Renesas

R01AN5548JJ0100
Rev.1.00
2020.08.31

ルネサス MCU におけるファームウェアアップデートの設計方針

要旨

このアプリケーションノートは、ルネサスMCUにおけるファームウェアアップデートの設計方針について説明します。

対象デバイス

- RX Family

関連ドキュメント

- RX ファミリに関する『RX ファミリにおけるファームウェアアップデートの実現方法 (R01AN5549JJ0100)』
- GitHub Renesas アカウント Amazon FreeRTOS リポジトリの wiki
  - https://github.com/renesas/amazon-freertos/wiki/OTA%E3%81%AE%E6%B4%BB%E7%94%A8
  - 本資料の原本です。最新情報はこちらをご参照ください

目次

1. 用語定義

ファームウェアアップデート	ファームウェアを更新すること
セルフプログラミング	ファームウェアの更新対象が自身である場合のその操作自体のこと
ブートローダ	Amazon Web Services が定義するブートローダ要件を満たすソフトウェア
セルフプログラミングを行うユーザプログラム領域(exe) 図中では「execute area」と呼称	ユーザシステム動作およびファームウェアアップデート動作を行うソフトウェアを格納する領域
書換用一時領域(tmp) 図中では「temporary area」と呼称	セルフプログラミングを行うユーザプログラム領域(exe)と同一容量で、ファームウェアアップデート時に新たなファームウェアを一時的に保持するための領域
ユーザプログラム領域の正当性確認を行うためのブートローダ領域(リセットベクタを含む) 図中では「bootloader」と呼称	ブートローダが格納される領域
ユーザプログラム領域の正当性確認を行うためのブートローダ領域(リセットベクタを含む)の写し。 図中では「bootloader(mirror)」と呼称	ブートローダの写し。RX65N 等のバンクスワップ機構を用いる時に必要

2. 概要

2.1 ファームウェアアップデートが求められる背景

組み込みシステムにおいてファームウェアアップデート機能が求められつつあります。その要因は以下のように考えられます。

- ソフトウェア不具合によるシステム動作不良が市場で顕在化しその修正のため
- 活用しているオープンソースソフトウェアのアップデート適用のため
- 市場要求の変化に追従するため

昨今、組み込みシステムがインターネットに繋がることにより、ソフトウェア不具合によるシステム動作不良が悪意のある第三者により引き起こされるリスクが増大しています。各国政府は特にインターネットに接続される電子機器についてファームウェアアップデート機能を搭載することをガイドラインに定め、さらに具体的な基準認証にファームウェアアップデート機能適用を定義するなど、対策を進めています。

日本国の例：

IoT セキュリティガイドライン

https://www.soumu.go.jp/main_content/000428393.pdf

電気通信事業法に基づく端末機器の基準認証に関するガイドライン(第 1 版)

https://www.soumu.go.jp/main_content/000615696.pdf

2.2 ファームウェアアップデート機能とインターネット接続機能

身近なファームウェアアップデートの例はスマートフォンです。スマートフォンは今や世界中の人々が使用する電子機器であり、その多くはインターネットと接続されています。スマートフォン上には OS (Operating System)が組み込まれており、OS がインターネット接続機能を提供しています。先述のようにスマートフォンでもファームアップデートが求められる背景が共通としてあり、日々OS ベンダによる OS のバージョンアップ、およびその配信が行われています。

仮にスマートフォンにファームウェアアップデート機能が無かったと仮定すると、悪意のある第三者により発見されたセキュリティホールが修正できないままとなり、スマートフォン内部に保存されている個人情報(名前、住所、年齢、メールアドレス、クレジットカード番号、各種パスワード等)が漏えいする等のリスクに晒されたままとなるでしょう。

仮にスマートフォンにインターネット接続機能が無かったと仮定すると、悪意のある第三者により上述のリスクが顕在化される可能性は下がることでしょう。

仮にスマートフォンにインターネット接続機能は有るがファームウェアアップデート機能が無かった場合は悪意のある第三者により上述のリスクが顕在化させられ、その状態が永続化することでしょう。

ファームウェアアップデート機能があれば、悪意のある第三者により上述のリスクが顕在化させられることはありますが、その状態が永続化することはありません。

以下にこの相関関係を表にまとめます。青字がメリット、赤字がデメリットです。

	インターネット接続機能有	インターネット接続機能無
ファームウェアアップデート機能有	リスク顕在化可能性高い リスク顕在化時永続化しない	リスク顕在化可能性低い リスク顕在化時永続化しない
ファームウェアアップデート機能無	リスク顕在化可能性高い リスク顕在化時永続化する	リスク顕在化可能性低い リスク顕在化時永続化する

セキュリティリスクの観点からは「ファームウェアアップデート機能有」「インターネット接続機能無」の組み合わせにおいて、最も安全となります。

ですが昨今、車や監視カメラや交通システム、家電製品や工業用ロボット、ビル設備やスマートメータなどあらゆる組み込み機器がインターネットに繋がり始めています。インターネット接続機能はその機能を搭載した電子機器に著しい機能向上をもたらすため、ファームウェアアップデートを代表とするセキュリティ対策が十分ではないまま製品化され市場でトラブルに繋がるケースが現れています。

この状態は上記表において「インターネット接続機能有」「ファームアップデート機能無」の組み合わせとなり、セキュリティの観点からは、最も危険となります。

従って「インターネット接続機能有」の製品を開発する場合は「ファームウェアアップデート機能有」を含むセキュリティ対策は必須と考えてください。このケースの場合「リスク顕在化可能性高い」となりますので、以下 2 点による製品ライフサイクルを通じたセキュリティ対策がさらに必要となります。

①リスク最小化できるようなセキュリティ開発プロセスを適用した製品開発(ハードウェア・ソフトウェア)

②リスク顕在化時に速やかにファームウェアアップデートを配信する運営体制の構築

また、「インターネット接続機能無」の場合でも「ファームウェアアップデート機能有」は「リスク顕在化時永続化しない」メリットのため推奨します。

本資料においては製品ライフサイクルについては省略し、ファームウェアアップデートの設計方針に注目して解説します。

2.3　ファームウェアアップデート機能の定義

スマートフォンや組み込み機器などの電子機器は一般的に「コンピュータ」に分類されます。多くのコンピュータは CPU、メモリ、I/O がバスで繋がった以下のような構成です。

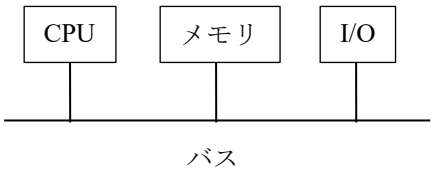


図 1. 一般的なコンピュータのシステムブロック図

次に実際のルネサス MCU の構成を確認します。以下は RX ファミリの RX65N のシステムブロック図の拡大図です。図 1 と図 2 において、CPU は RX CPU、メモリは ROM/RAM、I/O はポート x 等を指します。

