RH850 初期停止コアを搭載するデバイス、およびスタンバイモード中のデバイスにおけるデバッグ手法の紹介

R20AN0558JJ0100
Rev.1.00
2020.01.08

要旨

RH850 ファミリのデバイスには、初期停止コア（※）を搭載する製品や、デバイスの消費電力を抑制するスタンバイモードを搭載する製品があります。

本書では、初期停止コアを搭載するデバイスや、スタンバイモードに遷移するアプリケーションをデバッグするための手法を説明します。

※初期停止コア：リセット解除時に起動しない CPU コアのこと。初期停止コアが起動していない状態を初期停止状態と呼称します。

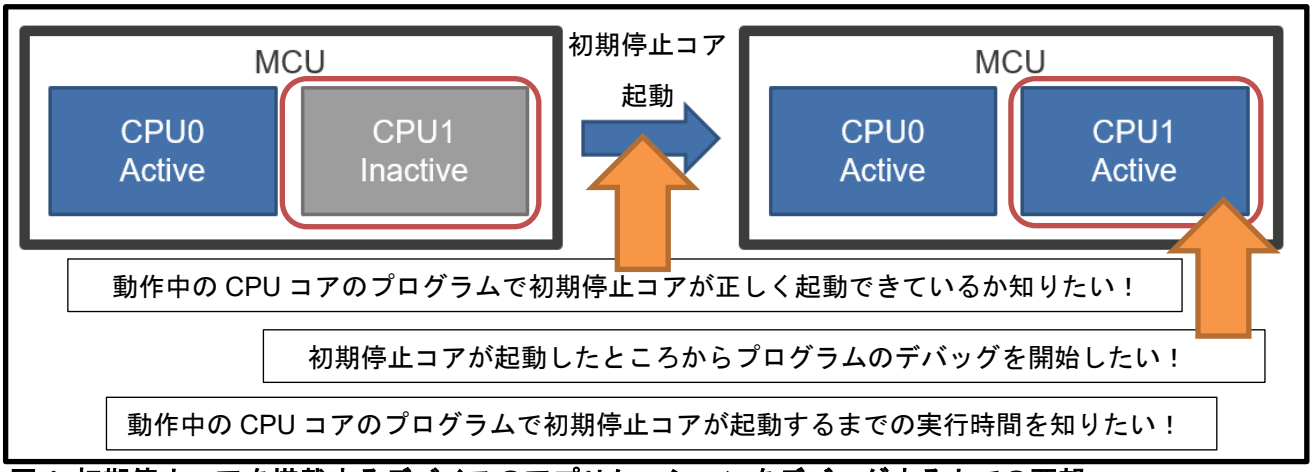

図 1 初期停止コアを搭載するデバイスのアプリケーションをデバッグする上での要望

図 2 スタンバイモードに遷移するアプリケーションをデバッグする上での要望

動作確認デバイス

RH850/F1KM-S1

RH850/F1KM-S4

RH850/F1KH-D8

RH850/E2x（スタンバイモード未搭載）

RH850/U2A

目次

用語説明

本書で使用する用語は、以下に示す定義で使用します。

統合開発環境（IDE）：

ルネサス製マイクロコンピュータに組み込むアプリケーションの開発を強力にサポートするツールです。ホストマシンからインタフェースを介してエミュレータを制御するエミュレータデバッガ機能を有しています。また、同一アプリケーション内でプロジェクトのエディットからビルドおよびデバッグまでを可能にし、バージョン管理をサポートしています。

CS+：

ルネサス製の統合開発環境です。

MULTI：

Green Hills Software 製の統合開発環境です。

エミュレータデバッガ：

統合開発環境から起動され、エミュレータを制御してデバッグを可能とするソフトウェアツール機能を指します。

ホストマシン：

エミュレータを制御するためのパーソナルコンピュータを指します。

ターゲットデバイス（MCU）：

デバッグ対象のデバイスを指します。

ユーザシステム：

デバッグ対象のデバイスを使用したお客様のアプリケーションシステムを指します。

ユーザインタフェース：

ターゲットデバイスと E1/E20/E2/IE850A エミュレータを接続するインタフェースを指します。

ユーザインタフェースケーブル：

ターゲットデバイスと E1/E20/E2/IE850A エミュレータを接続するケーブルを指します。

ICU-M

Intelligent Cryptographic Unit/Master の略称。RH850 ファミリのデバイスにはセキュリティ用 CPU コアとして ICU-M コアを搭載したデバイスがあります。ICU-M コアはオプションバイトの設定で有効/無効を切り替えられます。

マニュアル構成

本アプリケーションノートに関連するマニュアルは、以下の構成です。

・E1/E20 エミュレータユーザーズマニュアル,E2 エミュレータユーザーズマニュアル,
IE850A エミュレータユーザーズマニュアル

・E1/E20 エミュレータ,E2 エミュレータ,IE850A ユーザーズマニュアル別冊

・エミュレータデバッガのマニュアルおよびヘルプ

・RH850 初期停止コアを搭載するデバイス、およびスタンバイモード中のデバイスにおけるデバッグ手法の紹介 アプリケーションノート（本書）

(1) E1/E20 エミュレータ ユーザーズマニュアル,E2 エミュレータ ユーザーズマニュアル,
IE850A エミュレータユーザーズマニュアル

E1/E20 エミュレータ ユーザーズマニュアル、E2 エミュレータ ユーザーズマニュアル、IE850A エミュレータユーザーズマニュアルには、ハードウェア仕様を記載します。

・エミュレータの構成品

・エミュレータのハードウェア仕様

・エミュレータとホストマシンおよびユーザシステムとの接続

(2) E1/E20 エミュレータ,E2 エミュレータ,IE850A ユーザーズマニュアル別冊

E1/E20 エミュレータ、E2 エミュレータ、IE850A ユーザーズマニュアル別冊には、デバッガの機能説明および各デバイスに依存する内容、注意事項を記載します。

(3) エミュレータデバッガのマニュアルおよびヘルプ

エミュレータデバッガのマニュアルおよびヘルプには、E1/E20/E2/IE850A を使用する時のエミュレータデバッガの機能説明および操作方法を記載します。

(4) RH850 初期停止コアを搭載するデバイス、およびスタンバイモード中のデバイスにおけるデバッグ手法の紹介 アプリケーションノート（本書）

RH850 初期停止コアを搭載するデバイス、およびスタンバイモード中のデバイスにおけるデバッグ手法の紹介 アプリケーションノートには、初期停止コアを搭載する RH850 デバイスのデバッグや、スタンバイモードを搭載する RH850 ファミリのデバイスを同期デバッグするための手法を記載します。

1. 概要

RH850 ファミリのデバイスには、初期停止コアを搭載する製品や、スタンバイモードを搭載する製品があります。

初期停止コアはリセット解除時には起動せず、レジスタ操作によって起動する CPU コアです。CPU コアはオプションバイトによってリセット解除時に起動するか、初期停止コアにするかを設定することができます。詳細はデバイスのハードウェアマニュアルを参照してください。以降、初期停止コアはリセット解除時に初期停止状態になるものとして説明します。

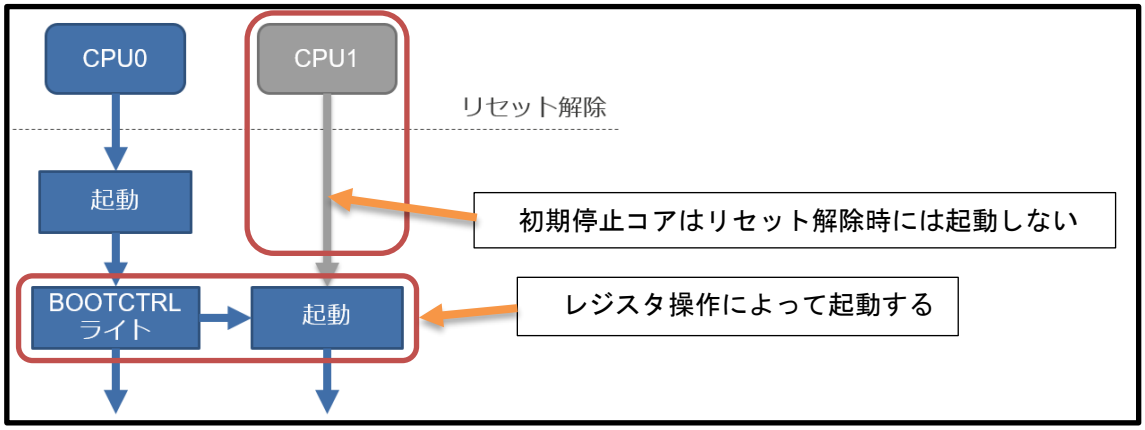

図 1-1 初期停止コアのリセット解除後の動作と起動シーケンス図

スタンバイモードは、デバイスの消費電力を抑制するための機能です。スタンバイモードには STOP モード、Deep STOP モード、Cyclic RUN モード、Cyclic STOP モードの 4 つのモードがあります。図 1-2 のように、レジスタ操作やウェイクアップ要因によってスタンバイモードに遷移します。詳細はデバイスのハードウェアマニュアルを参照してください。

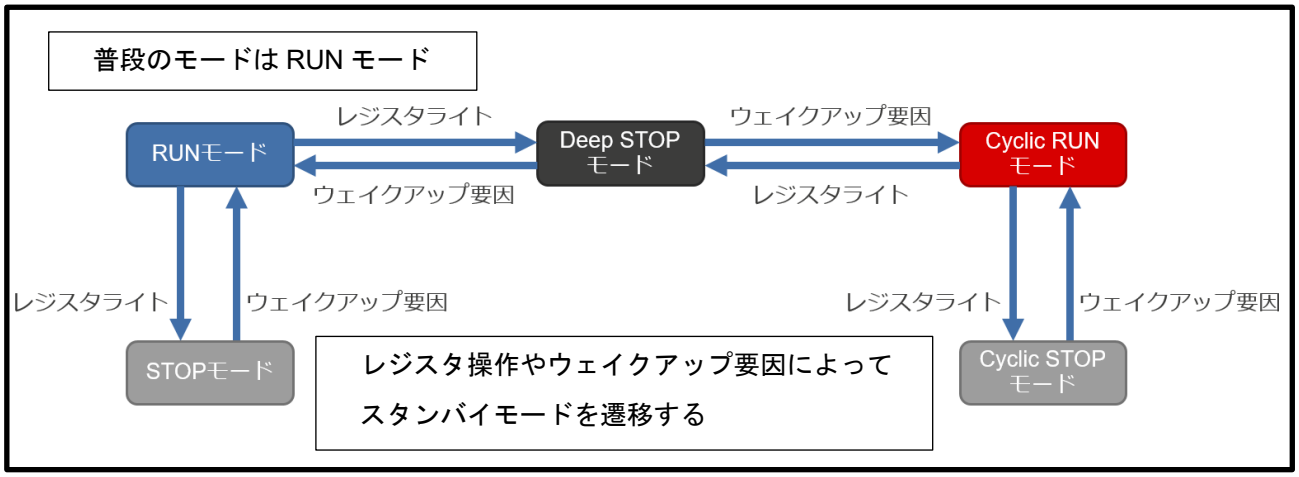

図 1-2 スタンバイモードの遷移図

1.1 デバッガの初期設定でのデバッグ仕様

デバッガの初期設定では、初期停止コアやスタンバイモードを搭載するデバイスで動作するアプリケーションをユーザが同期デバッグ（※）する場合に、デバッガがデバッグ開始時に初期停止コアを起動するデバッグ仕様や、ブレークを入れるタイミングに注意が必要といったデバッグ仕様が存在します。このため、初期停止コアが実際の動作と異なる動きになってしまい本来の動作をデバッグできない、スタンバイモード中の動作をデバッグできない状態になってしまいます。詳細は E1/E20 エミュレータ,E2 エミュレータ,IE850A ユーザーズマニュアル別冊を参照してください。

※同期デバッグ：プログラム実行やブレーク時に、すべての CPU コアを実行/ブレークさせるデバッグのこと。選択中の CPU コアだけ実行/ブレークさせるデバッグを非同期デバッグと呼称します。

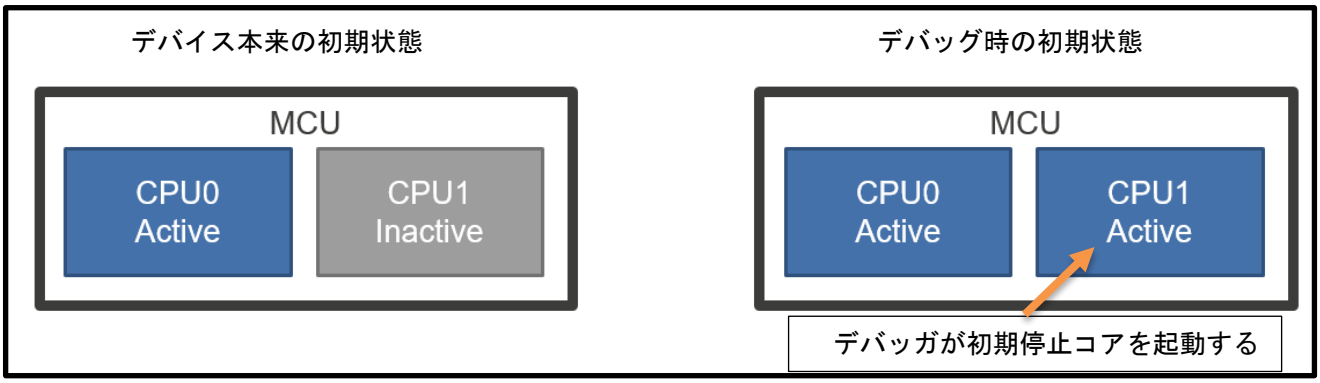

図 1-3 同期デバッグでのデバイスの初期状態の差異

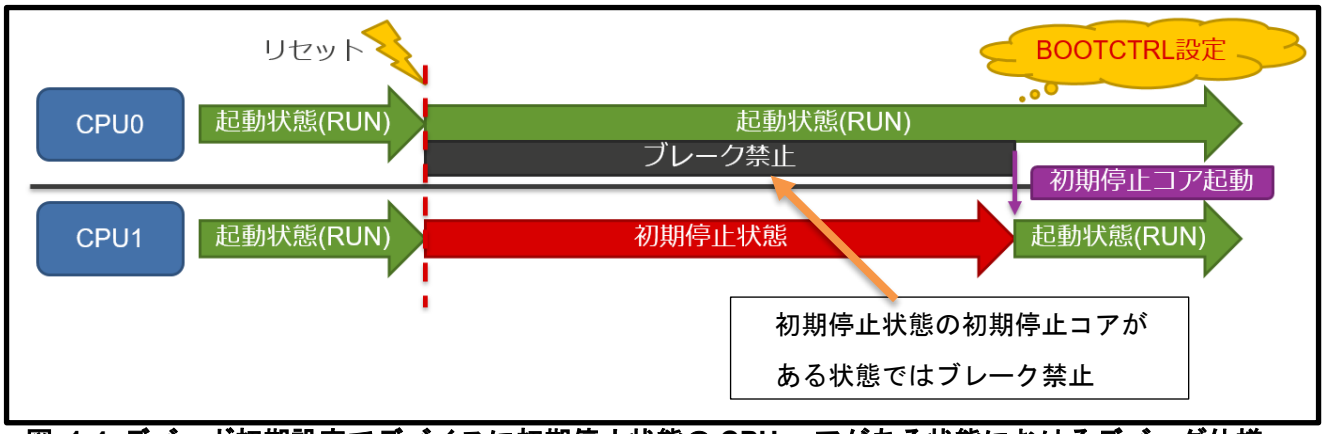

図 1-4 デバッガ初期設定でデバイスに初期停止状態の CPU コアがある状態におけるデバッグ仕様

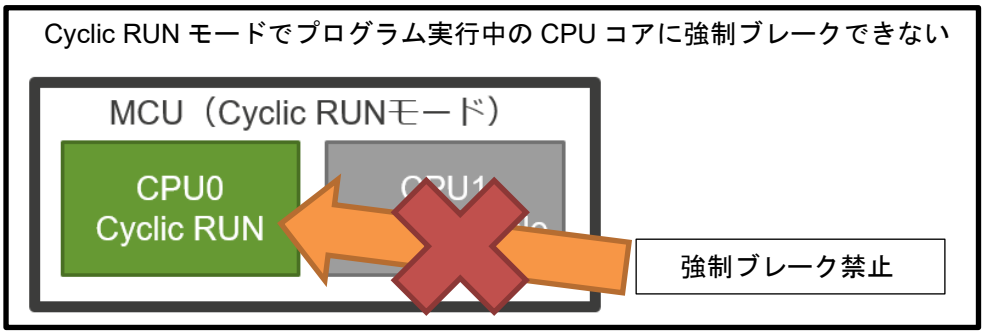

図 1-5 デバッガ初期設定での Cyclic RUN モードにおけるデバッグ仕様

1.2 初期停止コア・スタンバイモードをデバッグするには

デバッガに初期停止コア・スタンバイモードをデバッグする設定をすることで、初期停止コアの現在の状態を確認し、アプリケーションの実際の動作をデバッグできるようになります。また、現在のデバイスがどのスタンバイモードか確認し、スタンバイモード中のデバイスで動作するアプリケーションのデバッグができるようになります。

