have a field '*version*' for application management.
   SBL-35 - [SBR][SLB] SLB version management
   OPEN

Cryptographic Keys

1. When platform is in "*hardware test mode*" only STK is used for communication.
   SBL-44 - [SBR][CRYPTO] Key for Hardware test mode
   OPEN
2. When platform is in "*user mode*" only SSK/CSK are used for communication.
   SBL-45 - [SBR][CRYPTO] Key for User mode
   OPEN
3. SBR shall be able to manage customer key$_{(CSK)}$ with defined depth in order to allow revoke/update of customer key.
   SBL-46 - [SBR][CRYPTO] Update CSK management.
   OPEN
4. CSK is granted on platform if, and only if its signature has been checked successfully with SSK hard coded in SBR. **SBL-47** - [SBR][CRYPTO] CSK acceptance OPEN
5. CSK is stored in non-volatile memory$_{(OTP)}$ large enough for CSK updating.
   SBL-48 - [SBR][CRYPTO] CSK storage
   OPEN
6. CSK must not be *stronger* than embedded SSK. It is to say, for instance, CSK can't be ECDSA secp384r1 if SSK is ECDSA secp384r1.
   SBL-49 - [SBR][CRYPTO] CSK strength constraints
   OPEN
7. CSK signature is checked before any use of key.
   SBL-50 - [SBR][CRYPTO] CSK check frequency
   OPEN
8. What ever signature check, process is done, at least twice, to harden its resistance to *glitch attack*. **SBL-51** - [SBR][CRYPTO] Signature check redundancy OPEN
9. If platform has no valid CSK at all, it shall process only with SSK/CUK.
   SBL-52 - [SBR][CRYPTO] CSK not present
   OPEN
10. SSK ans STK are owned by SiFive.
    SBL-53 - [SBR][CRYPTO] Hardcoded SiFive keys
    OPEN
11. For"*non-Secure Bootloader*", only test keys are set in platform.
    SBL-54 - [SBR][CRYPTO] Key(s) for non-secure platform
    OPEN
12. CUK is stored in internal non-volatile memory$_{(OTP)}$ but has no certificate. it is provided by customer to SiFive. CUK is programmed at part's customization step.
    SBL-55 - [SBR][CRYPTO] Customer Update Key storage
    OPEN

Architecture

Modules

In order to be flexible and configurable, bootloader shall be based on software modules. This means that features shall be isolated in dedicated modules.

Features

- Modules only deal with what is going to be used in SBR context. At first, it's not attended to provide upper layers software, any kind of public API.
- Each module shall have its "*stubbed*" version. This very specific version of the module implements the "*public layer*"(c.f. Module Structure) with predefined returned values/parameters/function outputs.
- A "*process*" shall gather all procedures mandatory to its operation, but exclusive to itself.
    - For instance, "*Secure Protocol*" gathers bus management, protocol management but it uses "Key Management" procedures.
- Software module shall be adaptable to platform specifications or stubbed if not relevant.

[...]

Hereafter, the features of *main()* are not exhaustive but represents the main concerns about SBR.

Platform Phase Management

PPM module - This software module deals with features of platform depending on its "*Life Cycle*".

Patch Initialization UNDER STUDY

PI module - This part checks and initializes patches stored in OTP. It readies internal RAM table which role is to make indirection for targeted function(s).

Secure Boot ROM Management

SBRM module - This module gathers functionalities that are "*transverse*" or "*non-specific*". It initializes SBR context, saves and restores platform context in order to restore this latter before jumping in SCL.

Key Management

KM module - This module manages the security keys used in generic SBL context. It interacts with cryptographic library and internal storage(OTP) mainly.

Second Level Boot Verification

SLBV module - Here is where SBR prepares the platform to exit from its context, check next "*application*" to launch(SLB/SFL) and if it has been verified successfully, it jumps into it.

Secure Protocol

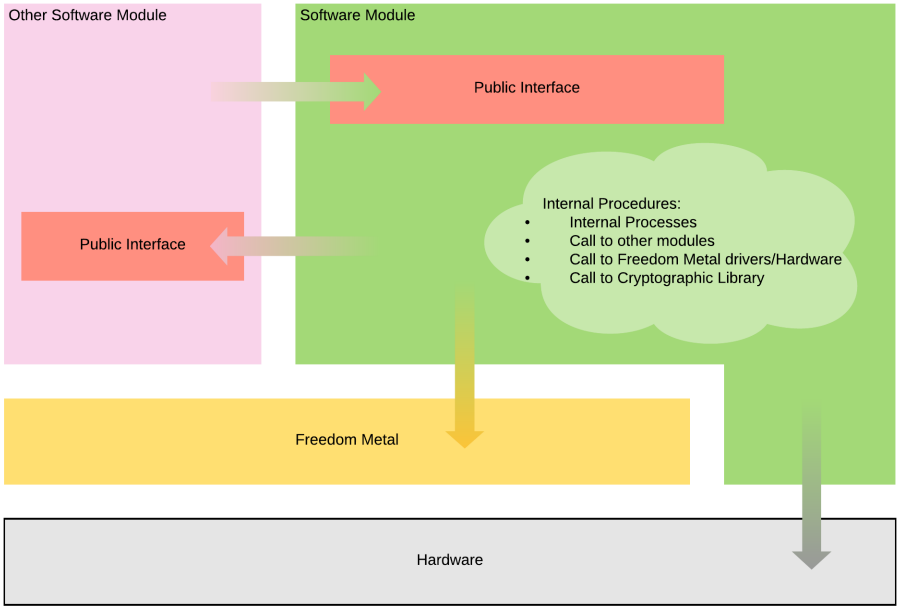
SP module - This module implements protocol(s) used in SBR context(for instance, SUP). A dedicated page for secure update protocol is located here.