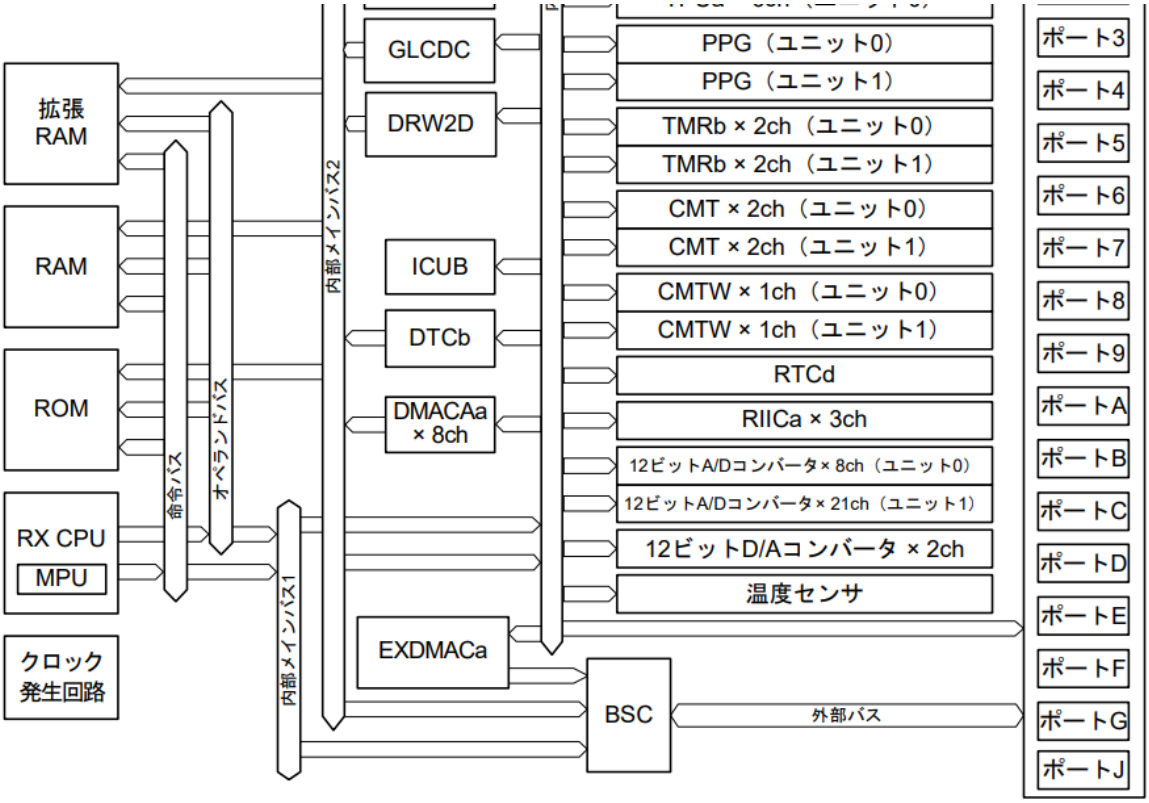


図 2.RX65N のシステムブロック図(拡大図)

RX65N において通常ソフトウェアコードは ROM に格納されます。RX65N はフラッシュメモリ内蔵マイコンであり、上記図において ROM=フラッシュメモリです。フラッシュメモリは不揮発性メモリであり電源が無くても書き込まれたデータが揮発することがありません。また、RX65N(他多くのフラッシュメモリ内蔵マイコン)ではフラッシュメモリに対し CPU から書込み・消去といったコマンドを送出できるメカニズムとなっています。この機能を活用することで、フラッシュメモリに書き込まれたソフトウェアを書き換えることが可能となります。市場投入後にこの機能を使用してソフトウェアを更新することを「ファームウェアアップデート」と呼びます。またその動作自体を「セルフプログラミング」と呼びます。

3. フラッシュメモリ書き換えの仕様

3.1 RX ファミリ

ここではフラッシュメモリを書き換える方法について RX65N を例にその具体例を交えて解説します。

オンボードプログラミング (シリアルプログラミング/セルフプログラミング)	ブートモード(SCIインタフェース)によるプログラム/イレーズ • 調歩同期式シリアルインターフェース(SCI1)を使用 • 通信速度は自動調整 ブートモード(USBインタフェース)によるプログラム/イレーズ • USBb を使用 • 特別なハードウェアが不要で、PC と直結可能 ブートモード(FINEインタフェース)によるプログラム/イレーズ • FINE を使用 セルフプログラミングによるプログラム/イレーズ • システムをリセットすることなくフラッシュメモリのプログラム/イレーズが可能	
オフボードプログラミング (注4)	パラレルプログラマを使用して、コードフラッシュメモリ、オプション設定メモリのプログラム/イレーズが可能	パラレルプログラマを使用したデータフラッシュメモリのプログラム/イレーズはできません

図 3. RX65N フラッシュメモリの仕様から抜粋

RX65N ではフラッシュメモリを書き換える方法は「オンボードプログラミング」「オフボードプログラミング」に大別されます。「オフボードプログラミング」は主に電子機器組み立て工程において用いられます。RX65N はルネサス出荷時にブランク状態(フラッシュメモリに何も書き込まれていない状態)でありブランク状態で書き込み業者工場においてソフトウェア書き込みを行い組み立て工場に移送、ソフトウェア書き込みが済んだ RX65N マイコンを電子機器内部の基板に実装し組み立てを行い、最終検査後出荷される、という工程が一般的です。

電子機器が市場投入されたあとに「オフボードプログラミング」するためには、電子機器内部の基板から RX65N を剥がす必要があり、エンドユーザにファームウェアアップデートのため「オフボードプログラミング」をさせることは非現実的です。従ってファームウェアアップデートは「オンボードプログラミング」で行います。

さらに「オンボードプログラミング」は「シリアルプログラミング」と「セルフプログラミング」に大別されます。「シリアルプログラミング」は「ブートモード(各種)」、「セルフプログラミング」は「シングルチップモード」で動作します。以下に RX65N の動作モードの状態遷移図を示します。

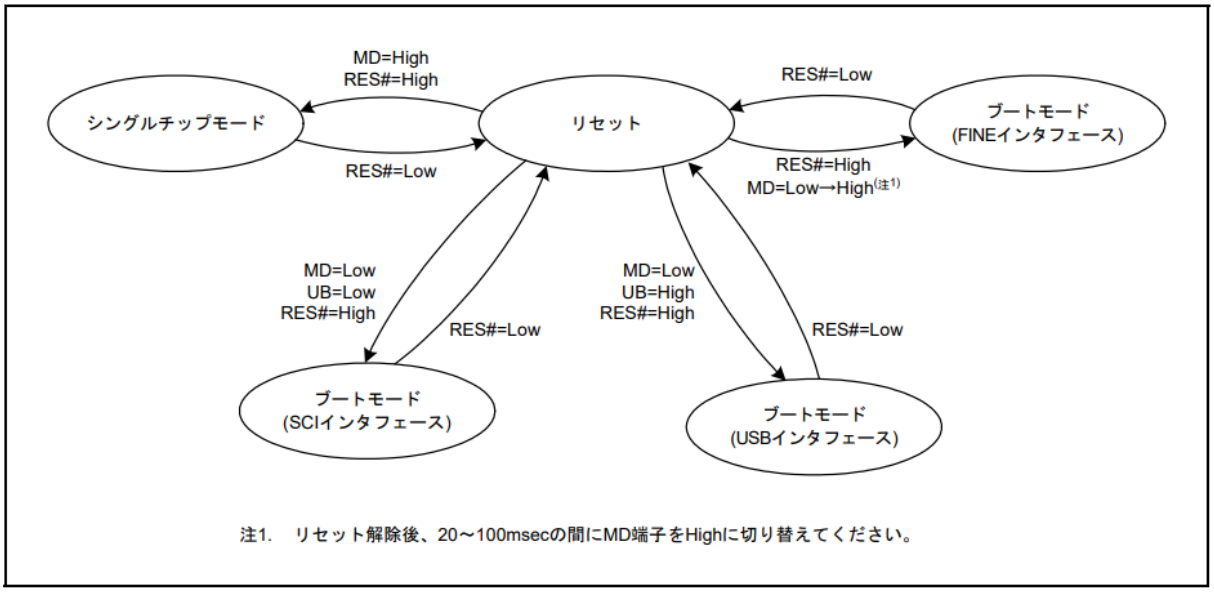


図 3. RX65N フラッシュメモリの仕様から抜粋

上記における MD や UB、RES# は RX65N の端子名です。そしてシステム動作はシングルチップモードで行います。つまり、「ファームウェアアップデート機能」を「ブートモード」で実行するということは、エンドユーザに対し MD や UB、RES#の操作を強いることになります。エンドユーザが人間でありいつでも製品上スイッチ等から MD や UB、RES#を操作できるのであれば「ブートモード」による「ファームアップデート機能」も可能です。