本書では、初期停止コアやスタンバイモードを搭載するデバイスで動作するアプリケーションをデバッグしたいユーザに対して、同期デバッグする手法を説明します。また、デバッグにおいて CPU コアの現在の状態を確認する手法や、デバイスの状態を遷移させるアプリケーション例およびプログラム例を用いてデバイスの状態が遷移した直後からデバッグを開始する手法、システムの時間制約要件が満たされているかを確認する手法、初期停止コアやスタンバイモードのデバッグでの注意事項を説明します。

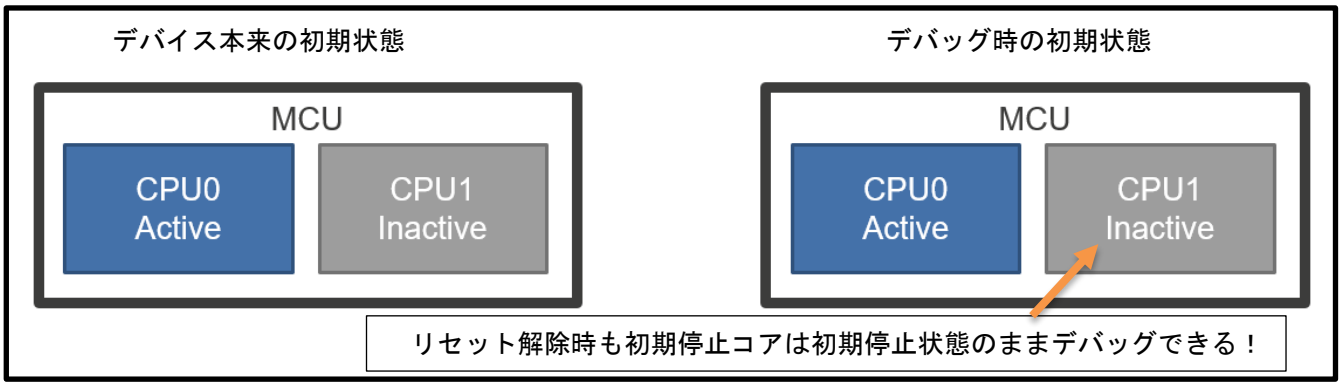

図 1-6 初期停止コア・スタンバイモードをデバッグする場合のデバイスの初期状態

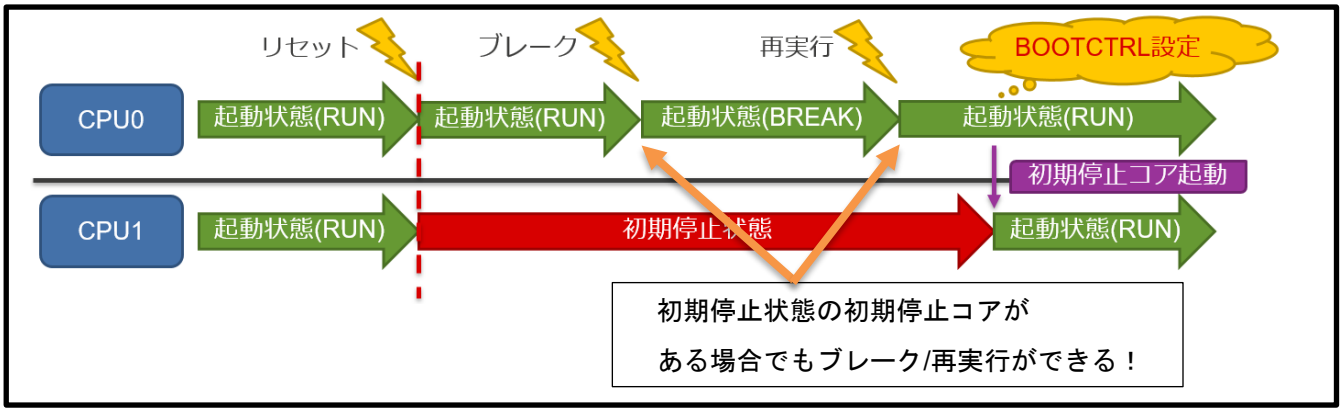

図 1-7 初期停止コア・スタンバイモードをデバッグする場合の初期停止状態の CPU コアがある状態でのデバッグ

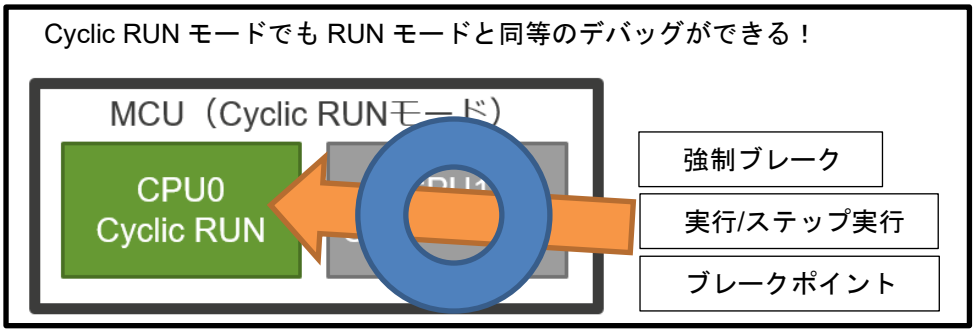

図 1-8 初期停止コア・スタンバイモードをデバッグする場合の Cyclic RUN モードデバッグ

2. セットアップ

初期停止コアを搭載するデバイスで動作するアプリケーション、およびスタンバイモード中のデバイスで動作するアプリケーションをデバッグするための環境のセットアップについて説明します。

2.1 システム構成と必要環境

本節では、システム構成と必要環境を説明します。

2.1.1 システム構成

図 2-1 および図 2-2 にシステム構成を示します。

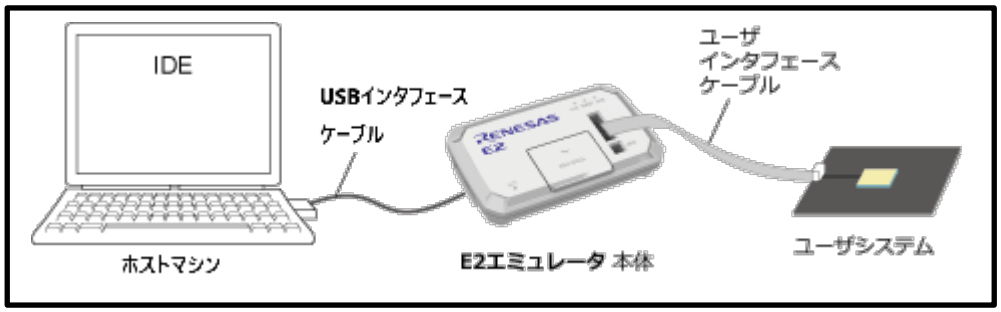

図 2-1 システム構成（E1/E20/E2 エミュレータ）

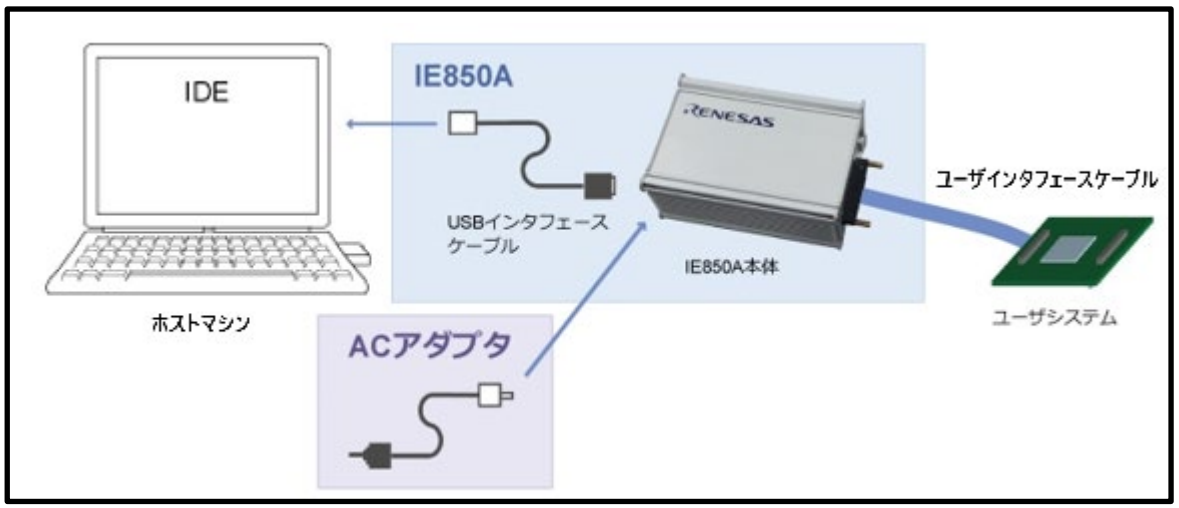

図 2-2 システム構成（IE850A エミュレータ）

2.1.2 必要環境

表 2-1 に必要環境を示します。

表 2-1 必要環境

項目	説明	
ターゲットデバイス 対応エミュレータ	[ターゲットデバイス] RH850/F1KM-S1 RH850/F1KM-S4 RH850/F1KH-D8	[対応エミュレータ] E1 エミュレータ E20 エミュレータ E2 エミュレータ
	[ターゲットデバイス] RH850/E2x RH850/U2A	[対応エミュレータ] E2 エミュレータ IE850A エミュレータ
統合開発環境 （バージョン）	Renesas CS+	V8.03.00 以降
	Green Hills Software MULTI	850eserv2 対応版 ※

※：Green Hills Software 様か販売代理店様にお問い合わせください

2.2 エミュレータとユーザシステムの電源投入

エミュレータとユーザシステムの起動方法について説明します。

① USB インターフェースケーブルの A プラグをホストマシンの USB インタフェースコネクタへ接続してください。

② USB インターフェースケーブルの mini-B プラグを E1/E20/E2/IE850A エミュレータの USB インタフェースコネクタへ接続してください。

③ E1/E20/E2 エミュレータの場合、ホストマシンと USB インターフェースケーブルを接続することで、エミュレータの電源が ON になります。

④ IE850A エミュレータの場合、AC アダプタを IE850A エミュレータに接続してください。電源の ON/OFF スイッチを ON にすることで、エミュレータの電源が ON になります。

⑤ ユーザシステムの電源を ON にしてください。

3. デバッグするための設定

初期停止コアやスタンバイモードを搭載するデバイスで、アプリケーションの実際の動作を同期デバッグする、またはスタンバイモードに遷移するアプリケーションを同期デバッグするための統合開発環境の設定について説明します。

この設定により、初期停止コアがリセット解除時も初期停止状態のままになり、実際の動作を同期デバッグすることができます。また、スタンバイモード中のデバイスで動作するアプリケーションを同期デバッグすることができます。

3.1 CS+での設定

CS+では、使用するデバッグ・ツールにエミュレータを選択し、デバッグ・ツールのプロパティにて接続用設定タブの「初期停止・スタンバイモードをデバッグする」を「はい」にしてください。

設定後、[デバッグ]メニューの[ビルド&デバッグ・ツールへダウンロード]を選択し、エミュレータデバッガの起動とダウンロードを行ってください。

図 3-1 CS+での初期停止・スタンバイモードのデバッグ設定

3.2 MULTI での設定

MULTI では、エミュレータデバッガの起動オプションに「-initstop」を指定してください。

この起動オプションで 850eserv2 と接続してください。

```
connect 850eserv2 -rh850 -e2lpd4=default -initstop -df=.¥DR7F702Z12.DVF ...
```

図 3-2 MULTI での初期停止・スタンバイモードのデバッグ設定

4. デバッグ手法

初期停止コアを搭載するデバイスで動作するアプリケーション、およびスタンバイモードに遷移するアプリケーションを同期デバッグする手法について説明します。

4.1 初期停止コアを搭載するデバイスで動作するアプリケーションのデバッグ手法

初期停止コアを搭載するデバイスで動作するアプリケーションを同期デバッグする手法について説明します。以下のアプリケーション例を用いてデバッグ操作例を示します。

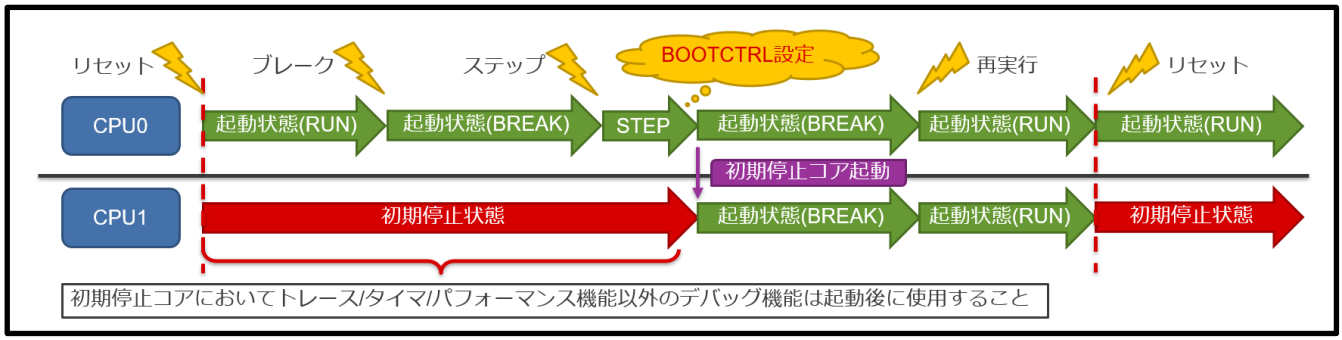

図 4-1 初期停止コアを搭載するデバイスで動作するアプリケーション例とデバッグ操作例

ユーザは初期停止コアの現在の状態（初期停止状態なのか、起動状態なのか）を確認することができます。また、初期停止状態から起動した直後からデバッグを開始することができます。さらに、初期停止コアを起動する処理までの時間を計測することで、時間制約要件を満たせているか確認することができます。

初期停止状態の初期停止コアで、トレースの取得や実行時間計測等がしたい場合は、プログラム実行前にトレース/タイマ/パフォーマンス関連機能を設定してください。それ以外のデバッグ機能に関しては、初期停止コアが初期停止状態から起動してから使用してください。