This module is in charge of managing bus or communication interfaces such as *UART*, *SPI*, *USB*, *Ethernet*, ...

Debug Authentication Interface Management UNDER STUDY

DAIM module - This module manages the AM hardware device in order to allow Secure Debug on platform. For instance, it could re-enable debug when DM has been disabled.

Module Structure



Public

This layer gathers all API called by "*user*"(i.e. other SBR modules). Note that "*public*" layer must implement a list of generic services functions such as:

- init()/shutdown() functions.

Internal Procedures

In this layer is located all the source code that is not module's public API.

This layer is enriched with more specific piece of code that have not to be in "*public*":

- It gathers the calls to other SBR modules. Therefore you find here the calls to "*public*" functions of other modules.
- It implements protocol functions(SUP for instance), sub-routines for module's internal mechanisms.
- It calls Freedom Metal drivers and uses directly(if needed) hardware registers/devices.

Secure Patch Management

This feature is introduced to deport specific code outside ROM in order to be updated/fixed/changed/removed functions in platform's life.

Please find here its dedicated page.

Parameters Mapping

In this paragraph is put together a list of major SBR parameters. Those are shown in OTP mapping page.

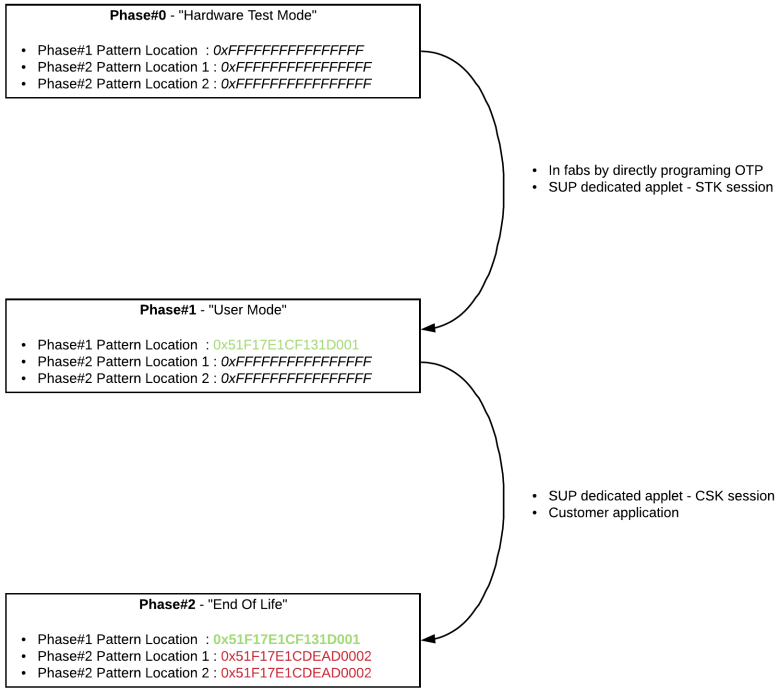
[...]

Life Cycle Management

Purpose of this feature is to constrain platform to several behavior depending its life cycle. Some functions are available in one life cycle. There 3 life cycle phases:

- **Phase#0** - In this phase, platform is not allowed to run second level bootloader. SBR considers platform in "*hardware test mode*"; as a consequence, only SUP is active with specific key(STK) and no SLB launch is allowed. No that "*RMA download mode*" is not active in this phase.
- **Phase#1** - In this phase, platform is allowed to run second level bootloader. SBR considers platform in "*user mode*". SUP is active with SSK and CSK, STK is no more accepted as valid key. "*RMA download mode*" is active. Pattern '**0x51f17e1cf131d001**' is set in "*Phase#1 Location*".
- **Phase#2** - **E**nd **O**f **L**ife. Note that platform is considered in **Phase#2** if, at least, one of the two "*Phase#2 Locations*" holds '0x51f17e1cDEAD0002' pattern. Normal behavior is to have this pattern in the two locations.
- **Phase#U** - When SBR cannot identify platform's phase, then it considers platform in this phase "Unknown". Here is only authorized SUP communication like in "*RMA download mode* ". Once in **Phase#U**, platform cannot move to another phase.

Platform's phase increment is monotonic; meaning that it can only move from **Phase#0** to **Phase#1**(for instance) and cannot go backward. Please refer to diagram below.
Here, pattern '*0xffffffffffffffff*' represents "*virgin value*", it is OTP dependent.



Special Case - RMA Mode

Only when platform is in **Phase#1**, "*RMA download mode*" can be activated with specific input at platform's boot time. When in **Phase#U**, this mode is normal behavior, nothing else is authorized. Compared to regular **Phase#1** behaviors, this mode differs in following points:

- Only SUP with field UID set, signed with CSK(SSK if no CSK programmed) is accepted by SBR.
- When/if exiting from SUP session, platform resets.
- No SLB launch of any kind.

Security Keys

SiFive Test Key

This key(ECDSA specp384r1 by default) is used only in **Phase#0**.

The public part of this key is hard coded in SBR. Private part is stored in SiFive's HSM.