ですが、スマートフォン等の「ファームウェアアップデート機能」では MD や UB、RES#を操作するようなことをユーザに強いることはなく、多くのエンドユーザはその環境に既に慣れているため、特段の事情が無い限り「シングルチップモード」でユーザシステム動作上に「ファームウェアアップデート機能」を実現することを推奨します。

また、スマートフォンのように人間一人ずつが所有するような組み込み機器ではなく、例えば橋梁状態監視システムや太陽光発電パネルのように、近くに人間がいない状態で稼働し続ける必要のある組み込み機器の場合、「シングルチップモード」での「ファームウェアアップデート機能」は必須要件になるでしょう。

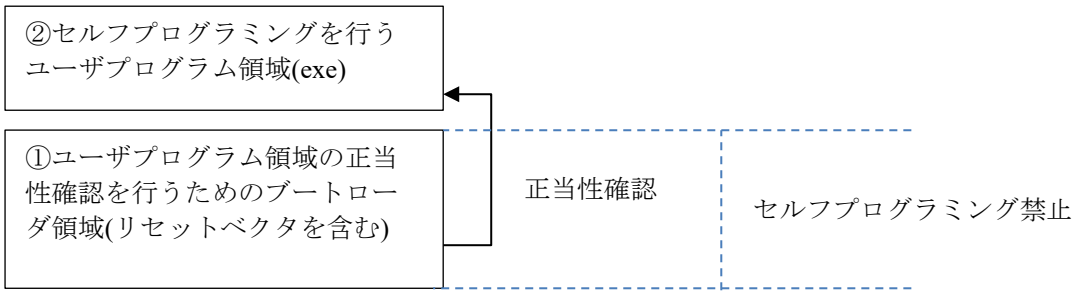
4. ファームウェアアップデートのメカニズム

4.1 設計方針

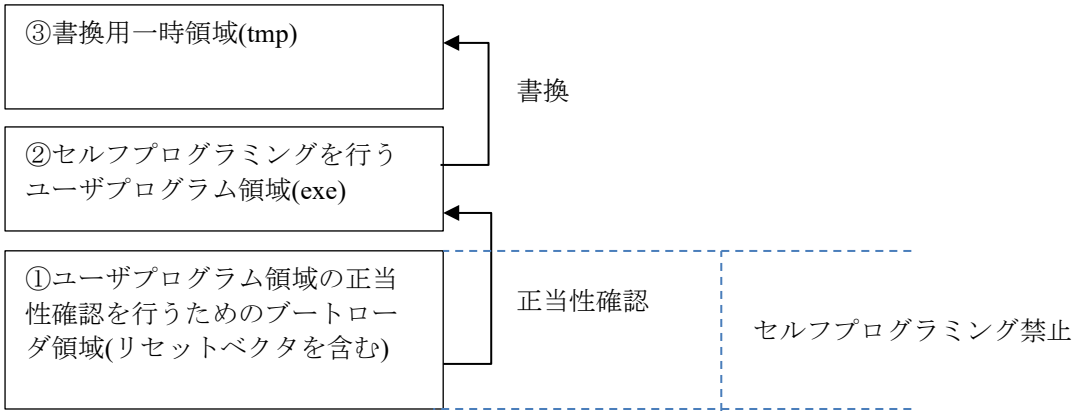
「ファームウェアアップデート機能」を実現するためには特定の通信路(UART 等)経由でマイコンの外部からアップデート用ファームウェアを受信しそれを「セルフプログラミング」します。ここで以下 2 点の重大な実装上の課題があります。

課題 1: セルフプログラミングの途中で電源が遮断された場合復旧可能か？
（電源復旧後どのように検出するのか？）
課題 2: セルフプログラミングを実行するソフトウェアを書き換えている途中で電源が遮断された場合は復旧可能か？

課題 1 を解消するためには、②セルフプログラミングを行うユーザプログラム領域と①ユーザプログラム領域の正当性確認を行うブートローダ領域を分離し、①をリセット解除後に必ず実行する必要があります。セルフプログラミングの異常により①の領域が破壊されないよう、①の領域のみセルフプログラミング禁止に設定できる必要があります。

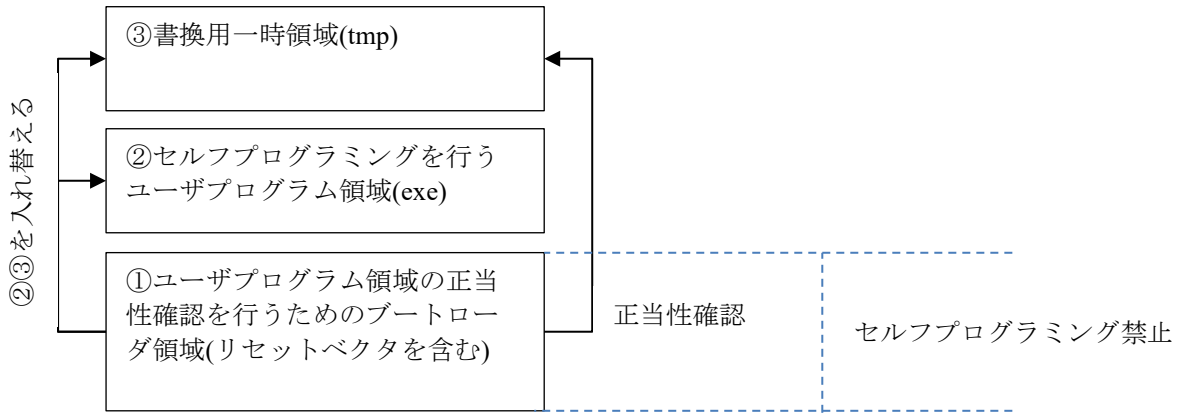


課題 2 を解消するためには、②セルフプログラミングを行うユーザプログラム領域(exe)と同じ容量の③書換用一時領域(tmp)の 2 面を持ち、③を書き換えている最中に電源が遮断されて正当性確認が成立しない状態(書換失敗)に陥ったとしても、②は正当性確認が成功する状態(書換成功)を維持できるようにします。



課題 2 を解消すると以下新たな課題が発生します。

課題 3:「ファームウェアアップデート機能」で転送されてくる新しいファームウェアは②セルフプログラミングを行うユーザプログラム領域(exe)で実行されることを前提に生成されていることが多いため、③書換用一時領域(tmp)に配置されたままだと実行不可能