この例において、デバッグしたい内容と各節とのリンクを以下に示します。

・初期停止コアが初期停止状態であることを確認したい

→4.1.1 初期停止コアが初期停止状態であることを確認する手法 を参照してください。

・初期停止コアが起動していることを確認したい

・初期停止コアを起動させたところからデバッグしたい

→4.1.2 初期停止コアを起動する手法 を参照してください

・リセット解除後から初期停止コアが起動するまでの時間を確認したい

→4.1.3 初期停止コアが起動するまでの時間制約要件を満たせているか確認する手法 を参照してください

・リセットを入れて初期停止コアが初期停止状態になることを確認したい

→4.1.4 リセット時の初期停止コアを確認する手法 を参照してください。

4.1.1 初期停止コアが初期停止状態であることを確認する手法

初期停止コアが初期停止状態であることを確認する手法を説明します。以下に示す手法により、ユーザは初期停止コアが初期停止状態であることを確認することができます。

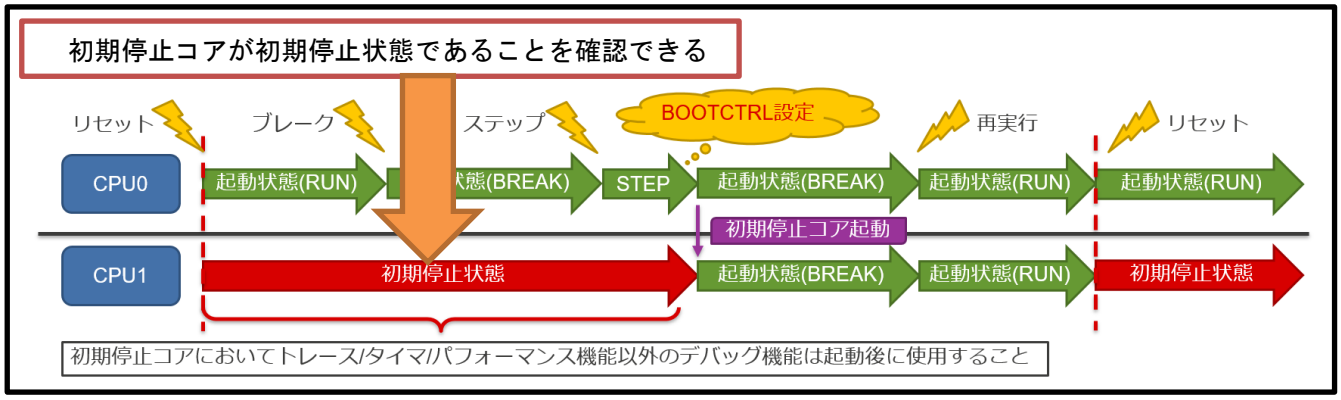

図 4-2 初期停止コアを搭載するデバイスで動作するアプリケーション例での初期停止状態確認

ステータスを取得することで初期停止コアが初期停止状態であることを確認できます。ステータスを取得した時の各デバッガでの表示を以下に示します。

●CS+

CS+では、初期停止コアが初期停止状態であることのステータスを、「Initial Stop」と表示します。

CS+では選択 CPU コアのステータスのみ表示します。すべての CPU コアのステータスを確認したい場合には、CPU コアを切り替えてステータスを確認してください。

図 4-3 CS+での初期停止コアが初期停止状態であることのステータス表示

●MULTI

MULTI では、初期停止コアが初期停止状態であることのステータスを、数字で(0x100)、文字列で「FETCH-STOP」と表示します

MULTI では cpustatus コマンドを発行することで、すべての CPU コアのステータスを確認できます。

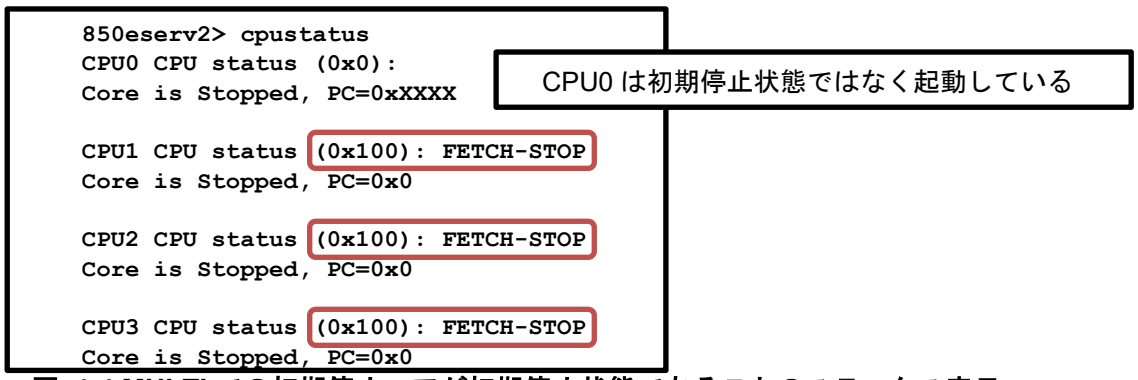

図 4-4 MULTI での初期停止コアが初期停止状態であることのステータス表示

4.1.2 初期停止コアを起動する手法

初期停止コアが起動した時のデバイスの状態を確認する手法、および初期停止コアが起動した直後からアプリケーションのデバッグを開始する手法を説明します。以下に示す手法により、ユーザは初期停止コアが起動状態であることを確認することができ、起動したところからデバッグを開始することができます。

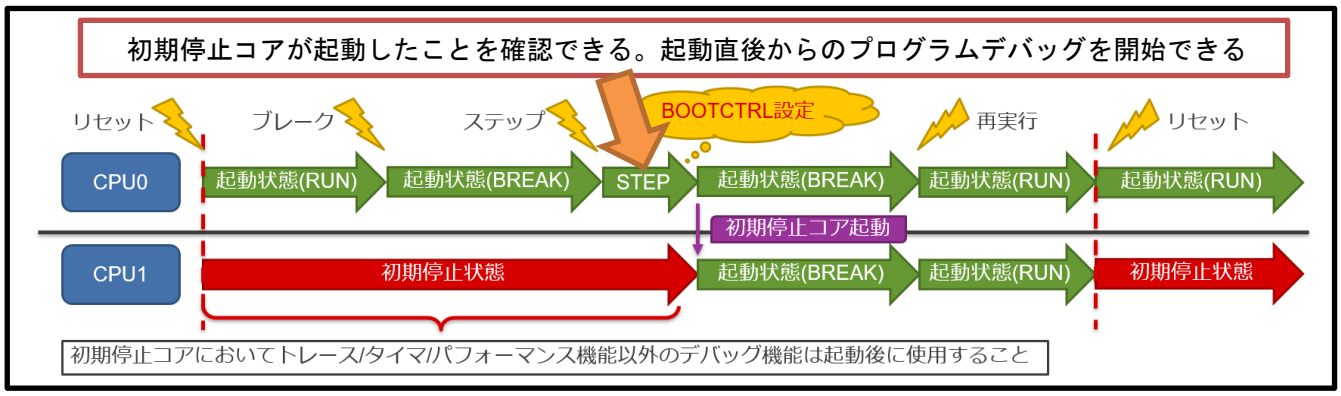

図 4-5 初期停止コアを搭載するデバイスで動作するアプリケーション例での初期停止コア起動

図 4-6 は初期停止コアを起動するプログラム例です。図 4-5 のアプリケーション例での CPU0 がこのプログラムを実行することにより、初期停止コアが起動します。レジスタの詳細およびプログラミング内容はデバイスのハードウェアマニュアルを参照してください。

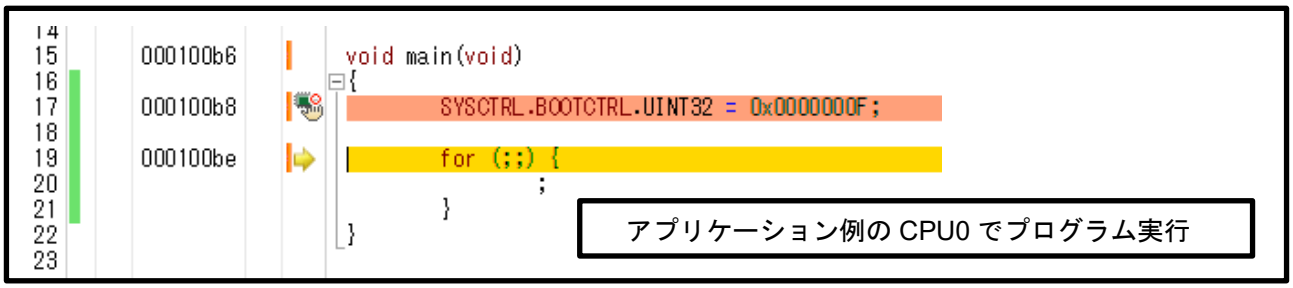

図 4-6 初期停止コアの起動例

初期停止コアが初期停止状態から起動したところからデバッグを開始したい場合には、CPU0 でこのプログラム例での BOOTCTRL レジスタ書き込み処理をステップ実行した後に、起動した初期停止コアに選択 CPU コアを切り替えることで実現できます。

ステータスを取得した時の各デバッガでの表示を以下に示します。

初期停止コアが起動した状態は通常の CPU コアと同等のステータス表示になります。

●CS+

CS+では、初期停止コアが起動し動作しているステータスの表示は通常の CPU コアと同等になります。

CS+では選択 CPU コアのステータスのみ表示します。すべての CPU コアのステータスを確認したい場合には、CPU コアを切り替えてステータスを確認してください。

図 4-7 CS+での初期停止コアが起動し動作しているステータス表示

●MULTI

MULTI では、初期停止コアが起動し動作しているステータスは通常の CPU コアと同等になります。

MULTI では cpustatus コマンドを発行することで、すべての CPU コアのステータスを確認できます。

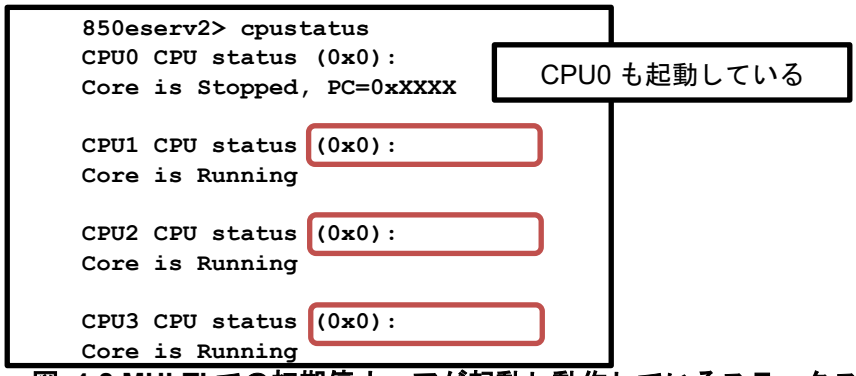

図 4-8 MULTI での初期停止コアが起動し動作しているステータス表示

4.1.3 初期停止コアが起動するまでの時間制約要件を満たせているか確認する手法

リセット解除後に CPU0 が起動しプログラムを実行し始めてから、CPU0 のプログラムにより初期停止コアが起動するまでの時間を計測する手法を説明します。以下に示す手法により、ユーザはデバイスが起動してから初期停止コアが起動するまでの時間制約要件が満たせているか確認することができます。

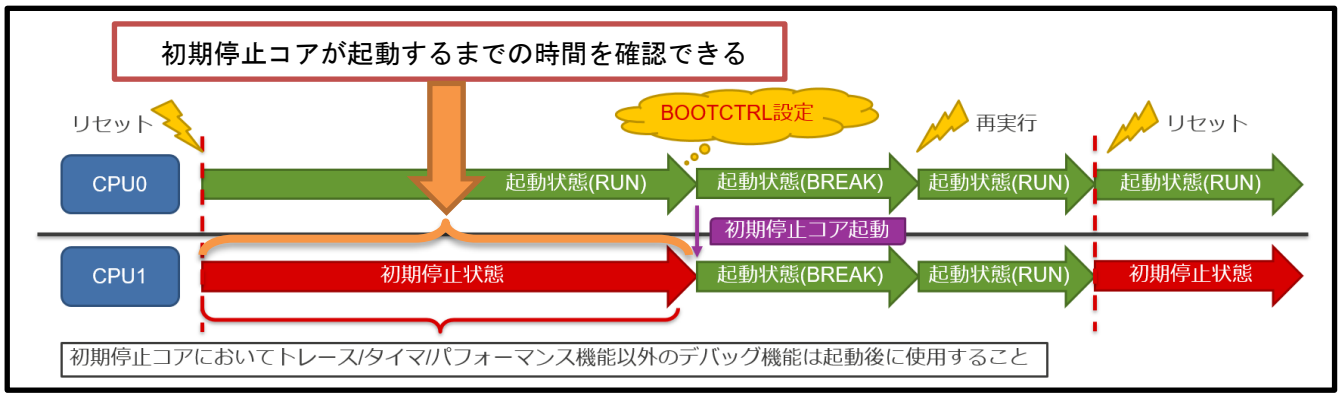

図 4-9 初期停止コアを搭載するデバイスで動作するアプリケーション例での起動までの時間

デバッガのタイマ機能を用いて、時間計測開始を CPU0 のリセットベクタアドレス、あるいはリセットブレーク状態においてプログラム実行開始時に設定、計測終了を BOOTCTRL レジスタ書き込み処理直後に設定します。その後プログラムを実行することで、CPU0 のプログラムがリセット解除から初期停止コアを起動するまでにかかった時間を計測することができます。

初期停止コアが起動するまでの時間が制約要件を満たしていなかった場合、デバッガのタイマ機能とトレース機能により、時間計測開始/終了を設定しての経過時間計測や、トレースによる各処理にかかった時間の表示を用いて、初期停止コアを起動するまでのプログラムにおける時間確認をすることができます。これにより、プログラムのどの処理にどれだけの時間がかかったのかを見ることができます。

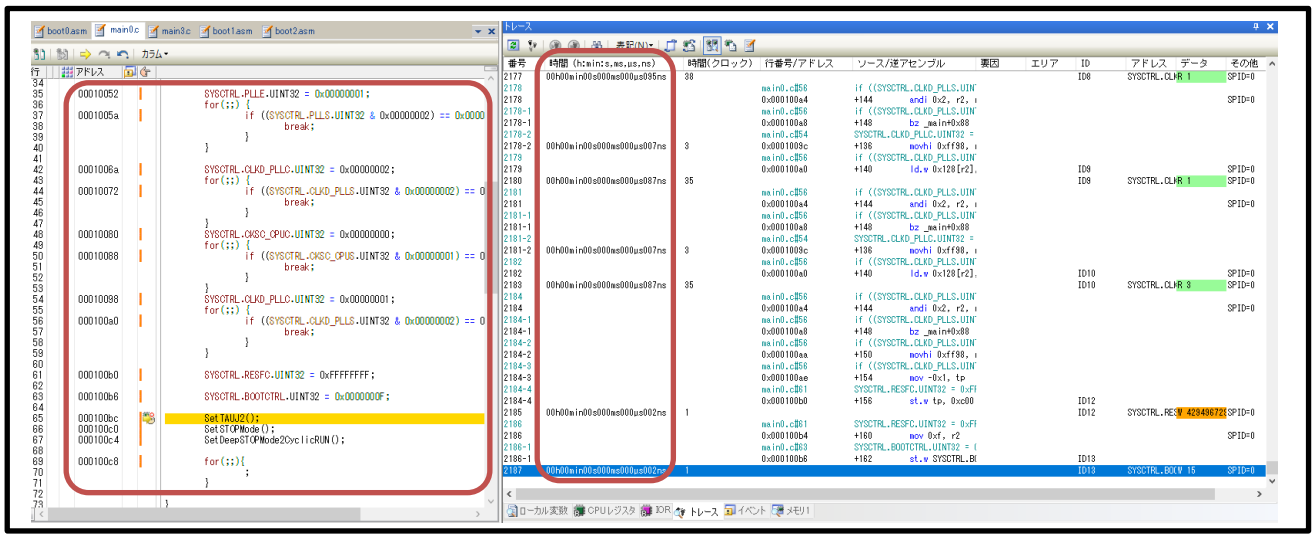

図 4-10 CS+での初期停止コアが起動するまでに実行したプログラムのトレース結果

4.1.4　リセット時の初期停止コアを確認する手法

初期停止コアが起動し、動作中にユーザやプログラムがリセットを入れた時には、初期停止コアが初期停止状態に遷移します。

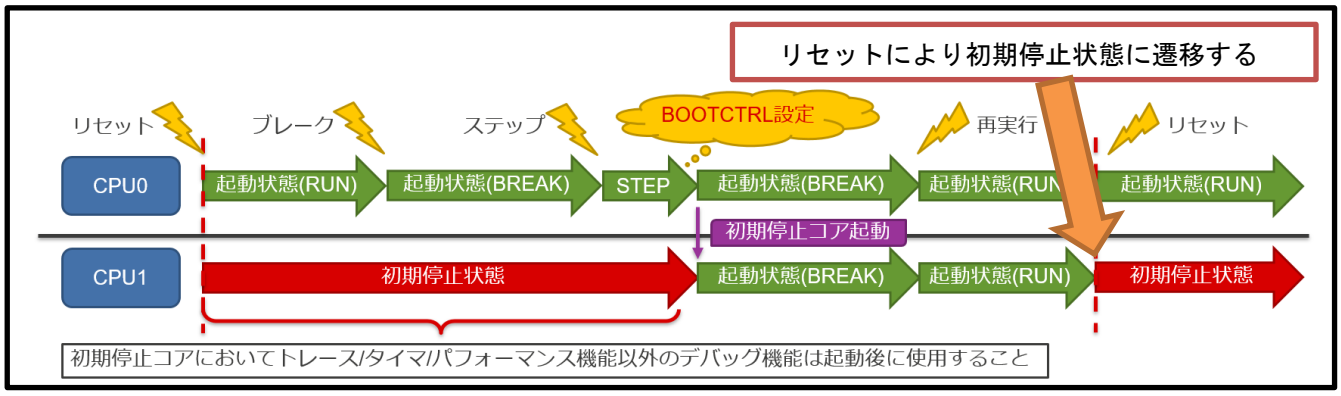

図 4-11 初期停止コアを搭載するデバイスで動作するアプリケーション例でのリセット後の状態

ステータス表示は 4.1.1 初期停止コアが初期停止状態であることを確認する手法 を参照してください。

4.2 スタンバイモードに遷移するアプリケーションのデバッグ手法

スタンバイモードを搭載するデバイスで動作するスタンバイモードに遷移するアプリケーションを同期デバッグする手法について説明します。以下のアプリケーション例を用いてデバッグ操作例を示します。

