Nios II はじめてガイド
Nios II SBT と BSP Editor オプション設定

ver.16

2016 年 8 月 Rev.1

Nios II はじめてガイド

Nios II SBT と BSP Editor オプション設定

目次

Nios II はじめてガイド

Nios II SBT と BSP Editor オプション設定

Nios II はじめてガイド

Nios II SBT と BSP Editor オプション設定

1. はじめに

この「Nios II はじめてガイド」シリーズは、Nios® II プロセッサをはじめて使用するユーザ向けの資料です。

この資料は、Nios II Software Build Tool（Nios II SBT）とサブ・ツールである BSP Editor について概説した資料です。主に、BSP Editor の機能を中心に説明します。

2. ツール・バージョンの対応表

Nios II SBT は、以下の 3 つのソフトウェアで構成されています。汎用のフリー・ツールをベースにしているので、それぞれのツール・バージョンとの関係に注意する必要があります。

表 2-1 ツール・バージョンの対応表

アルテラ・ツール・バージョン		GUI 環境のバージョン	C/C++ クロス・コンパイラ
Nios SBT	ver.9.1	Eclipse 3.4 （コードネーム: Ganymede）	GCC ver.3.4.4
	ver.10.0	Eclipse 3.5 （コードネーム: Galileo）	GCC ver.3.4.4
	Ver.11.0	Eclipse 3.6.1 （コードネーム: Helios）	GCC ver.3.4.4
	Ver.12.0	Eclipse 3.6.1 （コードネーム: Helios）	GCC ver.3.4.4
	Ver.13.0	Eclipse 3.7.2 （コードネーム: Indigo）	GCC ver.4.5.3
	Ver.14.0	Eclipse 3.7.2 （コードネーム: Indigo）	GCC ver.4.5.3
	Ver.15.0	Eclipse 4.3.2 （コードネーム: Kepler）	GCC ver.4.8.3
	Ver.16.0	Eclipse 4.3.2 （コードネーム: Kepler）	GCC ver.4.8.3

3. BSP Editor について

3-1. BSP と BSP Editor

一般的に BSP とは、OS メーカや CPU ボード・メーカが提供するソフトウェア・パッケージであり、その実体は CPU ボード上で特定の OS を実行させるために必要なソフトウェア・ライブラリやデバイス・ドライバの集合体です。BSP は通常、ソース・ファイルやバイナリ・ファイルで提供されます。 組込みソフトウェアの開発者は、 BSP を利用して、開発対象の CPU ボードに合わせたスタートアップ・ルーチンやデバイスのドライバを作成します。

Nios II SBT の場合、Qsys システム内のコンポーネント用のデバイス・ドライバや HAL System Library に関連するソース・コードやオブジェクトをツールが自動生成して、BSP プロジェクト内に展開されます。BSP を独自に設定することでより精度の高いシステム構成を実現できます。このような設定を行う GUI ツールとして、Nios II SBT では新たに BSP Editor が導入されました。

BSP Editor は、主に組込みソフトウェアで必要となるメモリ・マップを詳細に設定できます。特に、リンカ・スクリプトの設定を詳細に行う場合、GUI によるパラメータ設定を行うだけで、自動生成できるようになりました。

このように、BSP Editor は強力なツールです。BSP Editor は、図 3-1 のように BSP プロジェクトをハイライトして、右クリックから Nios II ⇒ BSP Editor で起動できます。