課題 3 を解消するためには、ブートローダ領域が③書換用一時領域(tmp)の正当性確認を行い、正当性確認が成功する状態(書換成功)であれば②③を入れ替える(②を消去して③を②の領域にコピー)操作を実装します。RX65N マイコンのようにマイコン機能としてデュアルバンク機能/バンクスワップ機能がある場合はそれを使用することで上記コピー操作が省けるため推奨します。

4.2 一般的なブートローダ要件 1

「ファームウェアアップデート機能」を実現するための基礎部分として一般的に「ブートローダ」の実装が必要です。ルネサス MCU によけるファームウェアアップデートにおいては Amazon Web Services 社が提起する以下要件を参考に 8 種類の要件を抽出しました。

https://github.com/aws/amazon-freertos/blob/4d52fdb0e85c14d88c895ae6c144668ced8409af/tests/Amazon%20FreeRTOS%20Qualification%20Developer%20Guide.pdf

→5.10 Appendix J: Bootloader

注意：上記 URL は Amazon FreeRTOS v1.4.7 のものであり、最新版では考え方が更新されている可能性がある。適宜最新版の考え方を元に見直しを図る必要がある。

原文

1. The bootloader shall be stored in non-volatile memory so it cannot be overwritten.
2. The bootloader shall verify the cryptographic signature of the downloaded application image. Signature verification must be consistent with the OTA image signer. See Appendix I: OTA Updates for supported signatures.
3. The bootloader shall not allow rolling back to a previously installed application image.
4. The bootloader shall maintain at least one image that can be booted.
5. If the MCU contains more than one image then the image that is executed shall be the latest (newest). The newest version can be determined based on implementation, for example a user defined sequence number, application version etc. As per other requirements, this can only be the case until a newer version has been verified and proven functional.
6. If the MCU cannot verify any images then it shall place itself into a controlled benign state. In this state it prevents itself from being taken over by ensuring no actions are performed.
7. These requirements shall not be breached even in the presence of an accidental or malicious write to any MCU memory location (key store, program memory, RAM, etc.)
8. The bootloader shall support self-test of a new OTA image. If test execution fails, the bootloader shall roll back to the previous valid image. If test execution succeeds, the image shall be marked valid and the previous version erased.

日本語訳

1. ブートローダは不揮発性メモリに格納され、上書きできてはいけない
2. ブートローダはダウンロードしたアプリケーションイメージに対して電子署名を用いて検証しなければならない。電子署名は OTA image signer と互換である必要がある。詳しくは Appendix I: OTA Updates を参照。
3. ブートローダは直前にインストールされていたアプリケーションにロールバックできてはいけない
4. ブートローダは少なくとも 1 個のイメージでブートできるようシステムを維持できなければならない
5. もし MCU が 1 個以上のイメージを持つ場合、起動できるイメージは最新のものでなければならない。バージョン選択は実装依存であり、それはたとえばユーザ定義のシーケンス番号であったりアプリバージョンであったりする。ただし、バージョン選択できるのは、最新のイメージが確定するまでの間である。
6. もし MCU がイメージの検証に失敗する場合であっても正しく制御された"検証が失敗した"状態を維持できなければならない。この状態においては何ら有効な動作が行われない状態となり、そこから遷移できてしまってはいけない。
7. これらの要件は事故や悪意のあるコードなどによる不正 ROM 書き換えによって妨げられてはならない。
8. ブートローダは新しい OTA イメージに対しセルフテストを行えなければならない。このセルフテストが失敗するならば、ブートローダは直前の正常なイメージにロールバックできなければならない。もしセルフテストに成功するならば、ブートローダは直前の正常なイメージを削除し、新しい OTA イメージを正常なイメージと認識するようマークすることができる。

4.3 一般的なブートローダ要件 2

IETF(Internet Engineering Task Force)がファームウェアアップデートに関する要件をまとめています。
https://datatracker.ietf.org/doc/draft-ietf-suit-architecture/

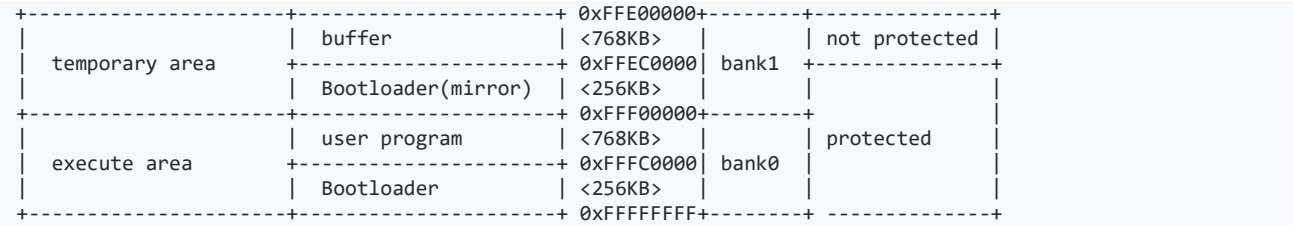
ルネサスはこのように一般的なブートローダ要件の情報を収集しつつ、適切なブートローダ実装を更新し続ける方針です。

4.4 一般的なブートローダ要件 と RX65N ブートローダ実装との整合

ここでは、一般的なブートローダ要件 1 で定義される 1~8 の項目について RX65N における実装との整合を確認します。

1. ブートローダは不揮発性メモリに格納され、上書きできてはいけない

[RX65N における実装]
ブートローダはフラッシュメモリ(不揮発性メモリ)に格納します。
Flash Access Window(FAW)機能を用いて 0xFFEC0000-0xFFFFFFFF の領域をセルフプログラミング禁止に指定することで上書き禁止となります。
さらに、FAW レジスタにある FSPR ビット(アクセスウィンドウプロテクトビット)にゼロをセットすることで FAW レジスタの内容を完全に固定化し、以降 FAW レジスタそのものの書き換えを完全に禁止することができます。同時に以下図における「protected」領域も完全に書き換えが禁止となりますので、注意が必要です。



補足：
RX65N はバンクスワップ機能を持ちます。上記図において bank0 と bank1 のアドレスをスワップできますが、スワップ対象にはブートローダも含まれており、0xfffffffc 番地にあるリセットベクタを含めてバンクスワップ対象となるため、bank1 側にもブートローダと同一内容のブートローダ(mirror)を配置する必要があります。
ブートローダは初回起動時にセルフプログラミングにより自身と同じデータをブートローダ(mirror)領域にコピーします。
なお、バンクスワップ時に bank0 と bank1 のアドレスはスワップしますが、FAW による保護対象は変化しません。すなわちセルフプログラミング可能な領域はいかなる場合でも buffer 領域のみです。

2. ブートローダはダウンロードしたアプリケーションイメージに対して電子署名を用いて検証しなければならない。電子署名は OTA image signer と互換である必要がある。詳しくは Appendix I: OTA Updates を参照。

[RX65N における実装]
ブートローダおよびユーザプログラムには電子署名検証のため ECDSA+SHA256 のアルゴリズムを実装します。このアルゴリズムは Amazon Web Services の OTA image signer と互換です。

3. ブートローダは直前にインストールされていたアプリケーションにロールバックできてはいけない

[RX65N における実装]
ブートローダは buffer 領域(新しいアプリケーションイメージが格納されている)の検証後、user program 領域(直前にインストールされていたアプリケーションが格納されている)の削除を行うため直前にインストールされていたアプリケーションにロールバックができません。

4. ブートローダは少なくとも 1 個のイメージでブートできるようシステムを維持できなければならない

[RX65N における実装]
以下 2 点のタイミングにおいてフラッシュメモリの書換が発生します。
①user program 領域が buffer 領域の消去・書換を行う
②Bootloader 領域が user program 領域の消去・書換を行う