There are two kinds of value depending on platform development step.

- Test version, known by "everyone" to ease development and tests

- Production version, specific to production.

SiFive Signing Key

This key(ECDSA secp384r1 by default) is used only in **Phase#1**.

The public part of this key is hard coded in SBR. Private part is stored in SiFive's HSM.

There are two kinds of value depending on platform development step.

- Test version, known by SiFive only to ease development and tests. It does not depend on platform.
- Production version, specific to production.

Customer Signing Key

This key(ECDSA specp384p1 by default) is used only in **Phase#1**.

The key is stored in OTP with it signature(from SSK) and a descriptor.

Private part of this customer key should be stored in customer's HSM.

There could have two kinds of value depending on platform development step.

- Test version, known by "*everyone*" to ease development and tests. It does not depend on platform.
- Production version, specific to production.

Customer Update Key

This key(ECDSA specp384p1 by default) is used only in **Phase#1**.

The key is stored in OTP without signature but with a descriptor.

Private part of this customer key should be stored in customer's HSM.

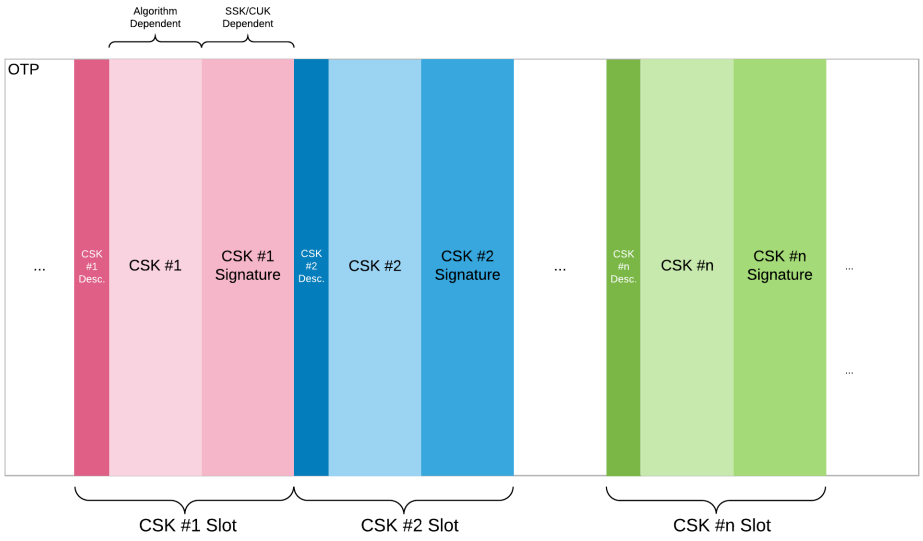
This key is completely managed by customer and shall be programmed during either part's customiza tion, either with SUP applet (signed with SSK/CSK)

CSK Management

CSK is customer dependent, even if SiFive uses a "*test version*" common to all platforms but unique for a specific cryptographic algorithm.

This key must be signed by platform's SSK and its signature is stored with it. Therefore key storage element location gathers key, its signature and a parameter describing key nature(ECDSA, RSA, ...).

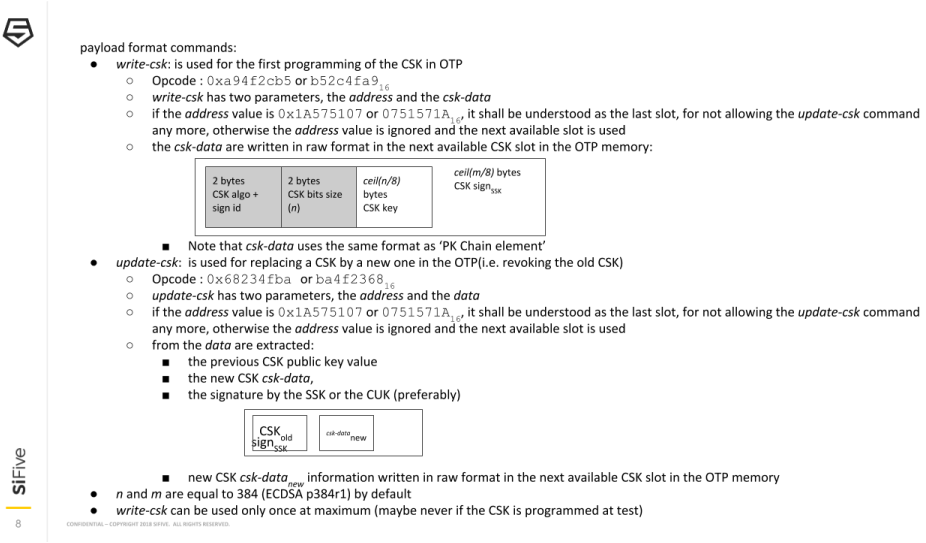
Hereafter a mapping of CSK storage location in OTP.



Please consider 'n' as an example; but platform shall have, at least two, locations to store CSK, meaning that after CSK has been programmed once, it can be updated. When CSK is updated(by programming a new one), the "*old*" is revoked.

