

StartUp Code

1. はじめに

C言語のプログラムは `main()` 関数から始まりますが、その `main()` が呼ばれる前に細かい準備を担当するスタートアップルーチンがあります。秋月 ARM ボードで安定するように作成したスタートアップルーチンの内容説明を記載するドキュメントです。

2. RAM 動作用と ROM 動作用

RAM 動作用とは、NJARM7 により直接 AT91R40807 の内蔵 SRAM ヘダウンロードして使用することを想定したスタートアップルーチンです。

ROM 動作用とは、ARM ボードに搭載されているフラッシュ ROM AT29LV1024 へ書き込んだ後、PC 無しでも動作できることを想定したスタートアップルーチンです。

起動開始番地が異なるのですが、スタートアップルーチン以外の部分は共通のモノが使えるように、動作上の互換性を持たせました。スタートアップルーチンのサイズが異なること以外は `main()` 動作開始時点で同様の動作になります。

3. ソース解説

3. 1. RAM 動作用

ソースファイル名は `ram_crt0.s` です。

このスタートアップルーチンが受け持つ処理を Fig.3.1.1 に示します。

RAM 実行のアプリは SecondaryRAM (つまり 0x1000000 番地以降) で動作することを前提に作成されています。(NJARM7 等で RAM へ転送して実行した場合など)