①においては user program 領域が、②においては buffer 領域が完全な状態で維持されるため任意のタイミングで電源遮断等が発生してシステムがリブートしたとしても Bootloader が buffer 領域、user program 領域のいずれかが完全な状態であることを検出し以下のように復旧を試みます。

表 1. Bootloader の挙動

	user program 領域 完全	user program 領域 不完全
buffer 領域 完全	user program 領域消去→ buffer 領域を user program 領域にコピー→ user program 起動	user program 領域消去→ buffer 領域を user program 領域にコピー→ user program 起動
buffer 領域 不完全	user program 起動	起動不可

5. もし MCU が 1 個以上のイメージを持つ場合、起動できるイメージは最新のものでなければならない。
バージョン選択は実装依存であり、それはたとえばユーザ定義のシーケンス番号であったりアプリバージョンであったりする。ただし、バージョン選択できるのは、最新のイメージが確定するまでの間である。

[RX65N における実装]
「表 1. Bootloader の挙動」において、新しいイメージは常に buffer 領域に格納されます。buffer 領域と user program 領域いずれも完全な状態であればその後起動されるのは必ず buffer 領域にあるイメージ(が user program 領域にコピーされたもの)になります。
将来イメージに含まれる Sequence Number を活用し、1 個前のイメージの Sequence Number より大きな Sequence Number を持つイメージでないと、受け付けないように改良する予定です。このとき Sequence Number はより強固に守るため、セキュリティ IP と連動すると尚良いと考えます。現状はそこまで市場要求が高まってないと思われるため、フレームだけ用意して未実装とします。

6. もし MCU がイメージの検証に失敗する場合であっても正しく制御された"検証が失敗した"状態を維持できなければならない。この状態においては何ら有効な動作が行われない状態となり、そこから遷移できてしまってはいけない。

[RX65N における実装]
「表 1. Bootloader の挙動」において、「buffer 領域不完全」「user program 領域不完全」の状態がこの要件に当てはまります。この状態になると、起動不可となり他の状態に遷移できません。

7. これらの要件は事故や悪意のあるコードなどによる不正 ROM 書き換えによって妨げられてはならない。

[RX65N における実装]
FAW(Flash Access Window)レジスタおよび FSPR ビットの働きにより Bootloader は完全に書き換え禁止となります。また、上記要件を実現しているのは Bootloader です。仮に user program に不具合があり悪意のある第三者により不正に ROM 書き換えコマンドが発行されたとしても Bootloader が破壊されることはありません。

8. ブートローダは新しい OTA イメージに対しセルフテストを行えなければならない。このセルフテストが失敗するならば、ブートローダは直前の正常なイメージにロールバックできなければならない。もしセルフテストに成功するならば、ブートローダは直前の正常なイメージを削除し、新しい OTA イメージを正常なイメージと認識するようマークすることができる。

[RX65N における実装]

ブートローダは新しい OTA イメージの正当性確認はできますが、セルフテストを行うためには新しいイメージを起動し実際にネットワーク通信を試みてみる必要があります。一方でセルフテストが失敗するならばロールバックしなければならない、とあり矛盾しています。この要件は達成できないのでここでは無視します。今後、この要件の定義元と協議して矛盾を解消する、または標準規格での定義を確認する必要があると考えています。

5. ブートローダ方式

ブートローダの方式は、新しいファームウェアダウンロード時にそのデータをどこにバッファリングするかにより以下 3 種類に大別されます。

１．方式①：バッファリング先＝バンクの片方(デュアルバンク/バンクスワップ機構を使用)
２．方式②：バッファリング先＝バンクの片方(デュアルバンク/バンクスワップ機構を未使用)
３．方式③：バッファリング先＝外部メモリ(EEPROM やシリアルフラッシュ)

また、ファームウェアがどの通信路を通じてダウンロードされるかについては、以下のような通信路が考えられます。

A. UART
B. SD カードや USB メモリなどファイルシステム経由
C. Ethernet や WiFi などネットワーク経由(OTA: Over The Air)

ルネサス MCU におけるファームウェアアップデートはこれらを柔軟に組み合わせられるよう設計を進めています。

2020/06/30 時点では、以下の組み合わせが実現可能です。

ブートローダ：１．A

ユーザプログラム：１．C

この組み合わせが実現可能なサンプルコードは以下に公開しています。

https://github.com/renesas/amazon-freertos

動作方法などの詳細は以下アプリケーションノートを参照してください。

- RX ファミリに関する『RX ファミリにおけるファームウェアアップデートの実現方法 (R01ANxxxx)』

5.1 方式①：バッファリング先 = バンクの片方(デュアルバンク/バンクスワップ機構を使用)

ここでは RX65N を代表例として「ブートローダの方式①：**バッファリング先 = バンクの片方(デュアルバンク/バンクスワップ機構を使用)**」を解説します。

5.1.1 フラッシュメモリの仕様

RX65N 世代以降、フラッシュメモリのコードフラッシュ領域はデュアルバンク機構を備えており、物理メモリ量 2MB(max)を 2 分割できます。
注意：以降の例では物理メモリ量は 2MB を前提に説明します。実際は ROM 容量 1.5MB 版などもあるので自身が使用するマイコンの ROM 容量を正しく認識し、読み替えてください。

RX65N ではこの 2 分割するモードを「デュアルモード」と呼称しています。「デュアルモード」では、「バンクスワップ」と呼ばれる機能を使うことができます。本稿では、以下のようにバンク面を定義します。bank0 は実行面(execute area)、bank1 はバッファ面(temporary area)です。リセットベクタは常に 0xfffffffc-0xffffffff 番地に存在します。従ってバンクスワップ後はリセット後に起動するファームウェアが bank1 (firmware 1)となります。

```
+---------------------------------+ 0xFFE00000 +
|                                 |            |
|       temporary area            |            | bank1 (firmware 1)
|                                 |            |
+---------------------------------+ 0xFFF00000 +
|                                 |            |
|       execute area              |            | bank0 (firmware 0)
|                                 |            |
+---------------------------------+ 0xFFFFFFFF + <- reset vector
```

バンクスワップを実行すると(具体的にはフラッシュ API の R_FLASH_Control()でバンクスワップコマンドを発行した後にリセット)、bank0 にあるコードと bank1 にあるコードがスワップ(入れ替わる)します。

```
+---------------------------------+ 0xFFE00000 +
|                                 |            |
|       temporary area            |            | bank0 (firmware 0)
|                                 |            |
+---------------------------------+ 0xFFF00000 +
|                                 |            |
|       execute area              |            | bank1 (firmware 1)
|                                 |            |
+---------------------------------+ 0xFFFFFFFF + <- reset vector
```

5.1.2 フラッシュメモリの仕様おさらい: 補足①

このバンクスワップ動作がない場合は、bank0 用のファームウェア、bank1 用のファームウェアといった形でコンパイル時にどちらのバンク用のファームウェアなのかを識別して管理する必要がありました。常に execute area 面で実行することを制限にすることで、この管理が不要となります。このデュアルバンク機構は、1 チップでファームウェアアップデートを実現するための常套手段です。1 チップでファームアップデートを実現する場合、新しいイメージを格納しておくためのバッファ領域がチップ内で必要となり、多くの場合はこれまでに説明した通り、「物理メモリ容量の半分をバッファとして使用する」ことになります。RX65N は最大 2MB の ROM 容量がありますが、ユーザが使用できる量は 1MB であることに先ず注意が必要です。外部シリアルフラッシュ・EEPROM 等のストレージを配するための部品代に制約がないならばこの限りではありません。この場合はむしろ RX65N のようにデュアルバンク機構をもった MCU であったとしても、バッファ領域を外部シリアルフラッシュ・EEPROM とし、MCU 内部フラッシュメモリを全面ユーザ領域とするのが健全です。