CSK Descriptor

This descriptor is 32bits large for ECDSA secp384r1, as described below(slide 6):



payload format commands:

- *write-csk*: is used for the first programming of the CSK in OTP
  - Opcode : `0xa94f2cb5` or $\texttt{b52c4fa9}_{16}$
  - *write-csk* has two parameters, the *address* and the *csk-data*
  - if the *address* value is `0x1A575107` or $\texttt{0751571A}_{16}$, it shall be understood as the last slot, for not allowing the *update-csk* command any more, otherwise the *address* value is ignored and the next available slot is used
  - the *csk-data* are written in raw format in the next available CSK slot in the OTP memory:

2 bytes CSK algo + sign id	2 bytes CSK bits size (*n*)	*ceil(n/8)* bytes CSK key	*ceil(m/8)* bytes CSK sign_{ssk}

    - Note that *csk-data* uses the same format as 'PK Chain element'
- *update-csk*: is used for replacing a CSK by a new one in the OTP(i.e. revoking the old CSK)
  - Opcode : `0x68234fba` or $\texttt{ba4f2368}_{16}$
  - *update-csk* has two parameters, the *address* and the *data*
  - if the *address* value is `0x1A575107` or $\texttt{0751571A}_{16}$, it shall be understood as the last slot, for not allowing the *update-csk* command any more, otherwise the *address* value is ignored and the next available slot is used
  - from the *data* are extracted:
    - the previous CSK public key value
    - the new CSK *csk-data*,
    - the signature by the SSK or the CUK (preferably)

$\text{CSK sign}_{ssk\ old}$	csk-data_{new}

    - new CSK csk-data_{new} information written in raw format in the next available CSK slot in the OTP memory
- *n* and *m* are equal to 384 (ECDSA p384r1) by default
- *write-csk* can be used only once at maximum (maybe never if the CSK is programmed at test)

siFive

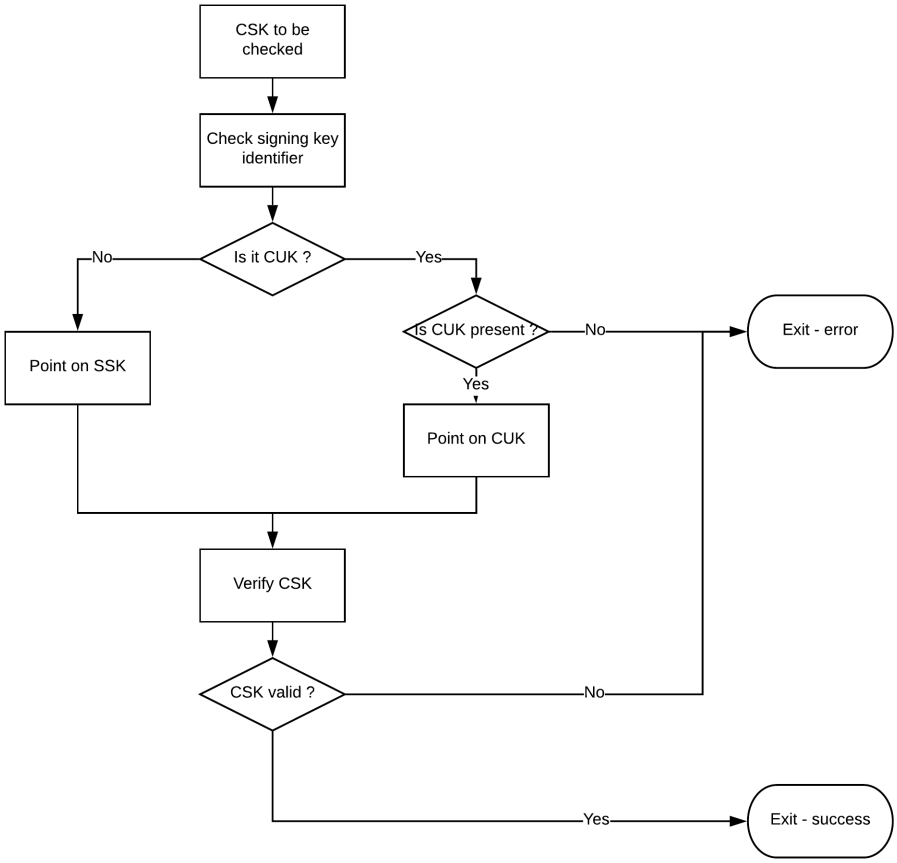
8

CONFIDENTIAL – COPYRIGHT 2018 SIFIVE. ALL RIGHTS RESERVED.