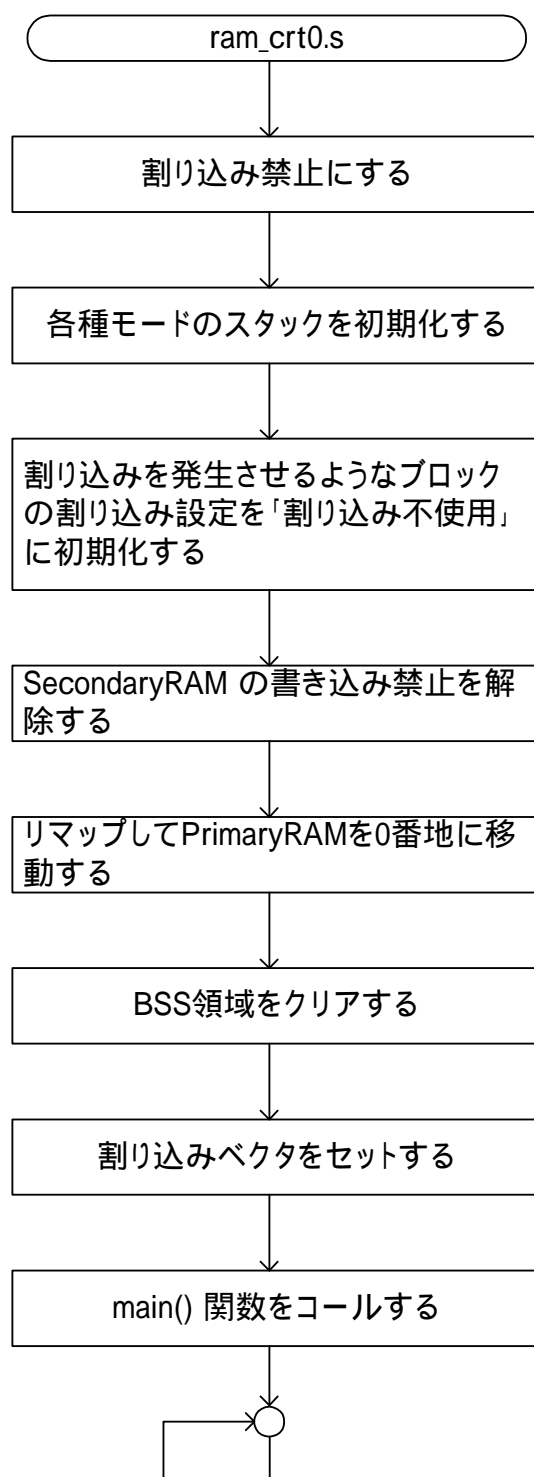


Fig.3.1.1. ram crt0.s の大まかな流れ

NJARM7 で RAM ヘダウンロードした直後のメモリ状態を Fig.3.1.2.に示します。

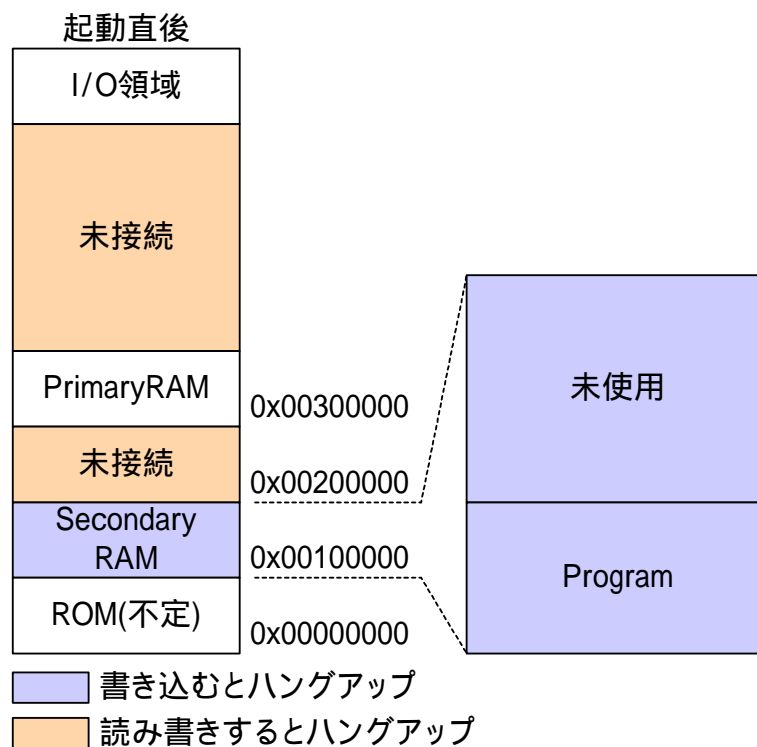


Fig.3.1.2. ダウンロード直後

・割り込み禁止

通常、正常な状態で起動された場合は ARM とその周辺がリセットされるので、すべて割り込み不使用に初期化されています。しかし、強制的に RAM ヘダウンロードして使うような場合には、各ブロックがそれまでの状態を保持している場合があるため、設定中に不要な割り込みが発生しないように、真っ先に割り込みを禁止してしまいます。

といっても、割り込みを禁止する処理を実行するまでに割り込まれてしまうこともあるので、これが万全とは言えません。安定動作を望むなら、確実にリセットしてから実行開始するのが望ましいです。

・スタックの初期化

ARM にはいくつかの動作モードがあり、そのモードごとに個別のスタックポインタを持っています。割り込みを正常に使うために割り込みモードの時のスタックポインタと、通常動作時のスタックポインタを初期化しています。

その他にもモードはありますが、それらはスタックを使わないので初期化を省略しています。

- ・割り込み不使用設定

割り込みに関連するブロックの割り込み設定を確実に初期化します。

- ・ SecondaryRAM の書き込み禁止解除

正常起動した場合、0x1000000 番地以降に SecondaryRAM が現れていますが、これは書き込み禁止に設定されています。そのままの状態ですここに書き込むとハングアップします。BSS クリア処理で書き込みが発生するので、そのための対策です。