5.1.3 フラッシュメモリの仕様おさらい: 補足②

従来のフラッシュメモリ内蔵マイコンは、フラッシュメモリ書換時に RAM にファームウェア書換動作を行うプログラムを配置し、RAM にプログラムカウンタを飛ばし、RAM 上からフラッシュメモリ書換コマンドを実行しなければなりませんでした。このため、システム動作を維持したままフラッシュメモリを書き換えることは非常に難しいです。RX65N のようにバンク構造を持つフラッシュメモリ内蔵マイコンでは、BGO(バックグラウンドオペレーション)機能により、この問題を回避できる製品が存在します。BGO 機能を使うことで、例えば bank0 に bank1 に対するファームウェア書換動作を行うプログラムを配置し実行することが可能となります。このとき、割り込みベクタおよび割り込みベクタのジャンプ先にある割り込み処理関数もすべて bank0 に配置しておけば、システム動作を維持したままフラッシュメモリを書き換えることが非常に容易となります。

5.2 方式②：バッファリング先 = バンクの片方(デュアルバンク/バンクスワップ機構を未使用)

本項目は今後拡張予定です。

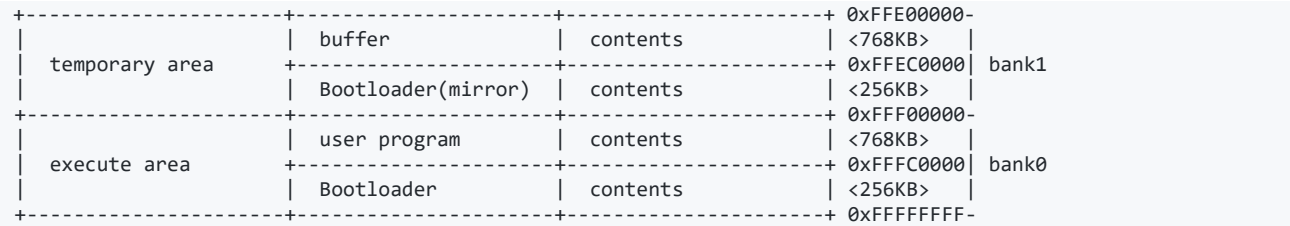
5.3 方式③：バッファリング先 = 外部メモリ(EEPROM やシリアルフラッシュ)

本項目は今後拡張予定です。

6. ブートローダ実装

6.1 メモリマップ定義

ファームウェアアップデート機構を実現するための RX65N のメモリマップを以下のように定義します。contents は初期状態(ブランクチップ)では blank とし、以降説明時に何かを書き込んだ場合、contents 部分が新しいデータ名に置き換わります。



ブートローダ用の 256KB はブート時に実行したい処理に依存するため、ユーザ依存です。256KB は相当余裕のある数値ですが、ネットワーク経由の初期ファームロードを考える場合は Amazon FreeRTOS などのネットワークスタックをブートローダに入れなければならずその場合は不足するかもしれません。逆にネットワークロードが不要で、初期ファームウェアのロード時に UART を使う場合は少ない容量(16KB 等)で済む場合もあります。本稿ではネットワーク経由の初期ファームロードも想定に入れているため、その動作に十分な 256KB としました。この場合、ユーザアプリが使用できる ROM 容量は 768KB となります。ユーザシステムによってブートローダに割り当てる容量は適切に変更してください。

Bootloader(mirror)というのは、バンクスワップ後に bank0 と bank1 の内容が入れ替わったあとでも、まったく同じ Bootloader が動作するようにするために必要な部分です。マイコンとしてのリセットベクタが 0xFFFFFFFC であるため、バンクスワップ動作に備えて bank0 と bank1 の下位エリアには同じ Bootloader を仕込んでおく必要があります。

user program の先頭 0x300 バイトは Bootloader が user program を検証するための署名データ等を入れる領域のため、user program の開始アドレスは 0x300 バイト後方にずらしてセクション設定を行う必要があります。データフォーマットについては別の章で詳細を解説しています。

6.2 ファームウェア書き込み順序

1. ブランクチップを用意する
2. Bootloader を ROM ライタで書き込む
3. Bootloader を実行し、Bootloader(mirror)をセルフプログラミングでコピー
4. Bootloader のロード機能を使用し、temporary area に initial firmware を書き込みソフトウェアリセット
5. Bootloader が temporary area の initial firmware が正しいことを検証する
6. ソフトウェアリセットを発行し、バンクスワップする
7. bank0 に再配置された Bootloader(mirror)が bank0 に再配置された temporary area にある initial firmware が正しいことを検証する
8. FAW によって、Bootloader(mirror), user program, Bootloader にロックをかける
9. Bootloader(mirror)が initial firmware にジャンプする（initial firmware が起動する）
10. initial firmware が temporary area に next firmware を書き込む
11. initial firmware が temporary area の next firmware が正しいことを検証する
12. ソフトウェアリセットを発行し、バンクスワップする
13. bank0 に再配置された Bootloader が bank0 に再配置された next firmware が正しいことを検証し、bank1 に残っている initial firmware を消去する
14. Bootloader が次期ファームウェアにジャンプする（next firmware が起動する）
15. 繰り返し

6.3 ファームウェア書き込み順序(図解)

1. ブランクチップを用意する

```
+---------------------+---------------------+---------------------+ 0xFFE00000-
|                     |  buffer             |  blank              | <768KB>   |
|  temporary area     +---------------------+---------------------+ 0xFFEC0000| bank1
|                     |  Bootloader(mirror) |  blank              | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFF00000-
|                     |  user program       |  blank              | <768KB>   |
|  execute area       +---------------------+---------------------+ 0xFFFC0000| bank0
|                     |  Bootloader         |  blank              | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFFFFFFF-
```

2. Bootloader を ROM ライタで書き込む

```
+---------------------+---------------------+---------------------+ 0xFFE00000-
|                     |  buffer             |  blank              | <768KB>   |
|  temporary area     +---------------------+---------------------+ 0xFFEC0000| bank1
|                     |  Bootloader(mirror) |  blank              | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFF00000-
|                     |  user program       |  blank              | <768KB>   |
|  execute area       +---------------------+---------------------+ 0xFFFC0000| bank0
|                     |  Bootloader         |  Bootloader         | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFFFFFFF-
```

3. Bootloader を実行し、Bootloader(mirror)をセルフプログラミングでコピー

```
+---------------------+---------------------+---------------------+ 0xFFE00000-
|                     |  buffer             |  blank              | <768KB>   |
|  temporary area     +---------------------+---------------------+ 0xFFEC0000| bank1
|                     |  Bootloader(mirror) |  Bootloader(mirror) | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFF00000-
|                     |  user program       |  blank              | <768KB>   |
|  execute area       +---------------------+---------------------+ 0xFFFC0000| bank0
|                     |  Bootloader         |  Bootloader         | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFFFFFFF-
```