Usecases

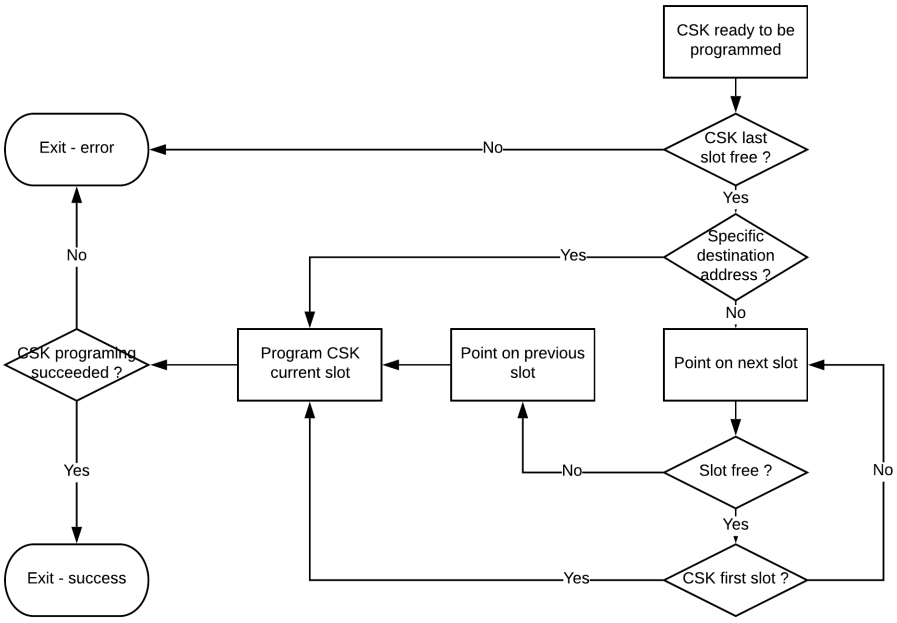
CSK SIGNATURE CHECK

Hereafter is shown procedure to verify CSK signature depending on which key to be used.



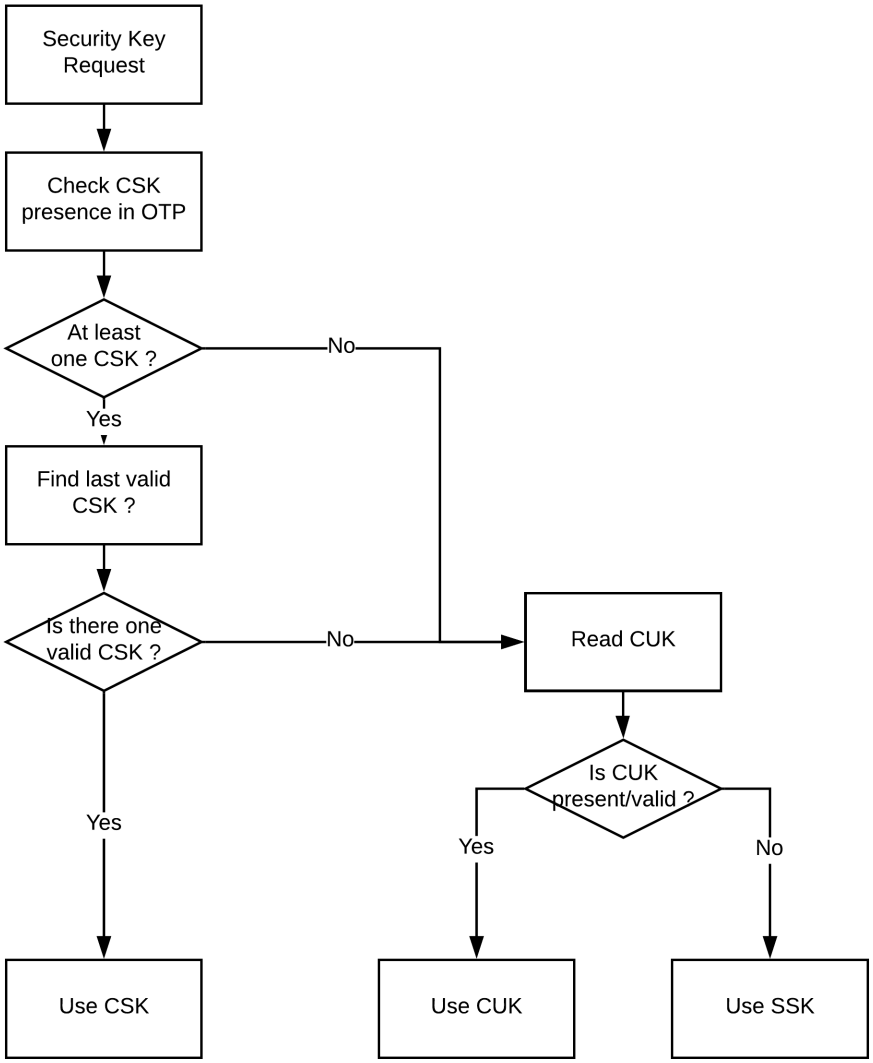
CSK SLOT MANAGEMENT

Hereafter is shown how is chosen the slot where to program CSK. Note that "*Program CSK*" success induces platform to reset. If procedure isn't successful, then error is returned to Host via SUP.



SECURITY KEY CHOICE

Whether it is for SUP or for SFL check and launch, signature has to be checked with security key. SBR has to choose the key to use depending on availability of it.



Secure Update Protocol

This protocol provides update software capacity to secure bootloader. It has a dedicated page here.