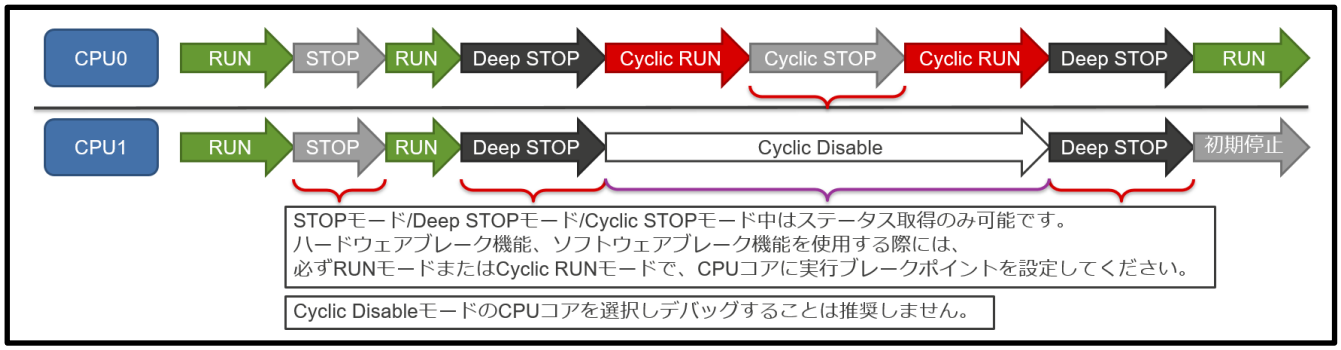

図 4-12 スタンバイモードに遷移するアプリケーション例とデバッグ操作例

ユーザは現在のスタンバイモードを確認することができます。また、STOP モード/Deep STOP モード/Cyclic STOP モードから RUN モード/Cyclic RUN モードに遷移した直後からのデバッグを開始することができます。さらに、スタンバイモードに遷移していた時間を計測することで、時間制約要件を満たせているか確認することができます。

デバイスが Cyclic RUN モードの時、ユーザは RUN モードと同等のデバッグができます。ただし、フラッシュメモリへのアクセスはできません。

デバイスが STOP モード/Deep STOP モード/Cyclic STOP モードの時、ユーザはステータス取得のみ可能です。それ以外のデバッグ機能は使用できません。ハードウェアブレーク機能、ソフトウェアブレーク機能を使用する際には、必ず RUN モードまたは Cyclic RUN モードで、CPU コアに実行ブレークポイントを設定してください。

Cyclic Disable モードの CPU コアは動作しておらず、起動するケースは Deep STOP モードに遷移し再び RUN モードに遷移した場合のみです。そのため、Cyclic Disable モードの CPU コアを選択しデバッグすることは推奨しません。

この例において、デバッグしたい内容と各節とのリンクを以下に示します。

・デバイスが STOP モードに遷移していることを確認したい

・STOP モードから RUN モードに遷移したところからデバッグしたい

・STOP モードである時間を確認したい

→4.2.1 STOP モード および

4.2.1.1 STOP モードから RUN モードに遷移した直後からデバッグする手法、

4.2.1.2 STOP モード中の時間制約要件を満たせているか確認する手法 を参照してください。

・デバイスが Deep STOP モードに遷移していることを確認したい

・Deep STOP モードから RUN モードに遷移したところからデバッグしたい

・Deep STOP モードから RUN モードに遷移するまでの Deep STOP モードである時間を確認したい

→4.2.2 Deep STOP モード および

4.2.2.1 Deep STOP モードから RUN モードに遷移した直後からデバッグする手法、

4.2.2.2 Deep STOP モードから RUN モードに遷移する時 Deep STOP モード中の時間制約要件を満たせているか確認する手法 を参照してください。

・デバイスが Cyclic RUN モードに遷移していることを確認したい

・Deep STOP モードから Cyclic RUN モードに遷移したところからデバッグしたい

・Deep STOP モードから Cyclic RUN モードに遷移までの Deep STOP モードである時間を確認したい

・Cyclic RUN モードである時間を確認したい

→4.2.3 Cyclic RUN モード および

4.2.3.1 Deep STOP モードから RUN モードに遷移した直後からデバッグする手法、

4.2.3.2 Deep STOP モードから Cyclic RUN モードに遷移する時の Deep STOP モード中の時間制約を満たされているか確認する手法、

4.2.3.3 Cyclic RUN モード中の時間制約要件を満たせているか確認する手法 を参照してください。

・デバイスが Cyclic STOP モードに遷移していることを確認したい

・Cyclic STOP モードから Cyclic RUN モードに遷移したところからデバッグしたい

・Cyclic STOP モードである時間を確認したい

→4.2.4 Cyclic STOP モード および

4.2.4.1 Deep STOP モードから Cyclic RUN モードに遷移した直後からデバッグする手法、

4.2.4.2 Cyclic STOP モード中の時間制約要件を満たせているか確認する手法 を参照してください。

4.2.1 STOP モード

デバイスを STOP モードに遷移させる手法、およびデバイスが STOP モードであることを確認する手法を説明します。以下に示す手法により、ユーザはデバイスが STOP モードであることを確認することができます。

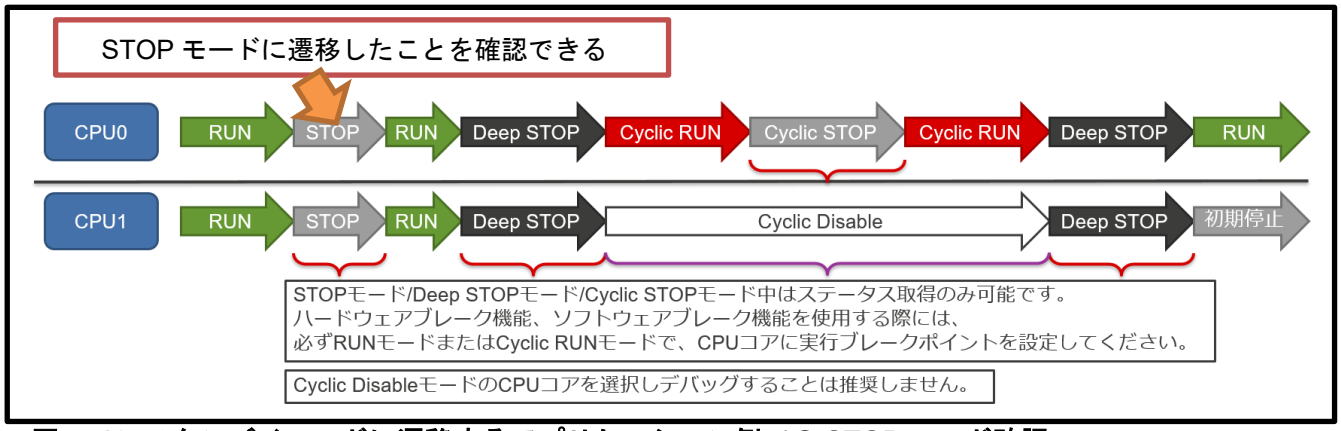

図 4-13 スタンバイモードに遷移するアプリケーション例での STOP モード確認

図 4-14 は STOP モードに遷移するプログラム例です。図 4-13 のアプリケーション例での CPU0 がこのプログラムを実行することにより、STOP モードに遷移します。レジスタの詳細およびプログラミング内容はデバイスのハードウェアマニュアルを参照してください。

```
109
110     00010156        SYSCTRL.STBCKCPROT.UINT32 = 0xA5A5A501;
111     0001015a        SYSCTRL.STBCOSTPT.UINT32 = 0x00000001;
112
113                     for(;;) {
114     0001015e                if (SYSCTRL.STBCOSTPT.UINT32 == 0x00000000) {
115                                     break;
116                             }
117                     }
```

アプリケーション例の CPU0 でプログラム実行

図 4-14 STOP モードに遷移するプログラム例

ステータスを取得することで STOP モードであることを確認できます。ステータスを取得した時の各デバッガでの表示を以下に示します。

●CS+

CS+では、STOP モードであることのステータスを、「Stop」と表示します。

図 4-15 CS+での STOP モードステータス表示

●MULTI

MULTI では、STOP モードであることのステータスを、数字で(0x8)、文字列で「HARDWARE STOP」と表示します。スタンバイモードはデバイスの状態であるため、CPU0 も STOP モードのステータスを表示します。

```
850eserv2> cpustatus
CPU0 CPU status (0x8): HARDWARE STOP
Core is Running

CPU1 CPU status (0x8): HARDWARE STOP
Core is Running

CPU2 CPU status (0x8): HARDWARE STOP
Core is Running

CPU3 CPU status (0x8): HARDWARE STOP
Core is Running
```

図 4-16 MULTI での STOP モードステータス表示

4.2.1.1 STOP モードから RUN モードに遷移した直後からデバッグする手法

STOP モードに遷移後、ウェイクアップ要因によって RUN モードに遷移した直後に CPU コアをブレークさせる手法を説明します。以下に示す手法により、ユーザは STOP モードから RUN モードに遷移した直後でのデバイスの状態を確認でき、RUN モードに遷移した直後からアプリケーションのデバッグを開始できます。

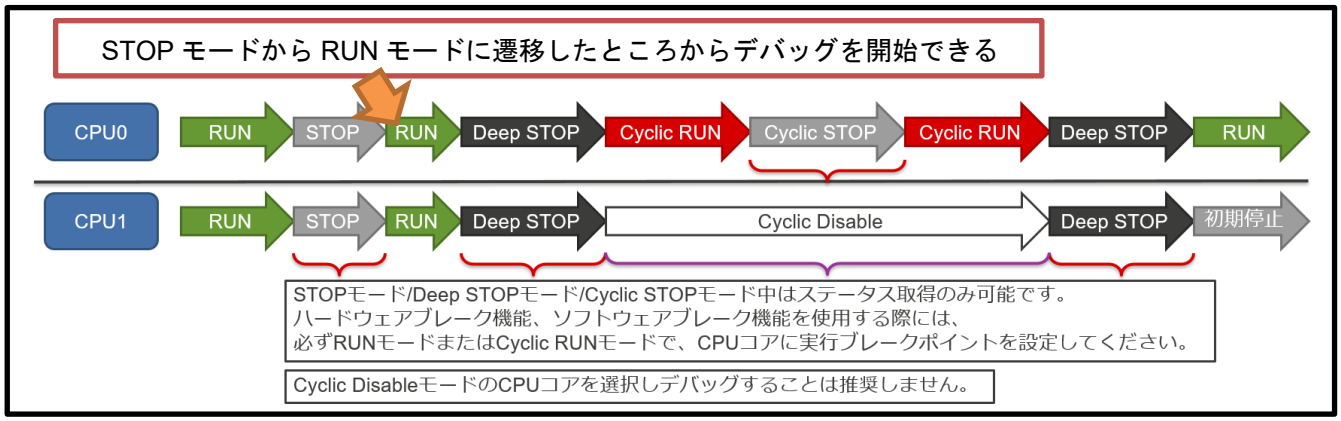

図 4-17 スタンバイモードに遷移するアプリケーション例での STOP モードから RUN モードへの遷移

図 4-18 は CPU コアが STOP モードから RUN モードに遷移したところでブレークしているプログラムデバッグ例です。CPU コアが STOP モードから RUN モードに遷移した時には、STBC0STPT レジスタ確認処理の後からプログラムを実行します。CPU コアが RUN モードに遷移したところからデバッグするためには、STOP モードに遷移する前に、あらかじめ STBC0STPT レジスタ確認処理の後に実行ブレークポイントを設定しておきます。これにより、CPU コアが RUN モードに遷移したところでブレークすることができます。

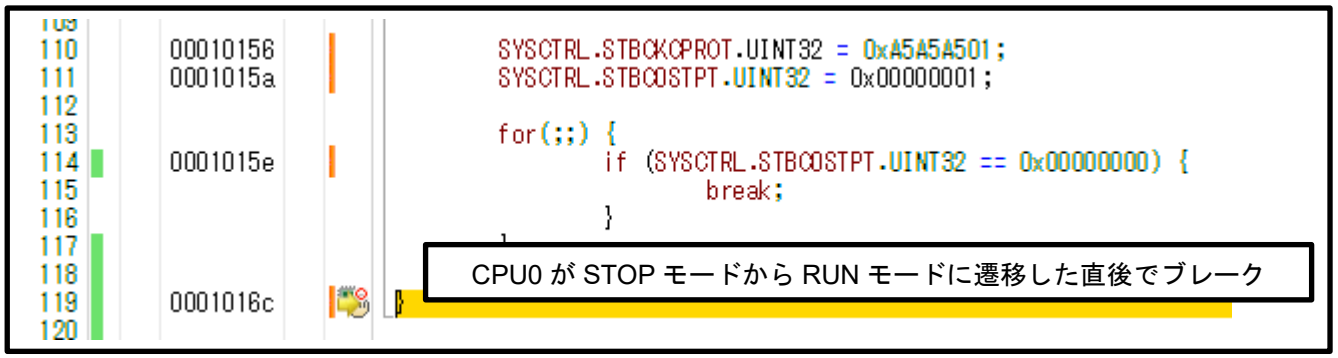

図 4-18 STOP モードから RUN モードに遷移時の実行ブレーク例

4.2.1.2 STOP モード中の時間制約要件を満たせているか確認する手法

STOP モードから RUN モードに遷移するまでの時間を計測する手法を説明します。以下に示す手法により、ユーザは STOP モードに遷移している時間の制約要件が満たせているか確認することができます。