書き込み禁止解除直後のメモリマップを Fig.3.1.3.に示します。

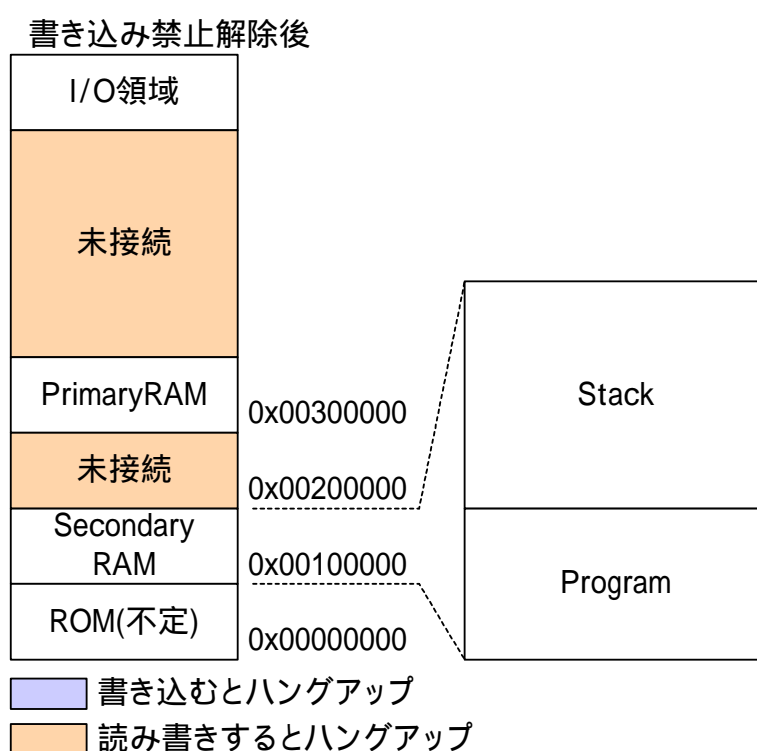


Fig.3.1.3. 書き込み禁止解除後

- ・リマップする

正常起動した場合、0x0000000 番地以降には外付け ROM が現れています。RAM 起動の場合、ROM には“都合の良い割り込みベクタ”が書かれているとは限らないので、リマップして 0x00000000 番地以降に PrimaryRAM を出現させます。これにより、割り込みベクタを書き換えられるようになります。このときのメモリマップを Fig.3.1.4 に示します。

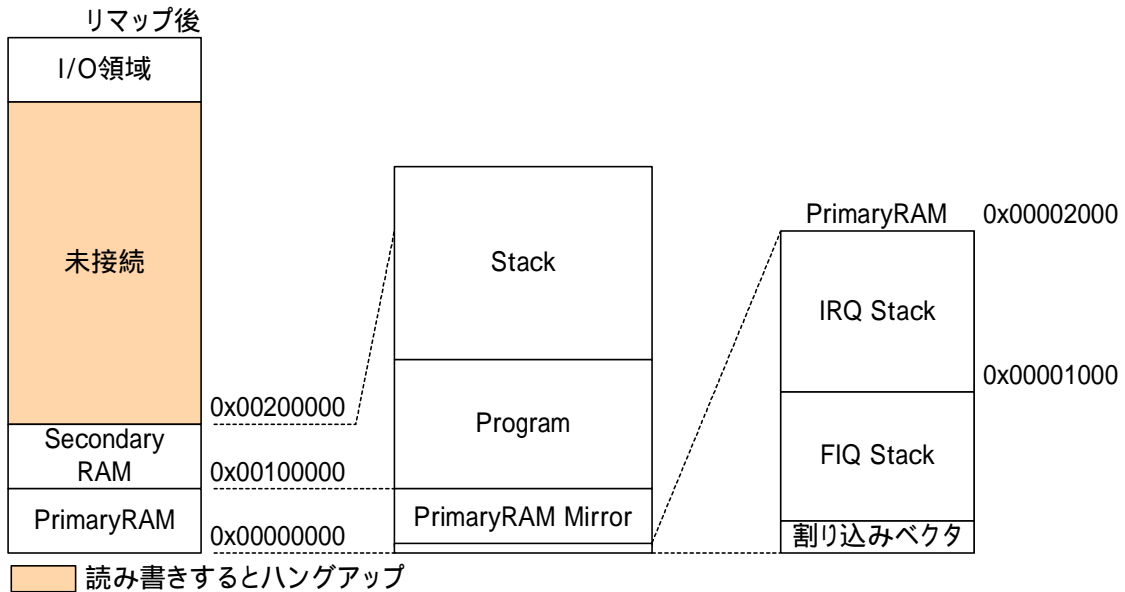


Fig.3.1.4. リマップ後

リマップ直後なので、PrimaryRAMにある割り込みベクタの内容は不定値です。

- ・BSS クリア

C言語の初期化子無しstatic変数などが置かれる場所です。C言語では、初期化子無しのstatic変数は0に初期化されることになっているので、ここで0を書き込んでいます。