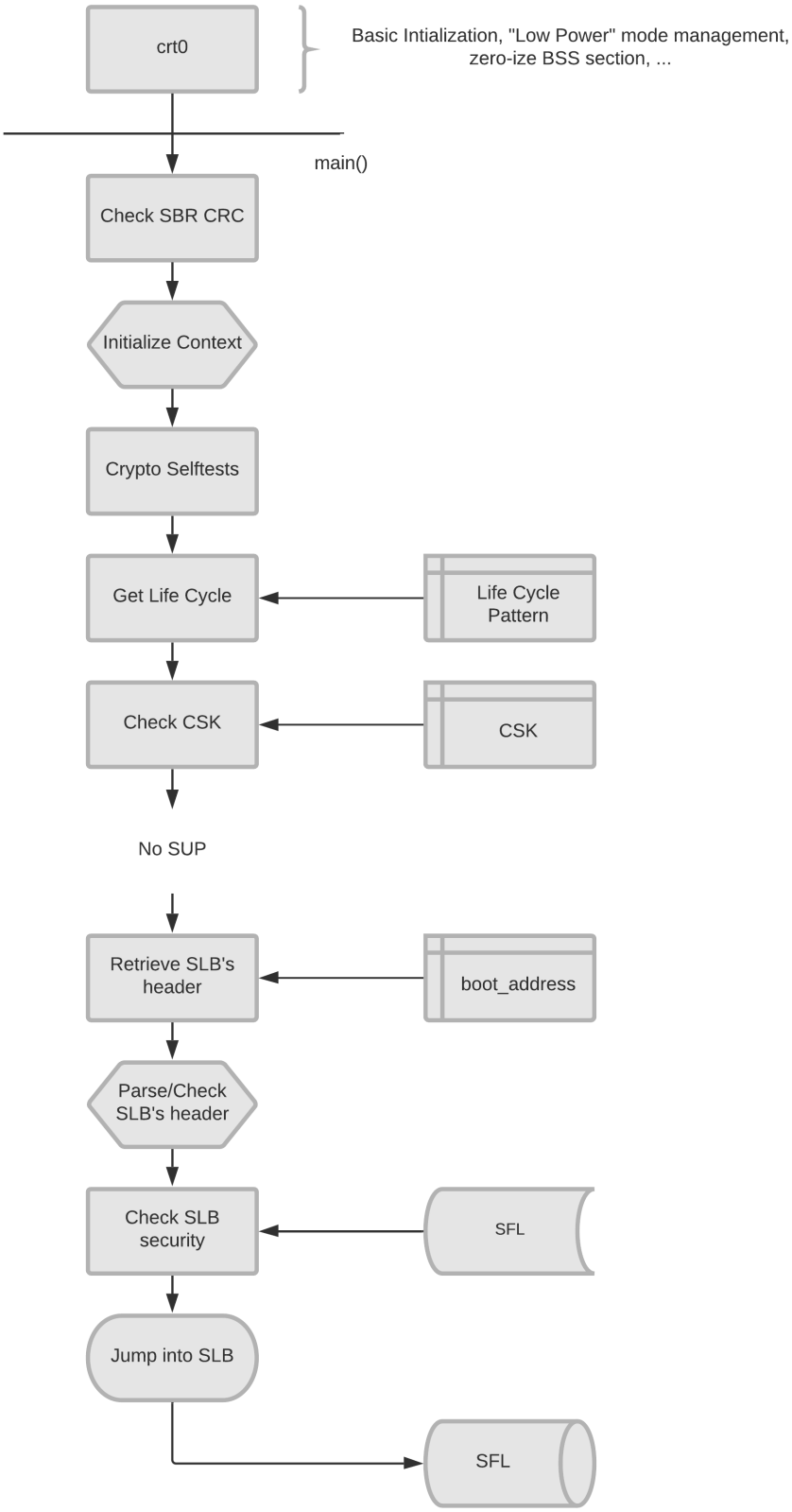
Second Level Boot

The SLB recalls at the same time the step where SBR check for next step software to launch and this software too. It has its dedicated page here.

Usecases

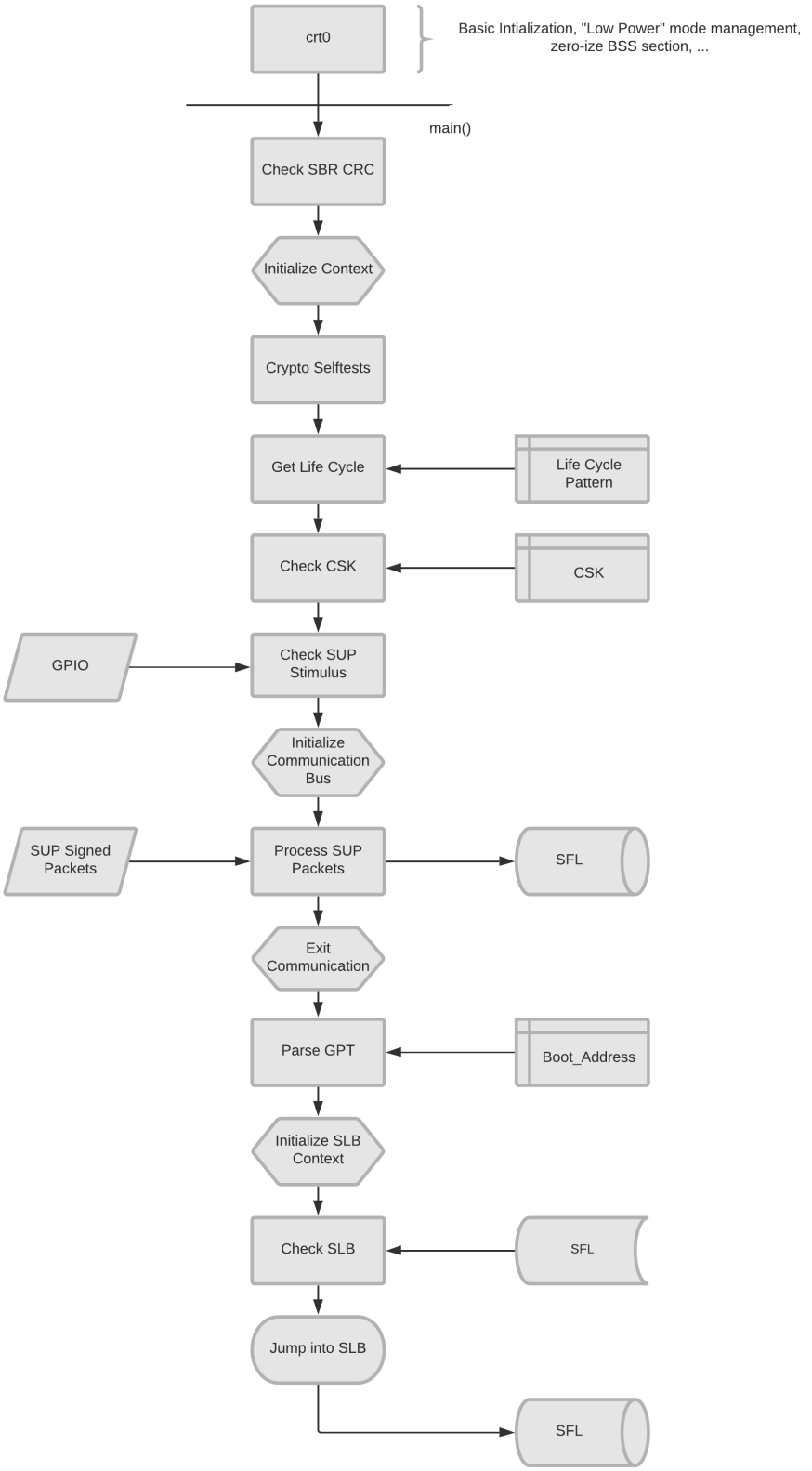
Nominal Boot - no update protocol - Valid SLB