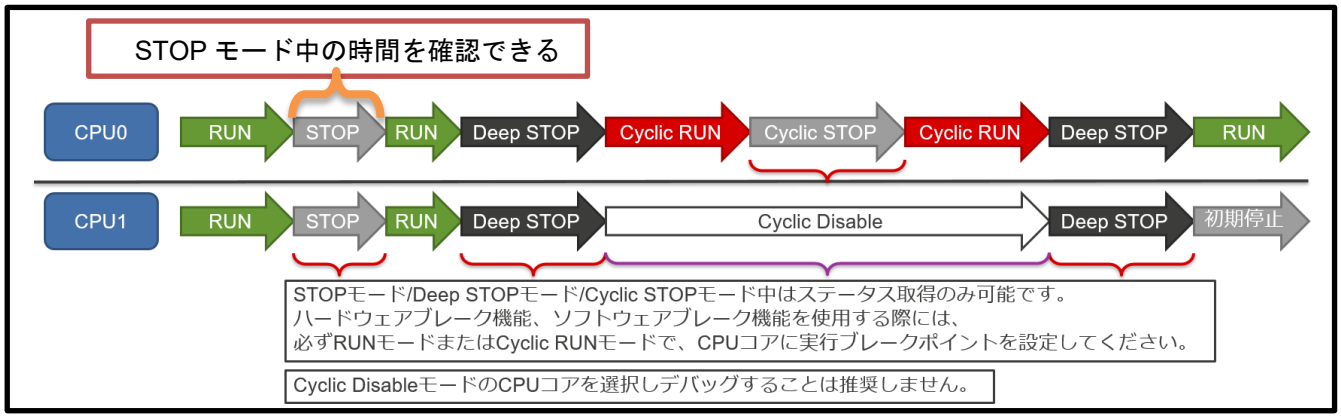

図 4-19 スタンバイモードに遷移するアプリケーション例での STOP モード中の時間確認

デバッガのタイマ機能を用いて、時間計測開始を STOP モードに遷移するプログラムアドレスに設定し、計測終了を STOP モードから RUN モードに遷移した直後のプログラムアドレスに設定します。その後プログラムを実行することで、CPU0 のプログラムで STOP モードに遷移してから、RUN モードに遷移するまでにかかった時間を計測することができます。

4.2.2 Deep STOP モード

デバイスを Deep STOP モードに遷移させる手法、およびデバイスが Deep STOP モードであることを確認する手法を説明します。以下に示す手法により、ユーザはデバイスが Deep STOP モードであることを確認することができます。

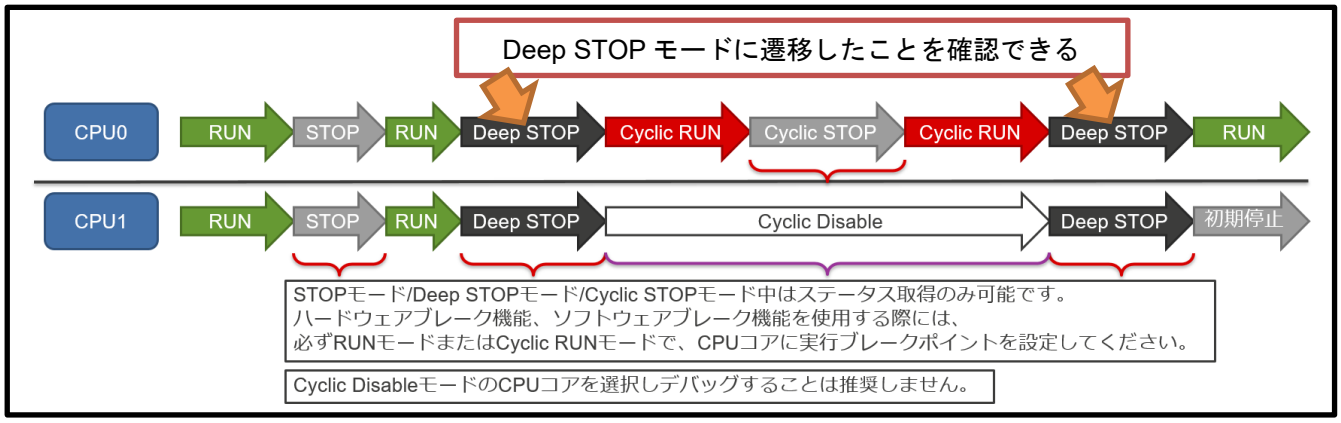

図 4-20 スタンバイモードに遷移するアプリケーション例での Deep STOP モード確認

図 4-21 は Deep STOP モードに遷移するプログラム例です。図 4-20 のアプリケーション例での CPU0 がこのプログラムを実行することにより、Deep STOP モードに遷移します。レジスタの詳細およびプログラミング内容はデバイスのハードウェアマニュアルを参照してください。

```
90      000101cc        SYSCTRL.STBCKCPROT.UINT32 = 0xA5A5A501;
91      000101d2        SYSCTRL.STBCOPSC.UINT32 = 0x00000002;
92
93      000101d6        for(;;){
94                              ;
95                      }
```

アプリケーション例の CPU0 でプログラム実行

図 4-21 Deep STOP モードに遷移するプログラム例

ステータスを取得することで Deep STOP モードであることを確認できます。ステータスを取得した時の各デバッガでの表示を以下に示します。

●CS+

CS+では、Deep STOP モードであることのステータスを、「Deep Stop」と表示します。

図 4-22 CS+での Deep STOP モードステータス表示

●MULTI

MULTI では、Deep STOP モードであることのステータスを、数字で(0x200)、文字列で「DEEP-STOP」と表示します。スタンバイモードはデバイスの状態であるため、CPU0 も Deep STOP モードのステータスを表示します。

```
850eserv2> cpustatus
CPU0 CPU status (0x200): DEEP-STOP
Core is Running

CPU1 CPU status (0x200): DEEP-STOP
Core is Running

CPU2 CPU status (0x200): DEEP-STOP
Core is Running

CPU3 CPU status (0x200): DEEP-STOP
Core is Running
```

図 4-23 MULTI での Deep STOP モードステータス表示

4.2.2.1 Deep STOP モードから RUN モードに遷移した直後からデバッグする手法

Deep STOP モードに遷移後、ウェイクアップ要因によって RUN モードに遷移した直後に CPU コアをブレークさせる手法を説明します。以下に示す手法により、ユーザは Deep STOP モードから RUN モードに遷移した直後でのデバイスの状態を確認でき、RUN モードに遷移した直後からアプリケーションのデバッグを開始できます。

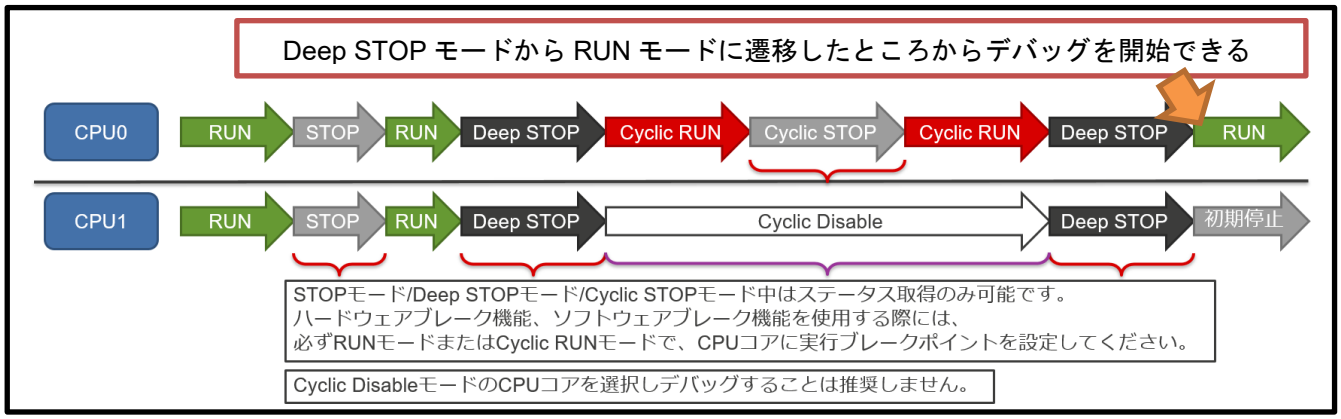

図 4-24 スタンバイモードに遷移するアプリケーション例での Deep STOP モードから RUN モードへの遷移

図 4-25 は CPU コアが Deep STOP モードから RUN モードに遷移したところでブレークしているプログラムデバッグ例です。CPU コアが Deep STOP モードから RUN モードに遷移した時には、Deep STOP リセットが入り、リセットベクタアドレスからプログラムを実行します。リセットベクタアドレスは RBASE に依存します。プログラムデバッグ例では RBASE は 0x00000000 を設定しています。CPU コアが RUN モードに遷移したところからデバッグするためには、Deep STOP モードに遷移する前に、あらかじめリセットベクタアドレスに実行ブレークポイントを設定しておきます。これにより、CPU コアが RUN モードに遷移したところでブレークすることができます。

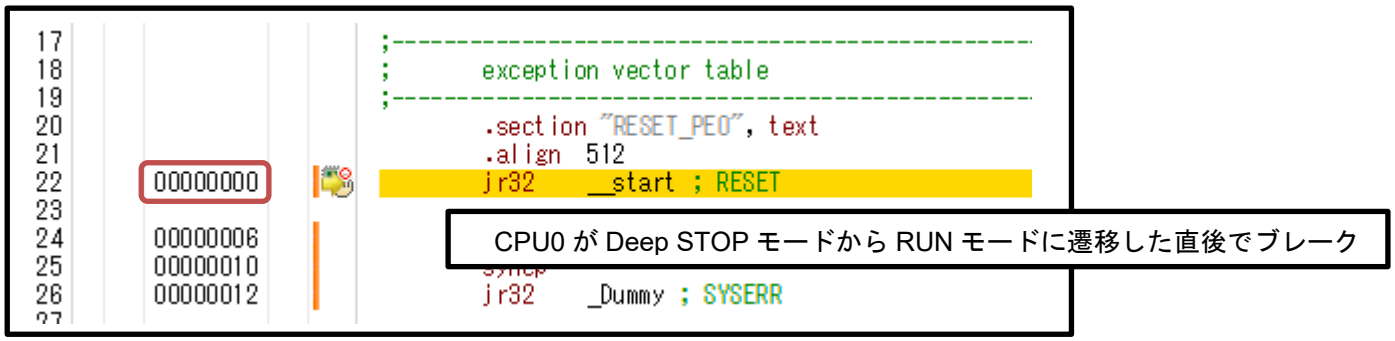

図 4-25 Deep STOP モードから RUN モードに遷移時の実行ブレーク例

Deep STOP モードから RUN モードに遷移する時に発生する Deep STOP リセットにより、初期停止コアは初期停止状態になります。初期停止コアのデバッグについては 4.1 初期停止コアを搭載するデバイスで動作するアプリケーションのデバッグを参照してください。

4.2.2.2 Deep STOP モードから RUN モードに遷移する時 Deep STOP モード中の時間制約要件を満たせているか確認する手法

Deep STOP モードから RUN モードに遷移するまでの時間を計測する手法を説明します。以下に示す手法により、ユーザは Deep STOP モードから RUN モードに遷移するまでの Deep STOP モードである時間の制約要件が満たせているか確認することができます。

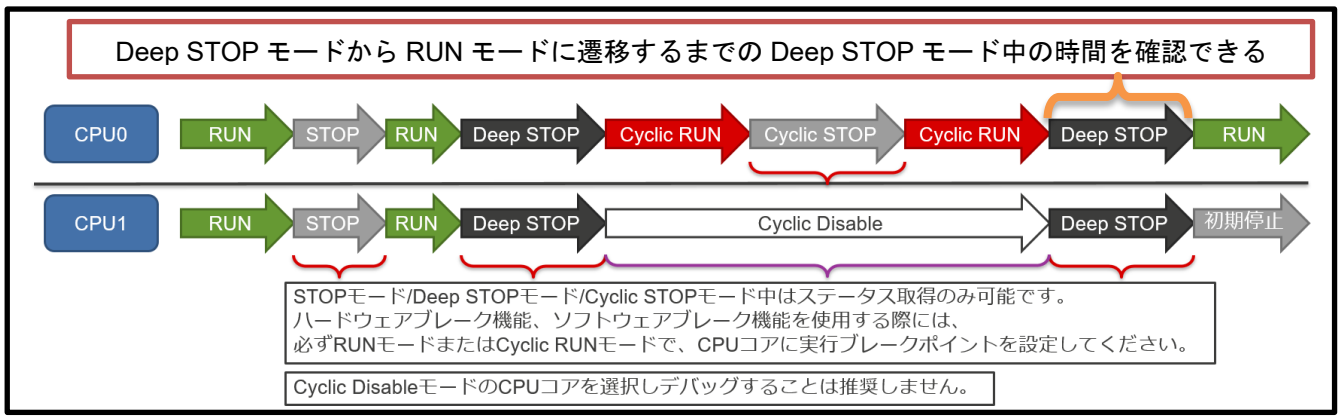

図 4-26 スタンバイモードに遷移するアプリケーション例での Deep STOP モードから RUN モードに遷移するまでの Deep STOP モード中の時間確認

デバッガのタイマ機能を用いて、時間計測開始を Deep STOP モードに遷移するプログラムアドレスに設定し、計測終了を Deep STOP モードから RUN モードに遷移した直後のリセットベクタアドレスに設定します。その後プログラム実行することで、CPU0 のプログラムで Deep STOP モードに遷移してから、RUN モードに遷移するまでにかかった時間を計測することができます。

4.2.3 Cyclic RUN モード

デバイスを Cyclic RUN モードに遷移させる手法、およびデバイスが Cyclic RUN モードであることを確認する手法を説明します。以下に示す手法により、ユーザはデバイスが Cyclic RUN モードであることを確認することができ、RUN モード時と同等のデバッグができます。ただし、フラッシュメモリへのアクセスはできません。

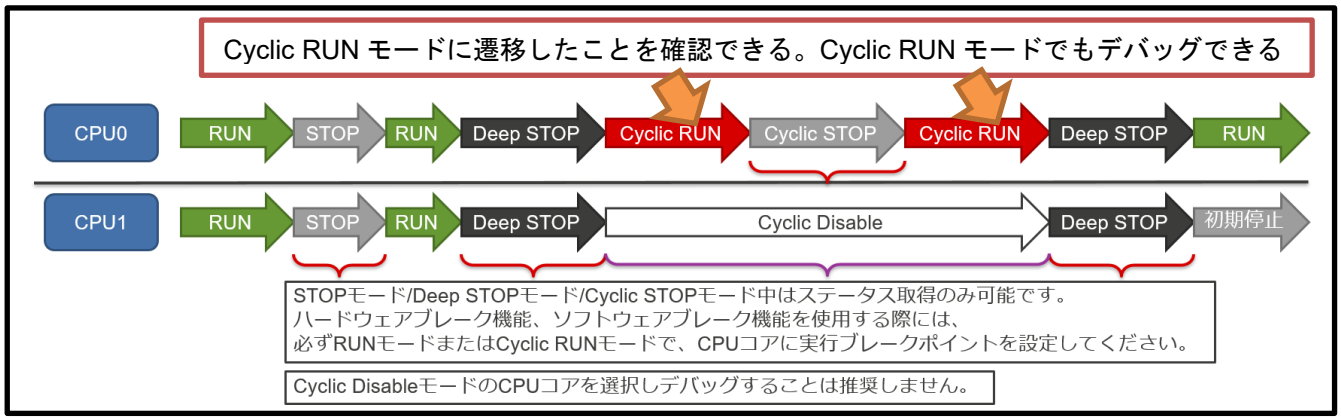

図 4-27 スタンバイモードに遷移するアプリケーション例での Cyclic RUN モード確認

図 4-28 は Cyclic RUN モードに遷移するプログラム例です。このプログラム例ではウェイクアップ要因に TAUJ2 の割り込みを指定しています。図 4-27 のアプリケーション例での CPU0 がこのプログラムを実行することにより、Deep STOP モードに遷移し、その後 TAUJ2 の割り込みをウェイクアップ要因として起動し、Cyclic RUN モードに遷移します。レジスタの詳細およびプログラミング内容はデバイスのハードウェアマニュアルを参照してください。