4. Bootloader のロード機能を使用し、temporary area に initial firmware を書き込みソフトウェアリセット

```
+---------------------+---------------------+---------------------+ 0xFFE00000-
|                     |  buffer             |  initial firmware   | <768KB>   |
|  temporary area     +---------------------+---------------------+ 0xFFEC0000| bank1
|                     |  Bootloader(mirror) |  Bootloader(mirror) | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFF00000-
|                     |  user program       |  blank              | <768KB>   |
|  execute area       +---------------------+---------------------+ 0xFFFC0000| bank0
|                     |  Bootloader         |  Bootloader         | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFFFFFFF-
```

5. Bootloader が temporary area の initial firmware が正しいことを検証する
6. ソフトウェアリセットを発行し、バンクスワップする

```
+---------------------+---------------------+---------------------+ 0xFFE00000-
|                     |  buffer             |  blank              | <768KB>   |
|  temporary area     +---------------------+---------------------+ 0xFFEC0000| bank0
|                     |  Bootloader(mirror) |  Bootloader         | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFF00000-
|                     |  user program       |  initial firmware   | <768KB>   |
|  execute area       +---------------------+---------------------+ 0xFFFC0000| bank1
|                     |  Bootloader         |  Bootloader(mirror) | <256KB>   |
+---------------------+---------------------+---------------------+ 0xFFFFFFFF-
```

7. bank0 に再配置された Bootloader(mirror)が bank0 に再配置された temporary area にある initial firmware が正しいことを検証する
8. FAW によって、Bootloader(mirror), user program, Bootloader にロックをかける

   この操作はフラッシュモジュールの R_FLASH_Control()でできます。このとき、FSPR "0" を書けば FAW が永久にロックされます。危険なのでユーザがソースコードの該当行を書き換えて使用することにしています。 ロックする場合、r_flash_type4.c の「faw.BIT.FSPR = 1;」を「faw.BIT.FSPR = 0;」に変更してください。

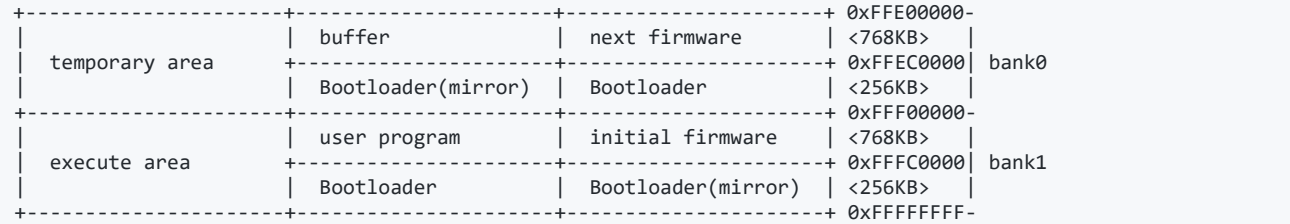
   →FSPR の設定変更のためのコード

   ブートローダにおいて Bootloader(mirror)のコピーが終わったところで、フラッシュ API の R_FLASH_Control()で FAW レジスタ設定を行います。サンプルコードを活用する実験中においてはチップをブランク状態に戻して再利用する必要があるため、ロックをかけるためのコードは未実装です。

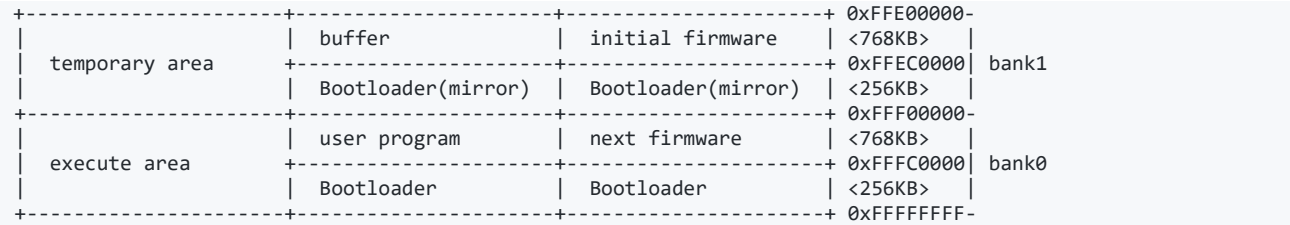
   → FAW ロックをかける位置

   バンクスワップすると FAW の設定範囲もスワップされるのではないかという疑問がありますが、スワップされません。下記図において FAW に 0xFFE00000-0xFFEC0000 を設定しバンクスワップしても、FAW は 0xFFE00000-0xFFEC0000 のままです。

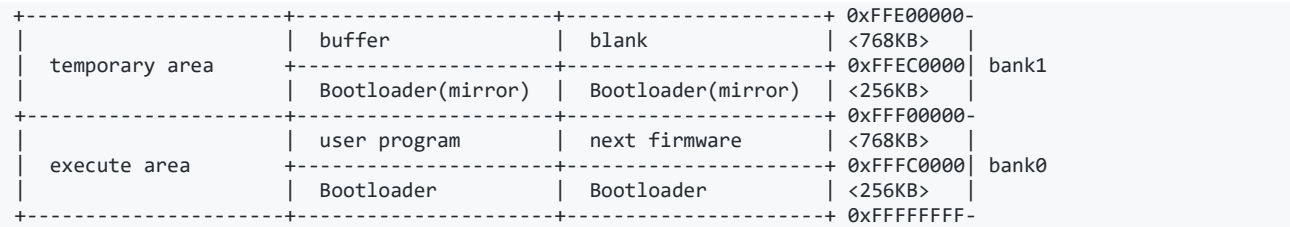
9. Bootloader(mirror)が initial firmware にジャンプする（initial firmware が起動する）
10. initial firmware が temporary area に next firmware を書き込む



11. initial firmware が temporary area の next firmware が正しいことを検証する
12. ソフトウェアリセットを発行し、バンクスワップする



13. bank0 に再配置された Bootloader が bank0 に再配置された next firmware が正しいことを検証し、bank1 に残っている initial firmware を消去する



14. Bootloader が次期ファームウェアにジャンプする（next firmware が起動する）
15. 繰り返し

7. ファームウェアアップデート用データ生成

一般的なファームウェアのデータフォーマットである MOT(モトローラ S レコードフォーマット)ファイルはデバイスに書き込むための実データ以外のデータ（署名値など）を保持する機構がありません。また MOT ファイルはテキストデータであるため、バイナリと比べてデータ量が 2 倍に増える傾向があります。ファームウェアアップデートではサーバからファームウェアが配信され、サーバへの課金は通信量に応じて増えるので、MOT ファイルはファームウェアアップデート用のデータフォーマットとして適切ではありません。

一方で、MOT ファイルのように汎用的なデータフォーマットで上記課題を解決できる適切なデータフォーマットはまだなく、今後デファクトスタンダードが定まってくるまでの間は、ルネサス独自方式である RSU(Renesas Secure Update)方式を規定しそれを使用することとします。