In this usecase, platform's life cycle allows to launch a second level bootloader. CSK is present in OTP and valid. No SUP has been requested(no stimulus). "boot_address" is present in OTP and match platform's memory mapping. SFL is stored as described in GPT and match SBR format(header + signature).



Nominal Boot - update protocol - Valid SLB

In this usecase, platform's life cycle allows to launch a second level bootloader. CSK is present in OTP and valid. Stimulus is applied at boot and during SUP communication. SUP packets are valid then accepted by SBR. "*Boot_Address*" is present in OTP and match platform's memory mapping. SFL is stored as described in GPT and match SBR format$_{(\text{header + signature})}$.



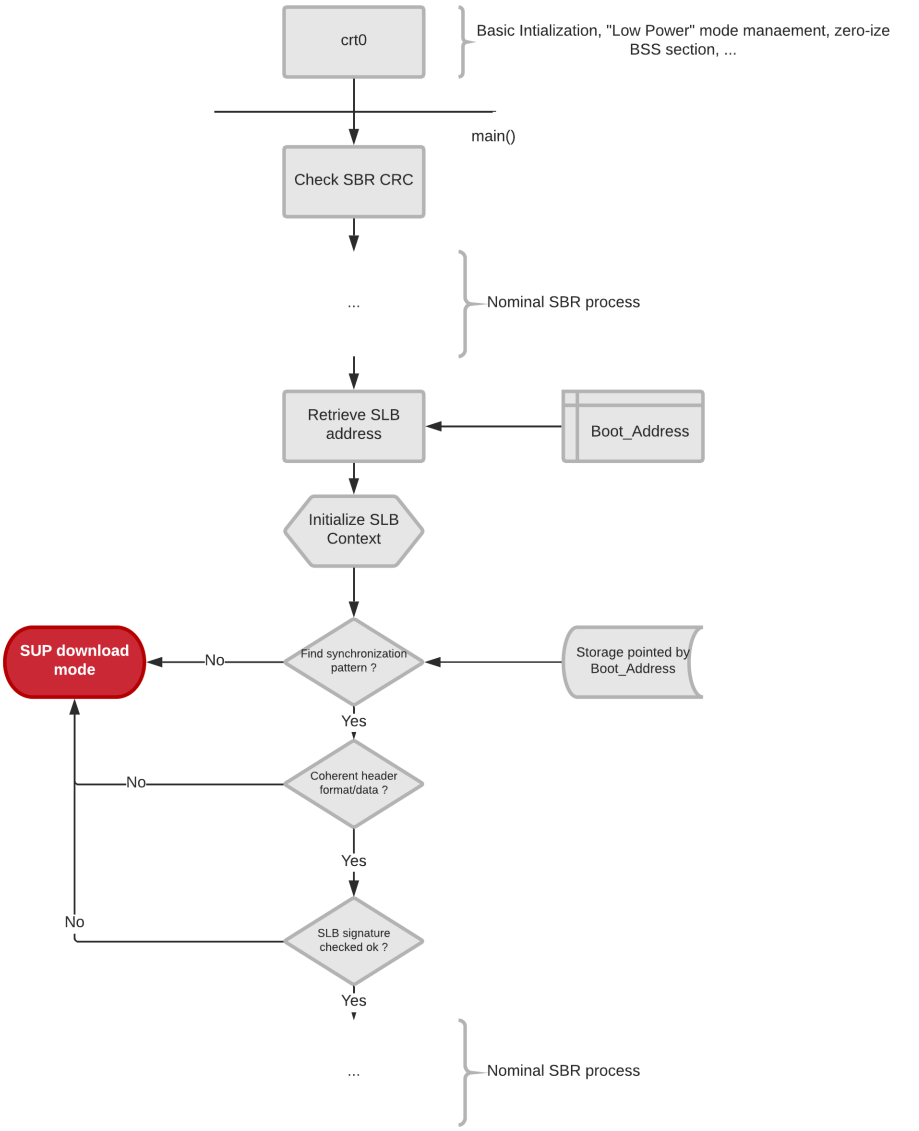
Nominal Boot - No update protocol - Invalid SLB

In this usecase, platform's life cycle allows to launch a second level bootloader. CSK is present in OTP and valid. No SUP has been requested(no stimulus).