Cyclic RUN モードに遷移した時、RH850/F1KM-S1 シリーズ, F1KM-S4 シリーズ, F1KH-D8 シリーズは CPU1、それ以外の RH850 ファミリのデバイスでは CPU0 のみが起動します。起動した CPU コアは Retention RAM の先頭から実行します。Cyclic RUN モードで動作させる場合、あらかじめ RUN モードにて Retention RAM にプログラムをダウンロードしておいてください。Retention RAM へのダウンロードに関する詳細はエミュレータデバッガのマニュアルおよびヘルプを参照してください。

```
124  0001016e    SYSCTRL.WUFC0_A2 = 0xFFFFFFFF;
125  0001017a    SYSCTRL.WUFMSK0_A2 = 0xFFFFFFFF;
126  0001017e    SYSCTRL.WUFC1_A2 = 0xFFFFFFFF;
127  00010184    SYSCTRL.WUFMSK1_A2 = 0xFFFFFFF3;
128
129  0001018e    SYSCTRL.CLKKCPROT1.UINT32 = 0xA5A5A501;
130  00010194    SYSCTRL.HSOSCSTPM.UINT32 = 0x00000001;
131
132  00010198    SYSCTRL.STBCKCPROT.UINT32 = 0xA5A5A501;
133  0001019e    SYSCTRL.STBCOPSC.UINT32 = 0x00000002;
134
135  000101a2    for(;;){
136                  ;
137              }
```

アプリケーション例の CPU0 でプログラム実行
Deep STOP モード遷移後、TAUJ2 の割り込みで
Cyclic RUN モードに遷移

図 4-28 Cyclic RUN モードに遷移するプログラム例

ステータスを取得することで Cyclic RUN モードであることを確認できます。ステータスを取得した時の各デバッガでの表示を以下に示します。

●CS+

CS+では、Cyclic RUN モードであることのステータスを、「Cyclic RUN」と表示します。

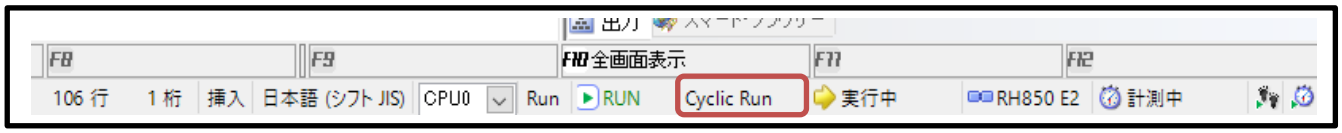

図 4-29 CS+での Cyclic RUN モードステータス表示

●MULTI

MULTI では、Cyclic RUN モードであることのステータスを、数字で(0x400)、文字列で「CYCLE-RUN」と表示します。スタンバイモードはデバイスの状態であるため、CPU0 に Cyclic RUN モードのステータスを表示します。

```
850eserv2> cpustatus
CPU0 CPU status (0x400): CYCLE-RUN
Core is Running

CPU1 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU2 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU3 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running
```

図 4-30 MULTI での Cyclic RUN モードステータス表示

4.2.3.1 Deep STOP モードから Cyclic RUN モードに遷移した直後からデバッグする手法

Deep STOP モードに遷移後、ウェイクアップ要因によって Cyclic RUN モードに遷移した直後に CPU コアをブレークさせる手法を説明します。以下に示す手法により、ユーザは Deep STOP モードから Cyclic RUN モードに遷移した直後でのデバイスの状態を確認でき、Cyclic RUN モードに遷移した直後からアプリケーションのデバッグを開始できます。

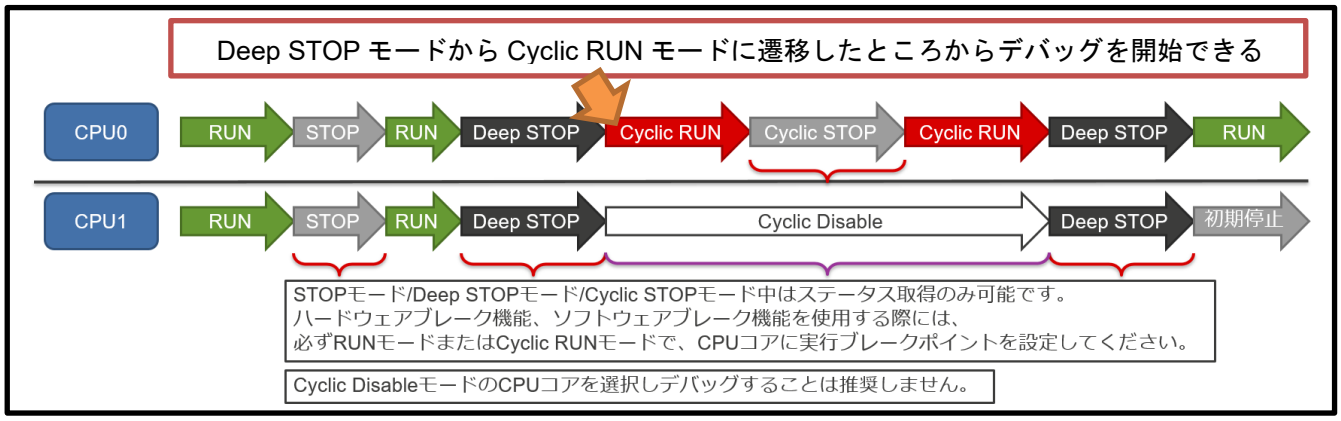

図 4-31 スタンバイモードに遷移するアプリケーション例での Cyclic RUN モード遷移

図 4-32 は CPU コアが Deep STOP モードから Cyclic RUN モードに遷移したところでブレークしているプログラムデバッグ例です。CPU コアが Deep STOP モードから Cyclic RUN モードに遷移した時には、Retention RAM の先頭アドレスからプログラムを実行します。CPU コアが Cyclic RUN モードに遷移したところからデバッグするためには、Deep STOP モードに遷移する前に、あらかじめ Retention RAM の先頭アドレスに実行ブレークポイントを設定しておきます。これにより、CPU コアが Cyclic RUN モードに遷移したところでブレークすることができます。

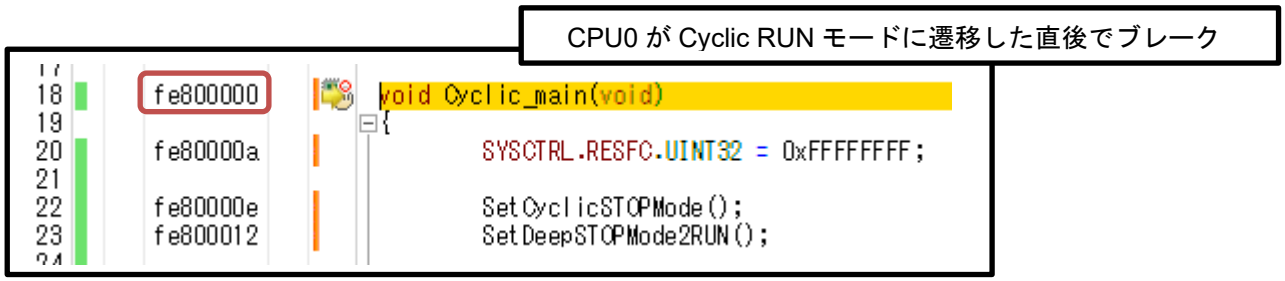

図 4-32 Deep STOP モードから Cyclic RUN モードに遷移時の実行ブレーク例

ステータスを取得した時、Cyclic RUN モードであることと共に、ブレークしていることも表示します。

●CS+

CS+では、「RUN」ステータスが「BREAK」ステータスに変化します。「Cyclic RUN」はそのまま表示します。PC 値は Retention RAM の先頭アドレスになります。

図 4-33 CS+での Cyclic RUN モードでブレーク状態に変化したステータス表示

●MULTI

MULTI では、「Core is Running」ステータスが「Core is Stopped」ステータスに変化します。Cyclic RUN モードであることのステータスはそのまま表示します。PC 値は Retention RAM の先頭アドレスになります。

```
850eserv2> cpustatus
CPU0 CPU status (0x400): CYCLE-RUN
Core is Stopped, PC=0xfe800000

CPU1 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU2 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU3 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running
```

図 4-34 MULTI での Cyclic RUN モードでブレーク状態に変化したステータス表示

4.2.3.2 Deep STOP モードから Cyclic RUN モードに遷移する時の Deep STOP モード中の時間制約を満たされているか確認する手法

Deep STOP モードから Cyclic RUN モードに遷移するまでの時間を計測する手法を説明します。以下に示す手法により、ユーザは Deep STOP モードから Cyclic RUN モードに遷移するまでの Deep STOP モードである時間の制約要件が満たせているか確認することができます。

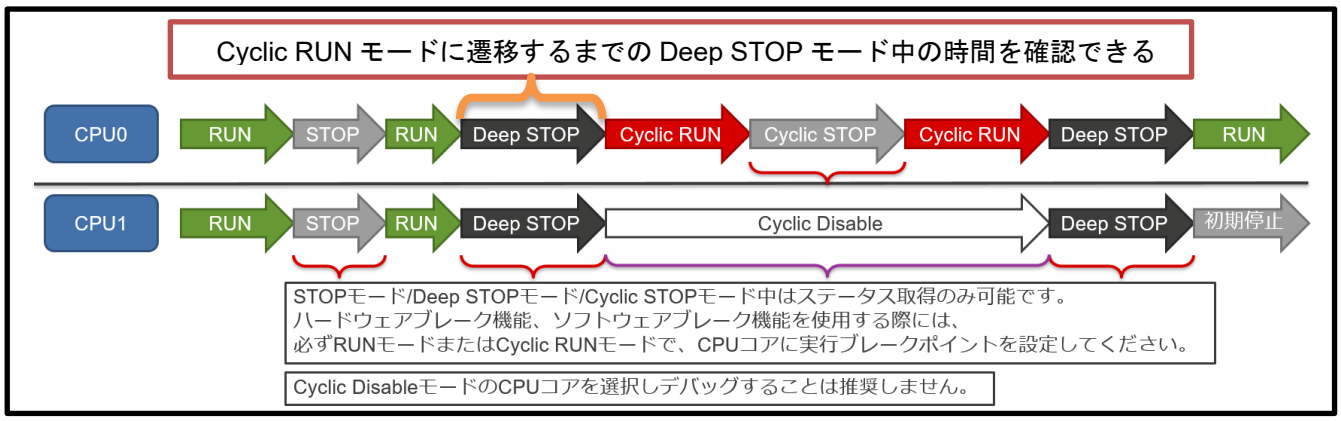

図 4-35 スタンバイモードに遷移するアプリケーション例での Cyclic RUN モードに遷移するまでの Deep STOP 中の時間確認

デバッガのタイマ機能を用いて、時間計測開始を Deep STOP モードに遷移するプログラムアドレスに設定し、計測終了を Deep STOP モードから Cyclic RUN モードに遷移した直後の Retention RAM の先頭アドレスに設定します。その後プログラムを実行することで、CPU0 のプログラムで Deep STOP モードに遷移してから、Cyclic RUN モードに遷移するまでにかかった時間を計測することができます。

4.2.3.3 Cyclic RUN モード中の時間制約要件を満たせているか確認する手法

Cyclic RUN モードでのプログラム実行時間を計測する手法を説明します。以下に示す手法により、ユーザは Cyclic RUN モードに遷移している時間の制約要件が満たせているか確認することができます。

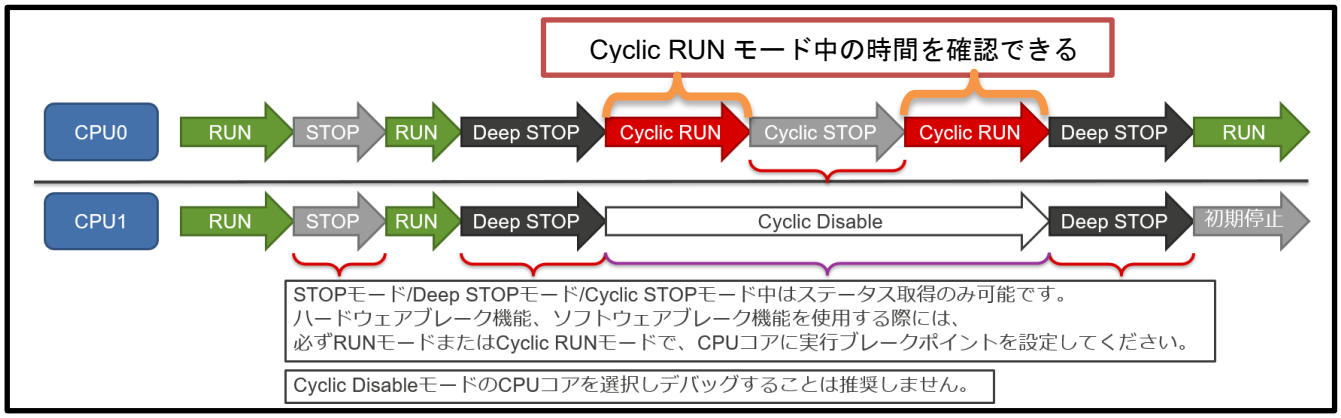

図 4-36 スタンバイモードに遷移するアプリケーション例での Cyclic RUN モード中の時間確認

デバッガのタイマ機能を用いて、時間計測開始を Cyclic RUN モードに遷移した Retention RAM の先頭アドレスに設定し、計測終了を Cyclic RUN モードから Cyclic STOP モードに遷移するプログラムアドレス、または Cyclic RUN モードから Deep STOP モードに遷移するプログラムアドレスに設定します。その後プログラムを実行することで、CPU0 のプログラムで Cyclic RUN モードに遷移してから、Cyclic STOP モードまたは Deep STOP モードに遷移するまでにかかった時間を計測することができます。

Cyclic RUN モード中の時間が制約要件を満たしていなかった場合、デバッガのタイマ機能とトレース機能により、時間計測開始/終了を設定しての経過時間計測や、トレースによる各処理にかかった時間の表示を用いて、Cyclic RUN 中のプログラムにおける時間確認をすることができます。これにより、プログラムのどの処理にどれだけの時間がかかったのかを見ることができます。

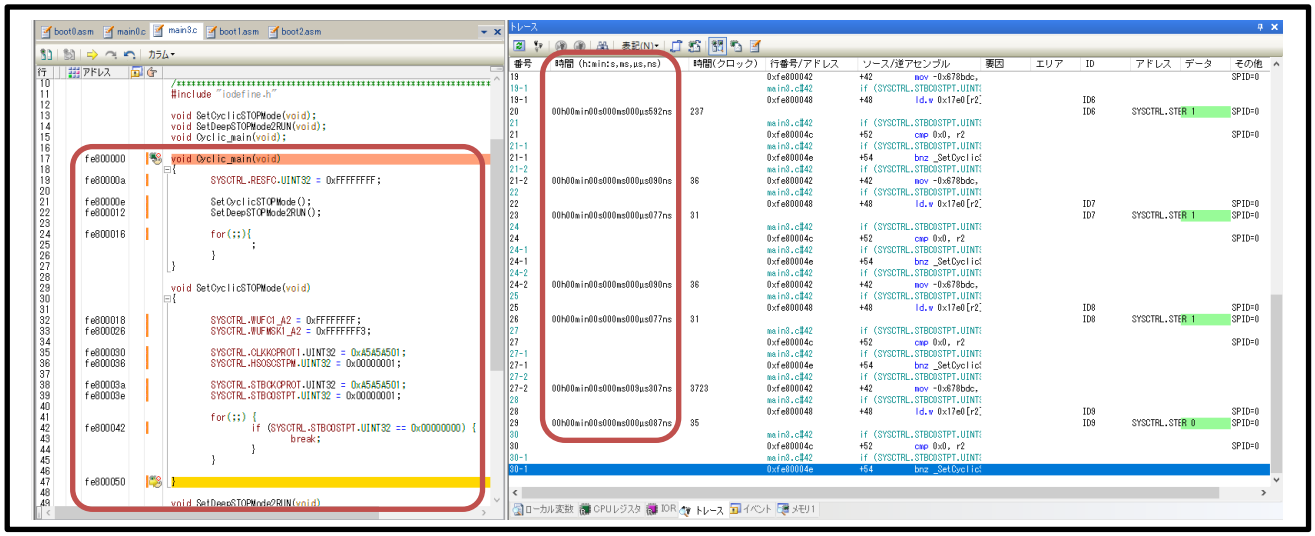

図 4-37 CS+での Cyclic RUN モードで実行したプログラムのトレース結果

4.2.4 Cyclic STOP モード

デバイスを Cyclic STOP モードに遷移させる手法、およびデバイスが Cyclic STOP モードであることを確認する手法を説明します。以下に示す手法により、ユーザはデバイスが Cyclic STOP モードであることを確認することができます。