7.1 ダウンロードデータフォーマット

```
-----------------------------------------------------------------------------------------------------
output *.rsu
reference: https://docs.aws.amazon.com/ja_jp/freertos/latest/userguide/microchip-bootloader.html
-----------------------------------------------------------------------------------------------------
offset              component           contents name               length(byte)    OTA Image(Signed area)
0x00000000          Header              Magic Code                  7
0x00000007                              Image Flags                 1
0x00000008          Signature           Firmware Verification Type  32
0x00000028                              Signature size              4
0x0000002c                              Signature                   256
0x0000012c          Option              Dataflash Flag              4
0x00000130                              Dataflash Start Address     4
0x00000134                              Dataflash End Address       4
0x00000138                              Image Size                  4
0x0000013c                              Resereved(0x00)             196
0x00000200          Descriptor          Sequence Number             4               ---
0x00000204                              Start Address               4                |
0x00000208                              End Address                 4                |
0x0000020c                              Execution Address           4                |
0x00000210                              Hardware ID                 4                |
0x00000214                              Resereved(0x00)             236              |
0x00000300          Application Binary                              N               --- <- provided as mot
0x00000300 + N      Dataflash Binary                                M                   <- provided as mot
-----------------------------------------------------------------------------------------------------
Magic Code              : Renesas
Image Flags             : 0xff アプリケーションイメージが空であり、決して実行されません。
                          0xfe アプリケーションイメージがアップデート中でありテスト中であることを表します。
                          0xfc アプリケーションイメージが初期インストール中を表します。
                          0xf8 アプリケーションイメージが有効とマークされています。
                          0xf0 アプリケーションイメージは無効とマークされています。
                          0xe0 アプリケーションイメージは End of Life とマークされています。
Firmware Verification Type
                        : ファームウェア検証方式を指定するための識別子です。
                          例: sig-sha256-ecdsa
Signature/MAC/Hash size : ファームウェア検証に用いる署名値や MAC 値やハッシュ値などのデータサイズです。
Signature/MAC/Hash      : ファームウェア検証に用いる署名値や MAC 値やハッシュ値です。
Dataflash Flag          : 【RX MCU 特有】データフラッシュ用のデータが含まれるか否かを表すフラグです。
Dataflash Start Address : 【RX MCU 特有】データフラッシュの開始アドレスです。
Dataflash End Address   : 【RX MCU 特有】データフラッシュの終了アドレスです。
Image Size              : OTA Image のバイトサイズです。
Sequence Number         : シーケンス番号は、新しい OTA イメージを構築する前に増加させる必要があります。
                          Renesas Secure Flash Programmer にてユーザが指定可能です。
                          ブートローダーは、この番号を使用してブートするイメージを決定します。
                          有効な値の範囲は 1～ 4294967295 です。
Start Address           : デバイス上の OTA Image の開始アドレスです。
                          Renesas Secure Flash Programmer が自動的に設定するため、ユーザ指定は不要です。
End Address             : イメージトレーラーを除く、デバイス上の OTA Image の終了アドレスです。
                          Renesas Secure Flash Programmer が自動的に設定するため、ユーザ指定は不要です。
Hardware ID             : OTA Image が正しいプラットフォーム用に構築されているかどうかを検証するために
                          ブートローダーによって使用される一意のハードウェア ID です。
                          例: 0x00000001    MCUROM_RX65N_2M_SB_64KB
```

7.2 MOT ファイル変換ツール

一般的な MOT ファイルからルネサス独自形式の RSU ファイルに変換するためのツールを準備しました。具体的な使用方法については、以下関連ドキュメントを参照ください。

- RX ファミリに関する『RX ファミリにおけるファームウェアアップデートの実現方法 (R01ANxxxx)』

7.3 OpenSSL での ECDSA+SHA256 用の鍵ペア生成方法

MOT ファイル変換ツールには、ECDSA+SHA256 の公開鍵や秘密鍵を指定する必要があります。これらは OpenSSL を使用して生成することができます。以下に OpenSSL での ECDSA+SHA256 の公開鍵や秘密鍵の生成方法を記します。各種入力値は適宜ご自身のものに置き換えて入力してください。

7.3.1 CA 証明書の作成

```
$ openssl ecparam -genkey -name secp256r1 -out ca.key
using curve name prime256v1 instead of secp256r1
$ openssl req -x509 -sha256 -new  -nodes -key ca.key -days 3650 -out ca.crt
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:JP
State or Province Name (full name) [Some-State]:Tokyo
Locality Name (eg, city) []:Kodaira
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Renesas Electronics
Organizational Unit Name (eg, section) []:Software Development Division
Common Name (e.g. server FQDN or YOUR name) []:Ishiguro
Email Address []:hiroki.ishiguro.fv@renesas.com
```

7.3.2 楕円曲線暗号(パラメータは secp256r1)の鍵ペアを生成

```
$ openssl ecparam -genkey -name secp256r1 -out secp256r1.keypair
using curve name prime256v1 instead of secp256r1
```

7.3.3 鍵ペアの証明書を作成

```
$ openssl req -new -sha256 -key secp256r1.keypair > secp256r1.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:JP
State or Province Name (full name) [Some-State]:Tokyo
Locality Name (eg, city) []:Kodaira
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Renesas Electronics
Organizational Unit Name (eg, section) []:Software Development Division
Common Name (e.g. server FQDN or YOUR name) []:Ishiguro
Email Address []:hiroki.ishiguro.fv@renesas.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
```

```
An optional company name []:
```

7.3.4 CA 証明書を使用して鍵ペアの証明書を作成

```
$ openssl x509 -req -sha256 -days 3650 -in secp256r1.csr -CA ca.crt -CAkey ca.key -
CAcreateserial -out secp256r1.crt
Signature ok
subject=/C=JP/ST=Tokyo/L=Kodaira/O=Renesas Electronics/OU=Software Development
Division/CN=Ishiguro/emailAddress=hiroki.ishiguro.fv@renesas.com
Getting CA Private Key
```

7.3.5 楕円曲線暗号(パラメータは secp256r1)の秘密鍵を抽出

```
$ openssl ec -in secp256r1.keypair -outform PEM -out secp256r1.privatekey
read EC key
writing EC key
```

7.3.6 楕円曲線暗号(パラメータは secp256r1)の公開鍵を抽出

```
$ openssl ec -in secp256r1.keypair -outform PEM -pubout -out secp256r1.publickey
read EC key
writing EC key
```

7.4 OpenSSL でのテスト用の署名生成/検証方法

以下に OpenSSL での ECDSA+SHA256 の公開鍵や秘密鍵を使用した署名データの生成/検証方法を記します。

7.4.1 test.rsu に先に OpenSSL で作った秘密鍵で署名を付ける

```
$ openssl dgst -sha256 -sign secp256r1.privatekey test.rsu > signature.dat
```

7.4.2 OpenSSL 作った公開鍵で検証

```
$ openssl dgst -sha256 -verify secp256r1.publickey -signature signature.dat test.rsu
Verified OK
```

8. 量産時のメモリ保護に対する考慮

ここまでで説明した内容は、システム動作中の保護に対する考え方です。一般的には量産時にデバッガや ROM ライタ接続を禁止してセキュリティ強度を引き上げる考え方もあり、RX マイコンでは以下のようなメモリ保護機能があります。合わせて使用することでよりシステム全体のセキュリティ強度を上げることができます。

- https://www.renesas.com/jp/ja/doc/products/mpumcu/apn/rx/002/r01an0896jj0400-rx.pdf

ホームページとサポート窓口

ルネサス エレクトロニクス Web サイト

http://www.renesas.com/

お問い合わせ先

http://www.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019.01.21	−	初版
1.01	2019.10.08	-	サポートデバイス追加 RX130グループ、RX230グループ、RX231グループ、 RX66Tグループ、RX72Tグループ、RX72Mグループ
1.02	2020.01.30	-	サポートデバイス追加 RX23Wグループ

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、VIL（Max.）から VIH（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、VIL（Max.）から VIH（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

本社所在地

〒135-0061 東京都江東区豊洲3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサスエレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。