"Boot_*Address*" present in OTP is checked, if it results with error, SBR goes into "*download mode*".

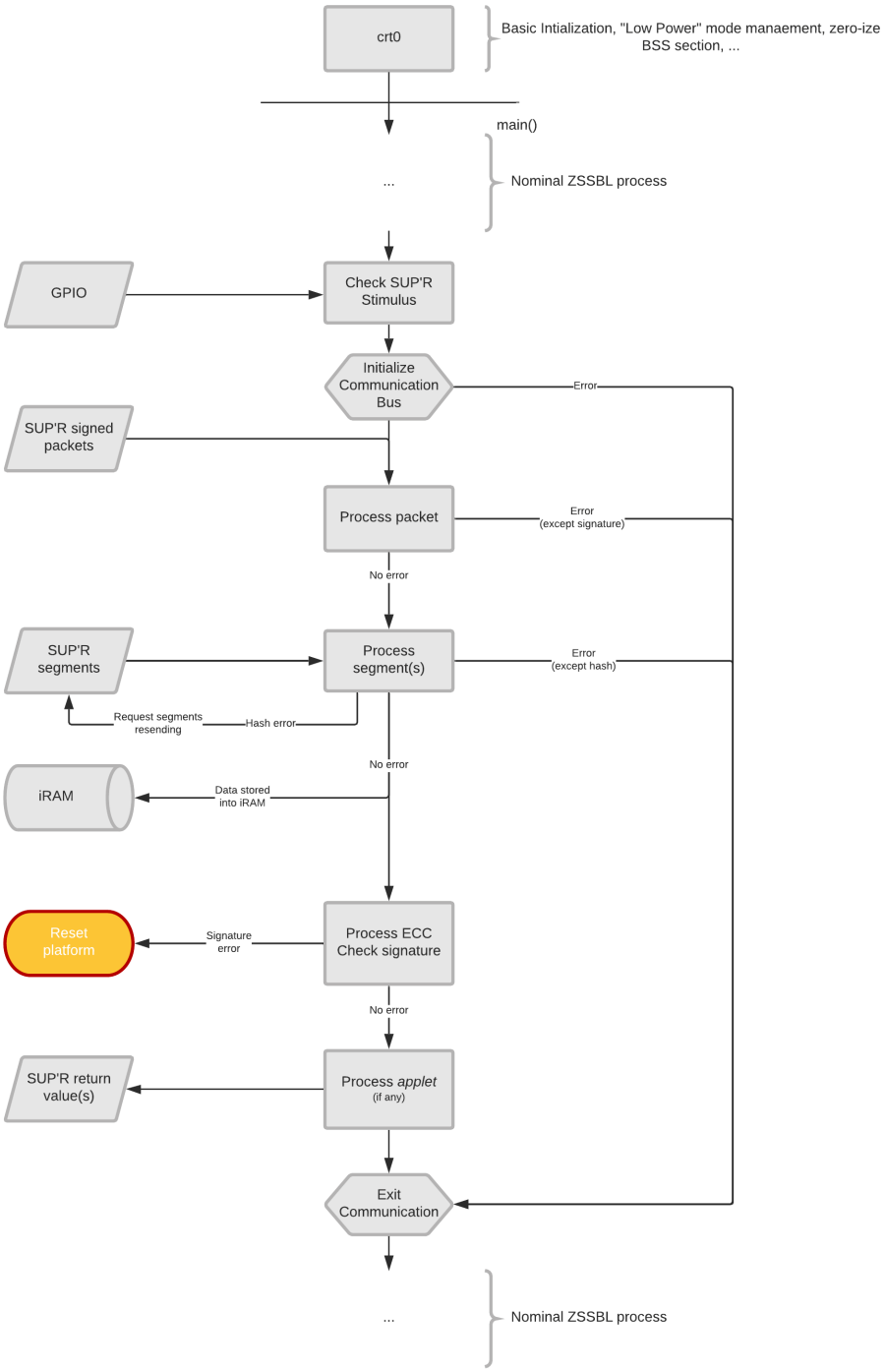
SLB's header is checked(synchronization pattern, addresses, ...), if it results with error, SBR goes into "*download mode*".

SLB's signature is checked(with CSK), if it results with error, SBR goes into "*download mode*".



Nominal Boot - Update protocols errors - N/A SLB

In this usecase is described only the "*high level*" error behaviors for SUP. For detailed, "*packet layer*" and "*segment layer*" processes, please refer to paragraphs describing SUP packet layer and SUP segment layer.


crt0
Basic Intialization, "Low Power" mode manaement, zero-ize BSS section, ...
main()
...
Nominal ZSSBL process
GPIO
Check SUP'R Stimulus
Initialize Communication Bus
Error
SUP'R signed packets
Process packet
Error (except signature)
No error
SUP'R segments
Process segment(s)
Error (except hash)
Request segments resending
Hash error
No error
iRAM
Data stored into iRAM
Reset platform
Signature error
Process ECC Check signature
No error
SUP'R return value(s)
Process applet (if any)
Exit Communication
...
Nominal ZSSBL process

[...]