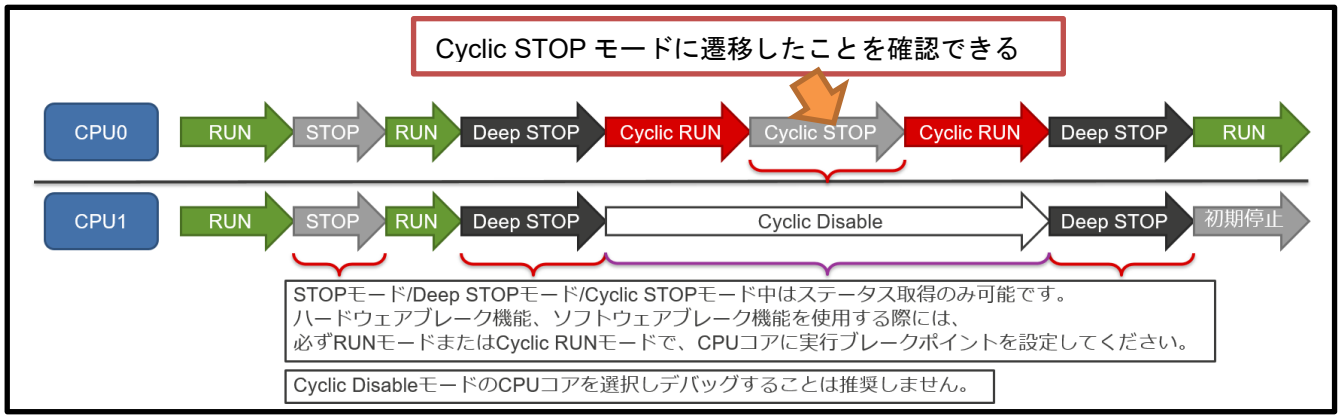

図 4-38 スタンバイモードに遷移するアプリケーション例での Cyclic STOP モード確認

図 4-39 は Cyclic STOP モードに遷移するプログラム例です。CPU コアが Cyclic RUN モードの時は Retention RAM 上でプログラムが実行されるため、Retention RAM にプログラムをダウンロードする必要があります。図 4-38 のアプリケーション例での CPU0 がこのプログラムを実行することにより、Cyclic STOP モードに遷移します。レジスタの詳細およびプログラミング内容はデバイスのハードウェアマニュアルを参照してください。

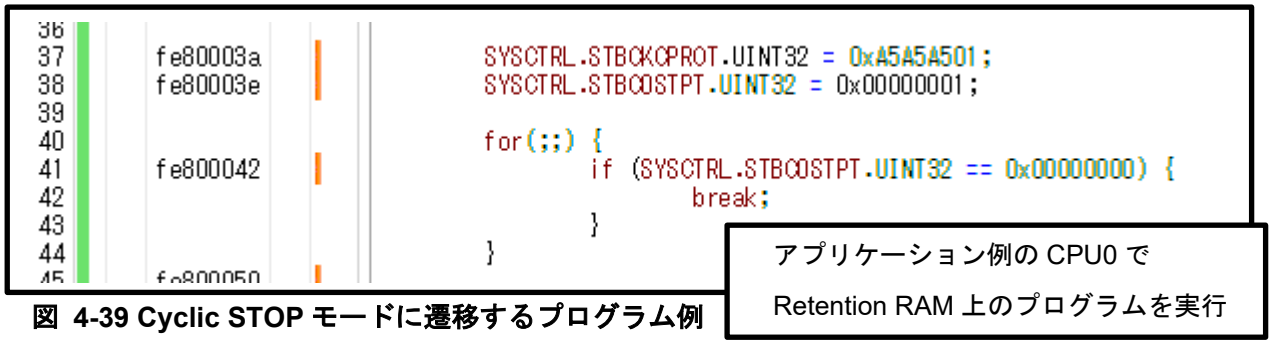

図 4-39 Cyclic STOP モードに遷移するプログラム例

ステータスを取得することで Cyclic STOP モードであることを確認できます。ステータスを取得した時の各デバッガでの表示を以下に示します。

●CS+

CS+では、Cyclic STOP モードであることのステータスを、「Cyclic Stop」と表示します。

図 4-40 CS+での Cyclic STOP モードステータス表示

●MULTI

MULTI では、Cyclic STOP モードであることのステータスを、数字で(0x800)、文字列で「CYCLE-STOP」と表示します。スタンバイモードはデバイスの状態であるため、CPU0 に Cyclic STOP モードのステータスを表示します。

```
850eserv2> cpustatus
CPU0 CPU status (0x800): CYCLE-STOP
Core is Running

CPU1 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU2 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU3 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running
```

図 4-41 MULTI での Cyclic STOP モードステータス表示

4.2.4.1 Cyclic STOP モードから Cyclic RUN モードに遷移した直後からデバッグする手法

Cyclic STOP モードに遷移後、ウェイクアップ要因によって Cyclic RUN モードに遷移した直後に CPU コアをブレークさせる手法を説明します。以下に示す手法により、ユーザは Cyclic STOP モードから Cyclic RUN モードに遷移した直後でのデバイスの状態を確認でき、Cyclic RUN モードに遷移した直後からアプリケーションのデバッグを開始できます。

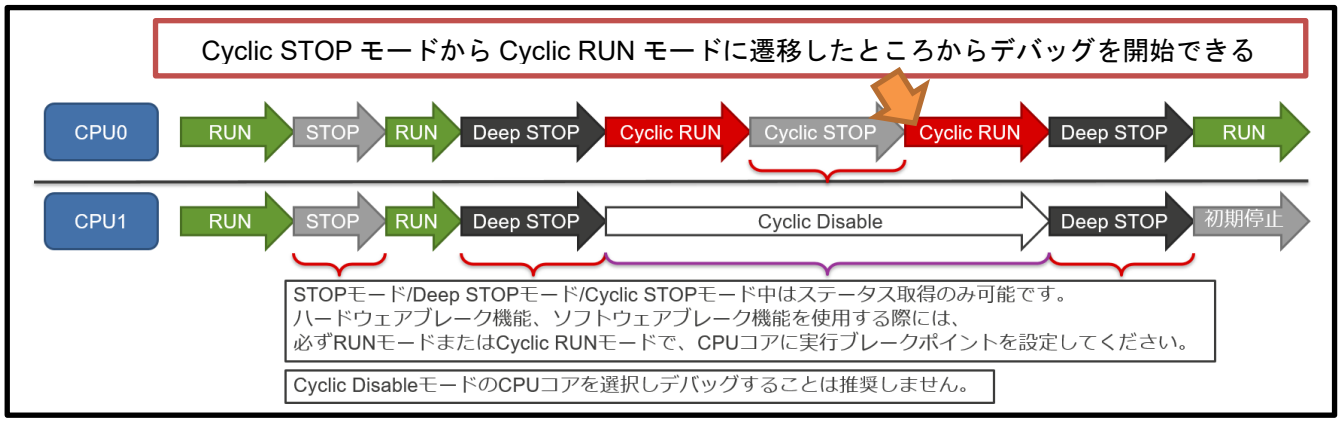

図 4-42 スタンバイモードに遷移するアプリケーション例での Cyclic STOP モードから Cyclic RUN モードへの遷移

図 4-43 は CPU コアが Cyclic STOP モードから Cyclic RUN モードに遷移したところでブレークしているプログラムデバッグ例です。CPU コアが Cyclic STOP モードから Cyclic RUN モードに遷移した時には、STBC0STPT レジスタ確認処理の後からプログラムを実行します。CPU コアが Cyclic RUN モードに遷移したところからデバッグするためには、Cyclic STOP モードに遷移する前に、あらかじめ STBC0STPT レジスタ確認処理の後に実行ブレークポイントを設定しておきます。これにより、CPU コアが Cyclic RUN モードに遷移したところでブレークすることができます。

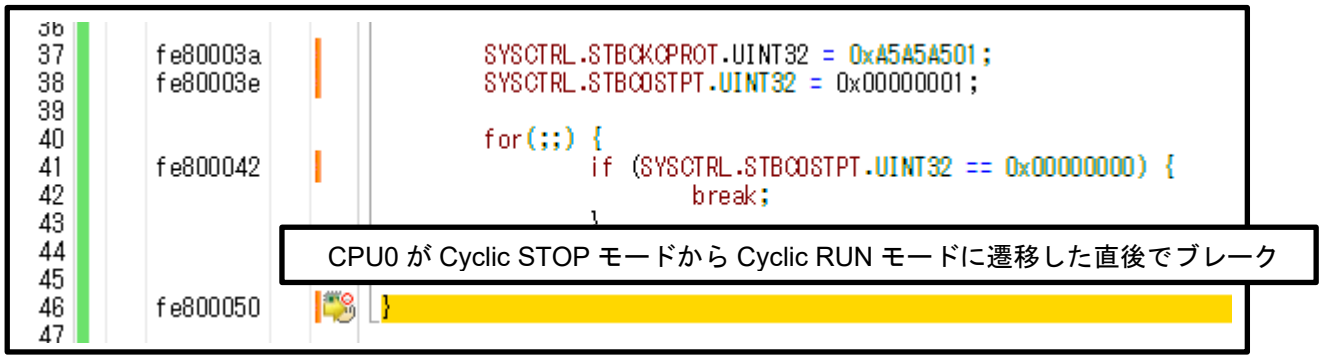

図 4-43 Cyclic STOP モードから Cyclic RUN モードに遷移時の実行ブレーク例

4.2.4.2 Cyclic STOP モード中の時間制約要件を満たせているか確認する手法

Cyclic STOP モードから Cyclic RUN モードに遷移するまでの時間を計測する手法を説明します。以下に示す手法により、ユーザは Cyclic STOP モードに遷移している時間の制約要件が満たせているか確認することができます。

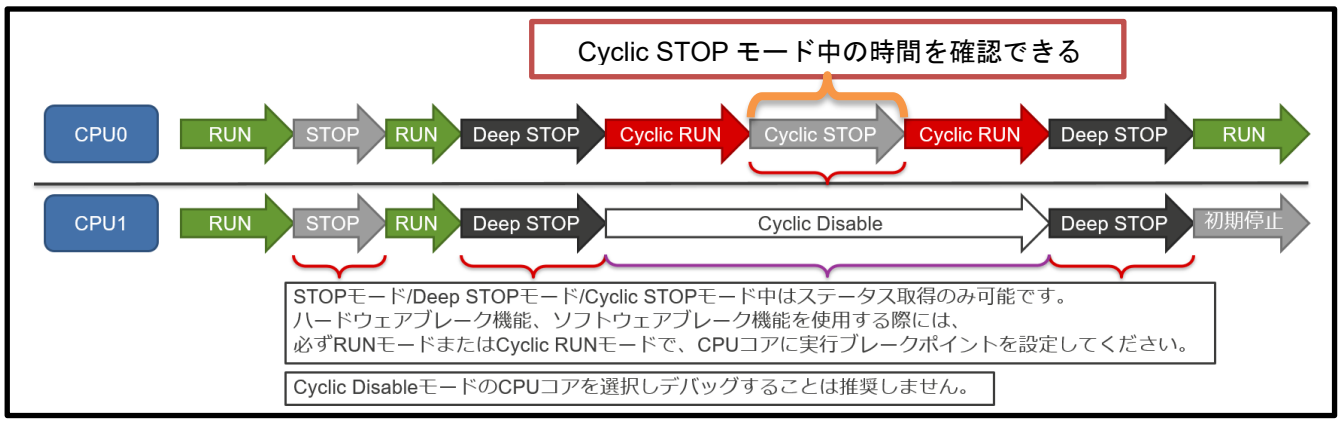

図 4-44 スタンバイモードに遷移するアプリケーション例での Cyclic STOP モード中の時間確認

デバッガのタイマ機能を用いて、時間計測開始を Cyclic STOP モードに遷移するプログラムアドレスに設定し、計測終了を Cyclic STOP モードから Cyclic RUN モードに遷移した直後のプログラムアドレスに設定します。その後プログラムを実行することで、CPU0 のプログラムで Cyclic STOP モードに遷移してから、Cyclic RUN モードに遷移するまでにかかった時間を計測することができます。

4.2.5 Cyclic Disable モード

CPU コアが Cyclic Disable モードであることを確認する手法を説明します。以下に示す手法により、ユーザは CPU コアが Cyclic Disable モードであることを確認することができます。

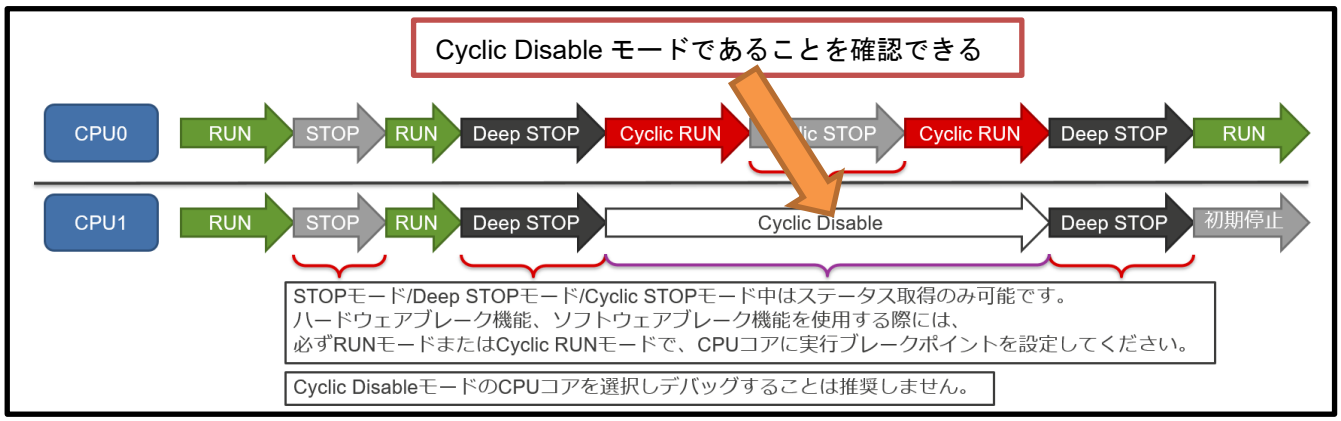

図 4-45 スタンバイモードに遷移するアプリケーション例での Cyclic Disable モード確認

ステータスを取得することで Cyclic Disable モードであることを確認できます。ステータスを取得した時の各デバッガでの表示を以下に示します。

●CS+

CS+では、Cyclic Disable モードであることのステータスを、「Cyclic Disable」と表示します。

図 4-46 CS+での Cyclic Disable モードステータス表示

●MULTI

MULTI では、Cyclic Disable モードであることのステータスを、数字で(0x1000)、文字列で「CYCLE-STOP INVALID」と表示します。スタンバイモードはデバイスの状態であるため、CPU0 以外に Cyclic Disable モードのステータスを表示します。

```
850eserv2> cpustatus
CPU0 CPU status (0x400): CYCLE-RUN
Core is Running

CPU1 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU2 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running

CPU3 CPU status (0x1000): CYCLE-STOP INVALID
Core is Running
```

図 4-47 MULTI での Cyclic Disable ステータス表示

5. 注意事項

初期停止コアを搭載するデバイス、およびスタンバイモード中のデバイスをデバッグする時の注意事項について説明します。

5.1 STOP/Cyclic STOP モードに遷移する処理のプログラム実行

STBC0STPT レジスタへの書き込み処理からプログラム実行開始すること、および STBC0STPT レジスタへの書き込み処理をステップ実行することは禁止です。また、STBC0STPT レジスタの確認ループ内にブレークポイントを設定することも禁止です。

STBC0STPT レジスタへの書き込み処理をデバッグする時には、STBC0STPT レジスタへの書き込み処理より前の処理でブレークした状態から、プログラム実行によって STOP モードまたは Cyclic STOP モードに遷移させてください。