- ・割り込みベクタをセットする

都合の良い割り込みベクタをPrimaryRAMへ転送します。

3. 2. ROM 動作用

ソースファイル名は rom_crt0.s です。

ROM動作用も基本的にRAM動作用と同じですが、リマップ前のROM上でそのまま動作する箇所と、SecondaryRAMへ転送後にそちらで動作する箇所の2通りが入り交じっています。コンパイラには全てSecondaryRAMで動作するように設定していますが、ROM上で動作するルーチンは再配置可能なコードのみで構築して0x00000番地にあるROMでの動作を実現しています。

大まかな流れを Fig.3.2.1 に示します。

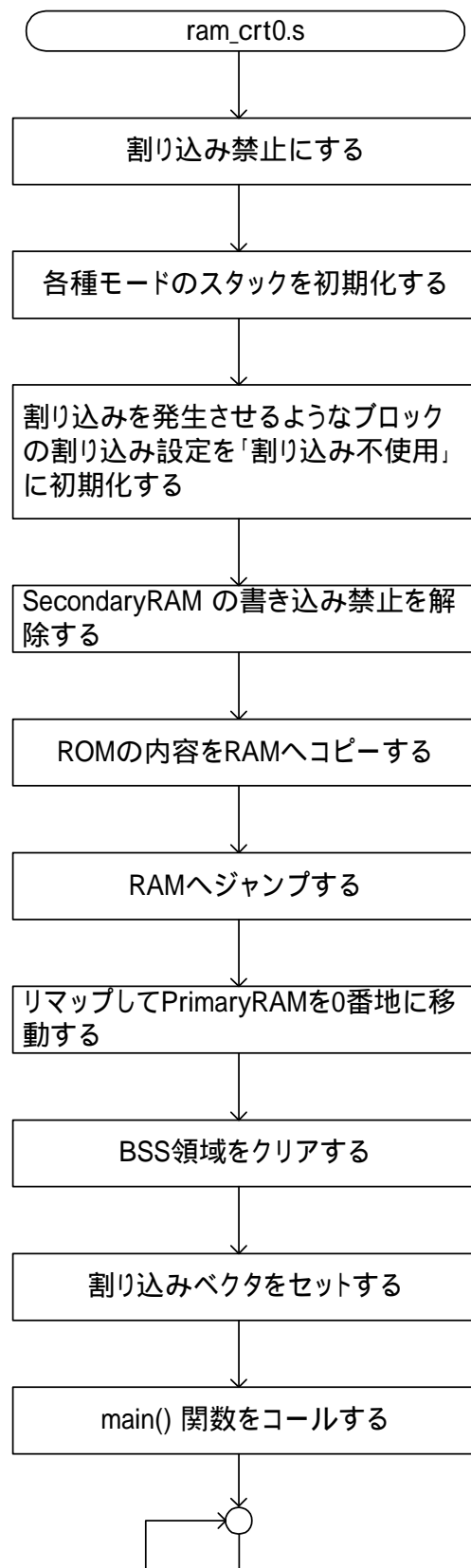


Fig.3.2.1 大まかな流れ

RAM ヘジャンプするところまでは、0x0000XX 番地にある ROM 上のコードが動作しています。RAM ヘジャンプするコードは次の箇所です。

ldr pc, =Remap Remap:

一見、すぐ次のアドレスヘジャンプしているだけの無意味なコードに見えますが、具体的に数値を割り付けると次のようになります。

0x0000XX	ldr	pc, =0x1000YY	
0x0000YY	Remap:		
		略	
0x1000XX	ldr	pc, =0x1000YY	@ RAM 上のコピー
0x1000YY	Remap:		@ RAM 上のコピー

ジャンプコードの直前に 0x00000000～0x01FFFF を 0x10000000～0x11FFFF ヘコピーしていますので、内容はそっくりです。しかし、ジャンプ命令 (ldr pc,=xxxx) は絶対ジャンプなので ROM 上のジャンプコードも、RAM 上のジャンプコードも、RAM 上の Remap ヘ飛んできます。

Remap ルーチンは、リマップ処理を実施して 0x0000000 番地にある ROM を隠してしまいましたが、すでに RAM ヘ待避済みなので暴走せずに動作します。

更新履歴：

更新日	リビジョン	更新内容
2004/08/19	1.0.0	初版