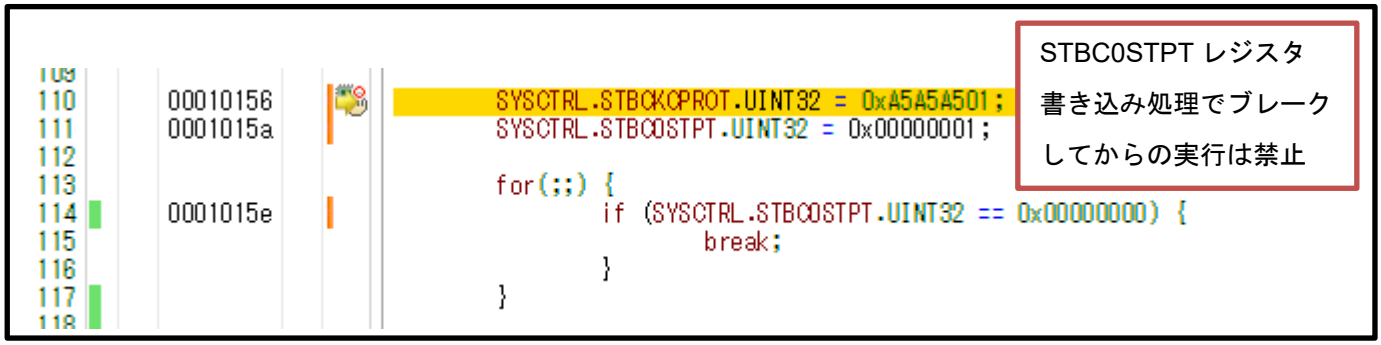

図 5-1 STBC0STPT レジスタ書き込み処理より前の処理でブレークする例

5.2 Deep STOP モードに遷移する処理のプログラム実行

STBC0PSC レジスタへの書き込み処理からプログラムを実行すること、および STBC0PSC レジスタへの書き込み処理をステップ実行することは禁止です。また、STBC0PSC レジスタへの書き込み処理後の無条件ループ内にブレークポイントを設定することも禁止です。

STBC0PSC レジスタへの書き込み処理をデバッグする時には、STBC0PSC レジスタへの書き込み処理より前の処理でブレークした状態から、プログラム実行によって Deep STOP モードに遷移させてください。

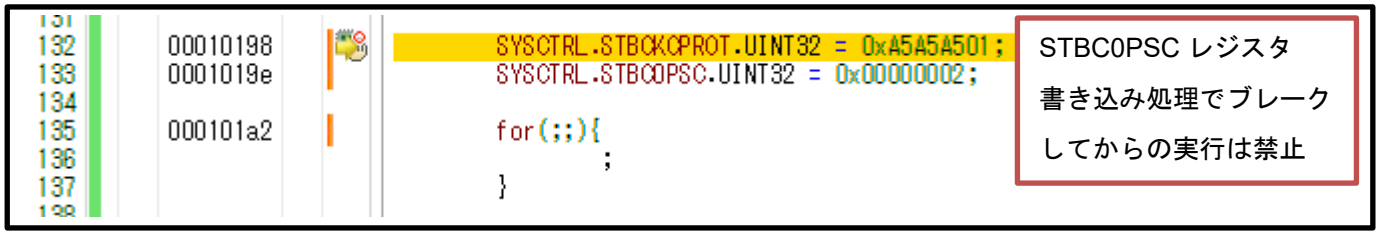

図 5-2 STBC0PSC レジスタ書き込み処理より前の処理でブレークする例

5.3 ICU-M コア有効デバイスでの初期停止コアのデバッグ

ICU-M コア有効デバイスでは、初期設定ではリセット解除時に起動するのは ICU-M コアであり、すべての CPU コアは初期停止コアになります。初期停止コアを動作させるには、ICU-M コアで初期停止コアを起動する必要があります。ICU-M コアのプログラムに初期停止コアを起動する処理を記述するか、デバッガ上で ICU-M コアを選択し、レジスタ操作で初期停止コアを起動してください。なお、ICU-M コア有効デバイスでもリセット解除時に CPU コアを起動させるかどうかは、オプションバイトによって設定することができます。

ICU-M コア有効であるが、ICU-M コアをデバッグ対象とせず CPU コアのみデバッグするメインコアデバッグの場合、すべての CPU コアは初期停止コアでありリセット解除時には初期停止状態になります。そして ICU-M コアのデバッグはできないため、デバイスのデバッグができない状態になります。メインコアデバッグで CPU コアのデバッグをする場合には、あらかじめ ICU-M コアのプログラムに初期停止コアを起動する処理を記述してください。プログラム例は 4.1.2 初期停止コアを起動 を参照してください。メインコアデバッグにおいて ICU-M コアはプログラム実行状態であるため、ICU-M コアが初期停止コアを起動する処理を実行することで、初期停止コアが起動します。これにより、デバイスのデバッグができます。

5.4 ICU-M コア有効デバイスでのスタンバイモードのデバッグ

ICU-M コア有効デバイスでは、スタンバイモードに遷移したデバイスを同期デバッグすることはできません。これによる影響として、ICU-M コア有効デバイスでは Cyclic RUN モードのデバッグはできなくなります。Cyclic RUN モードのデバッグを行う場合には、ICU-M コアの動作は不要であるため、ICU-M コアを無効にしてデバッグしてください。

スタンバイモードに遷移した ICU-M コア有効デバイスであっても、実行中のプログラムによりデバイスが RUN モードに遷移した時には、再び同期デバッグができます。

5.5 ホットプラグインデバッグ

デバイスが Cyclic RUN モードまたは Cyclic STOP モードの場合、ホットプラグイン接続はできません。デバイスが RUN モード、STOP モード、Deep STOP モードの時にはホットプラグイン接続できるため、ホットプラグイン接続を再度試みてください。

ホットプラグイン接続処理中にデバイスがスタンバイモードに遷移した場合には、ホットプラグイン接続に失敗します。その時にはホットプラグイン接続を再度試みてください。

ホットプラグイン接続直後のプログラム実行状態（以降、ホットプラグイン RUN 状態と呼称）では、ユーザは以下のデバッグ機能のみが使用可能であり、その他のデバッグ機能は使用できません。その他のデバッグ機能を使用したい場合には、ブレークによりホットプラグイン RUN 状態からブレーク状態に遷移させてください。

・RAM 領域のリード/ライト

・周辺 I/O レジスタのリード/ライト

・強制ブレーク ※

・強制リセット ※

※ デバイスに初期停止状態の初期停止コアがある場合またはデバイスが STOP モード/Deep STOP モード/Cyclic STOP モードである場合、ユーザは強制ブレークおよび強制リセットを使用できません。CS+にて、ホットプラグイン RUN 中のデバイスに初期停止状態の初期停止コアがある場合、ユーザが強制ブレークを使用した時に発生するエラーを図 5-3 に示します。MULTI にて、ホットプラグイン RUN 中のデバイスに初期停止状態の初期停止コアがある場合、ユーザが強制ブレークを使用した時に発生するエラーを図 5-4 に示します。強制ブレークおよび強制リセットを使用する時には、CPU コアのステータスを確認してください。

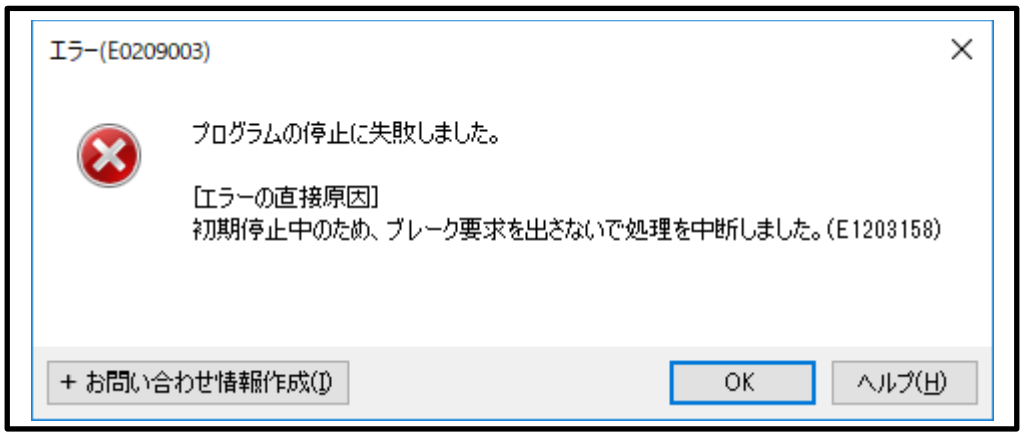

図 5-3 初期停止状態の初期停止コアがあるデバイスで強制ブレークを使用した時のエラー(CS+)

```
0x0c56:status err
(break request is canceled by fetch-stop)
```

図 5-4 初期停止状態の初期停止コアがあるデバイスで強制ブレークを使用した時のエラー(MULTI)

5.6 ハードウェアブレークポイントの設定/削除に関わる動作

起動中の CPU コアからプログラムの実行を開始した時、デバッガは初期停止状態の初期停止コアや Cyclic Disable の CPU コアに対してはハードウェアブレークポイントを設定できません。このため、起動中 CPU コアのプログラム実行中に初期停止コアが起動した時、または CPU コアが Cyclic Disable から起動した時に、起動した CPU コアがハードウェアブレークポイントを設定したアドレスを実行してもブレークしない場合があります。動作例を図 5-5 に示します。初期停止コアが起動している時、または CPU コアが Cyclic Disable ではない時に CPU コアに設定したハードウェアブレークポイントではブレーク動作します。

ハードウェアブレークポイントを設定する時には、4.1 および 4.2 を参照し、初期停止コアが起動または CPU コアが Cyclic Disable から起動した時に、一度ブレークさせてからハードウェアブレークポイントを設定しプログラム実行を開始することで、CPU コアに設定されます。

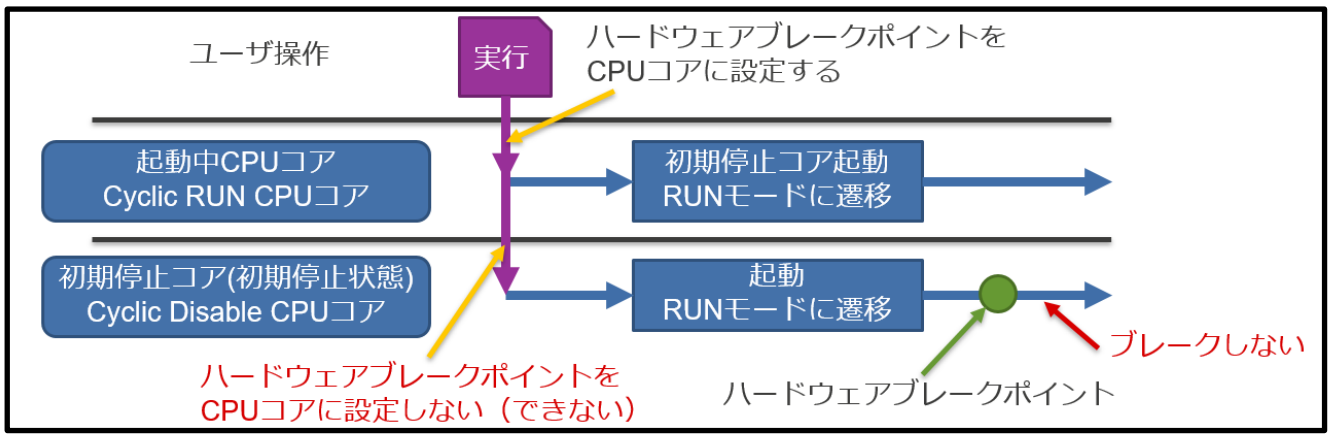

図 5-5 ハードウェアブレークポイントが CPU コアに設定されずブレークしない動作例

プログラム実行中に初期停止コアが初期停止状態または CPU コアが Cyclic Disable になった状態でブレークした時、再実行時にデバッガは設定されているハードウェアブレークポイントを削除できません。このため、ユーザが設定されているブレークポイントを無視してプログラム実行を再開し、実行中に初期停止コアが起動または CPU コアが Cyclic Disable から起動した時、削除できなかったハードウェアブレークポイントが成立し、ブレークが発生します。動作例を図 5-6 に示します。

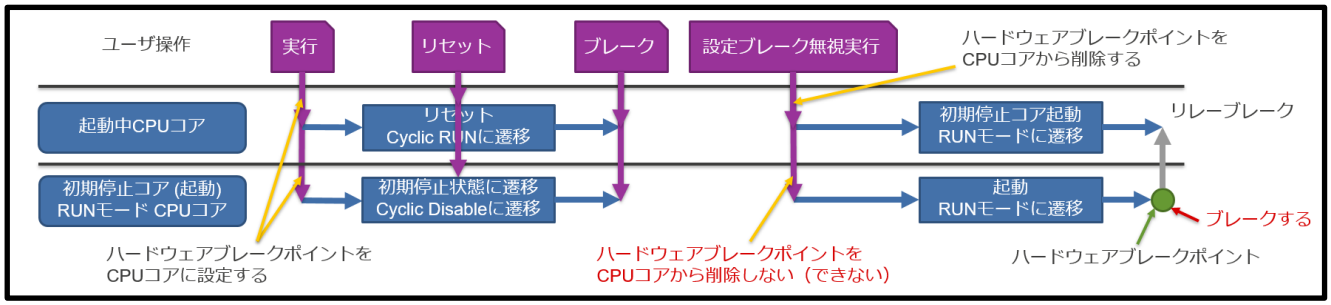

図 5-6 ハードウェアブレークポイントが CPU コアから削除されずブレークする動作例

5.7 ダウンロード時での初期停止コアのハードウェアブレークポイント設定状態

デバッガの設定によっては、プログラムダウンロード後に、デバッガはデバイスにリセットを発行します。これにより、初期停止コアは初期停止状態になります。

CS+および MULTI では、プログラムをダウンロードする前の最後の実行時に起動状態の初期停止コアに設定されていたハードウェアブレークポイントが設定された状態になります。最後の実行操作を行った後からダウンロード操作を行う前までに設定したハードウェアブレークポイントは、デバイスに設定されません。

プログラムの書かれたアドレスをハードウェアブレークポイントとしてデバイスに設定します。プログラムを修正してダウンロードした場合でも、デバイスに残っているハードウェアブレークポイントは修正前のプログラムでのハードウェアブレークポイントになります。

動作例を図 5-7 に示します。デバッガが修正後のプログラムで設定しているハードウェアブレークポイントをデバイスに設定するのは、初期停止コアが起動後にブレークし再実行した時です。

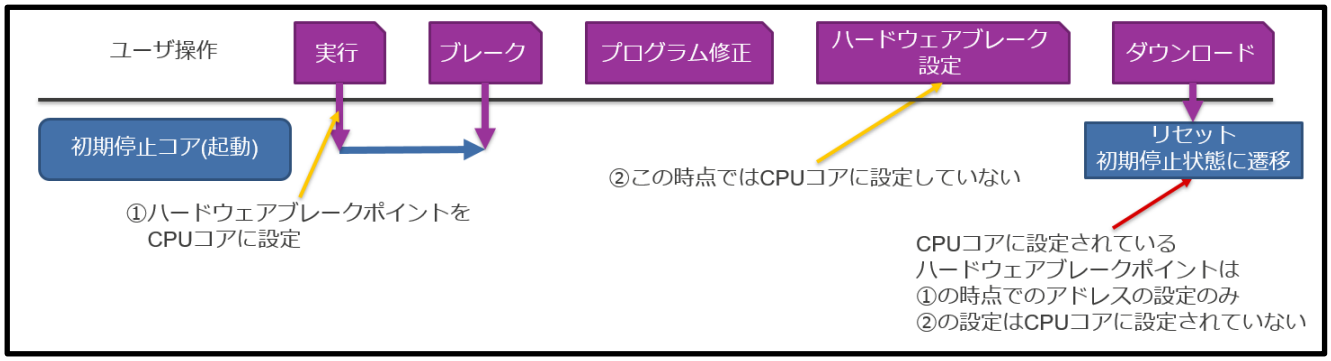

図 5-7 CS+/MULTI でのダウンロード後におけるハードウェアブレークポイントの設定状態

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2020/01/08	-	初版

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、V_{IL}（Max.）から V_{IH}（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、V_{IL}（Max.）から V_{IH}（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。