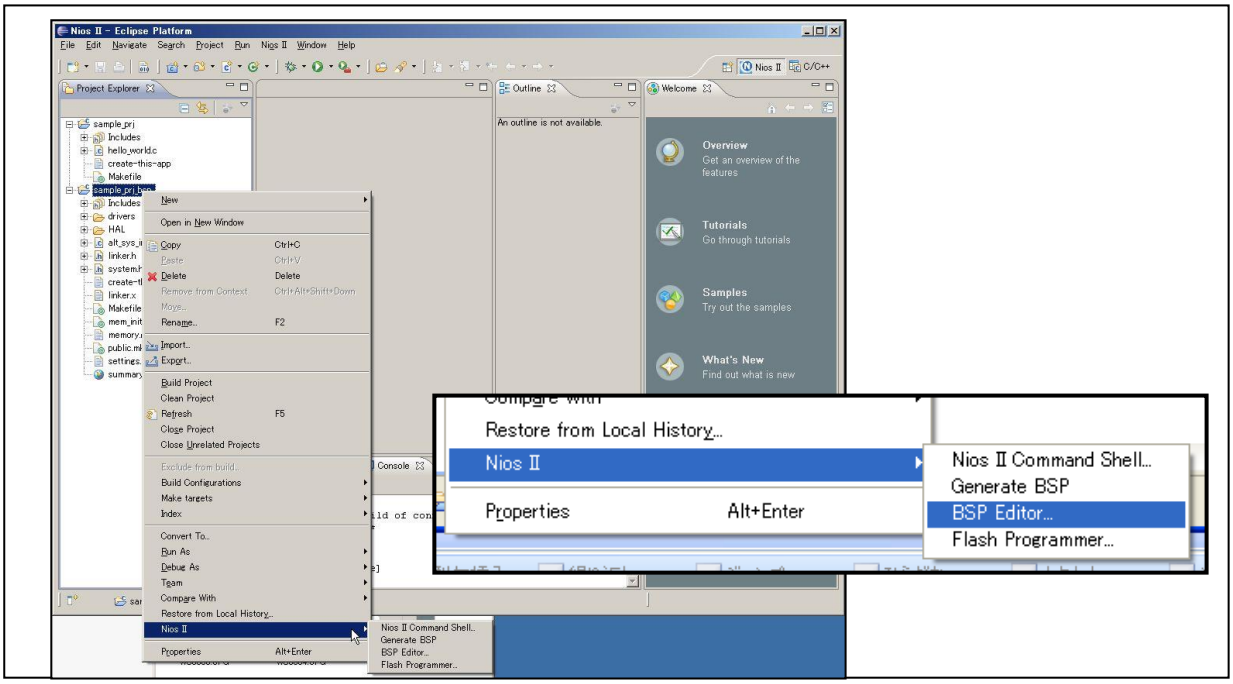

図 3-1. BSP Editor の起動

3-2. System Library Properties から BSP Editor のオプション設定へ

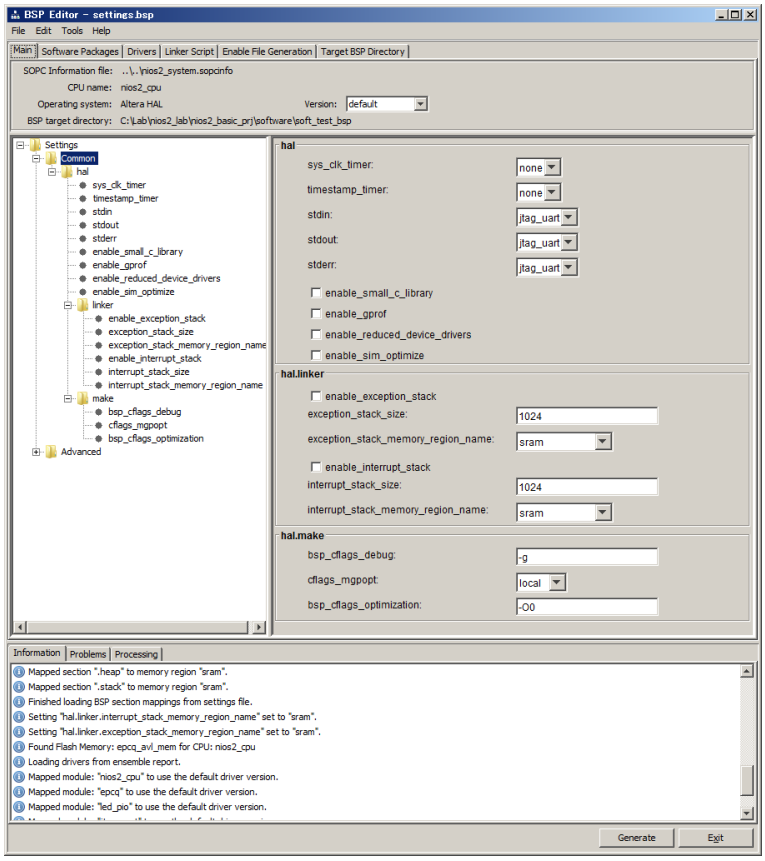

図 3-2. BSP Editor オプション設定①

左欄で、オプション設定の階層構造を見ることができます。

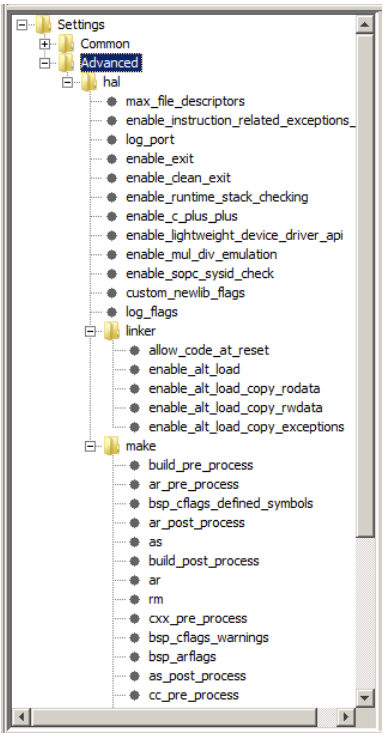

図 3-3. BSP Editor オプション設定②

左欄のオプション設定を個別にハイライトすると、右欄に、図 4-4 のようなオプションの説明が Description 欄に表示されます。

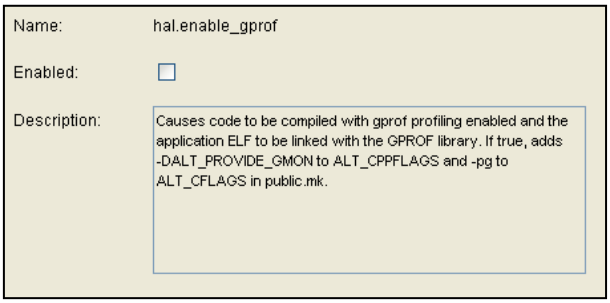

図 3-4. BSP Editor オプション設定③

3-3. BSP Editor のオプション設定

Main タブ、Software Packages タブ、Drivers タブを個別に選択すると、各タブ内に様々なオプション設定が表示されます。この章では、上記の図4-4 の Description 欄の説明を参考にして、各オプションについて説明します。また、上記の図 4-3 のように各オプションは用途に応じて分類されていますが、Advanced ディレクトリ内に分類されているオプションについては、「上級者向け」という表記を追加しています。

3-3-1. Main タブ内のオプション設定

Main タブを選択すると、以下に掲載されている多数のオプション設定が表示されます。

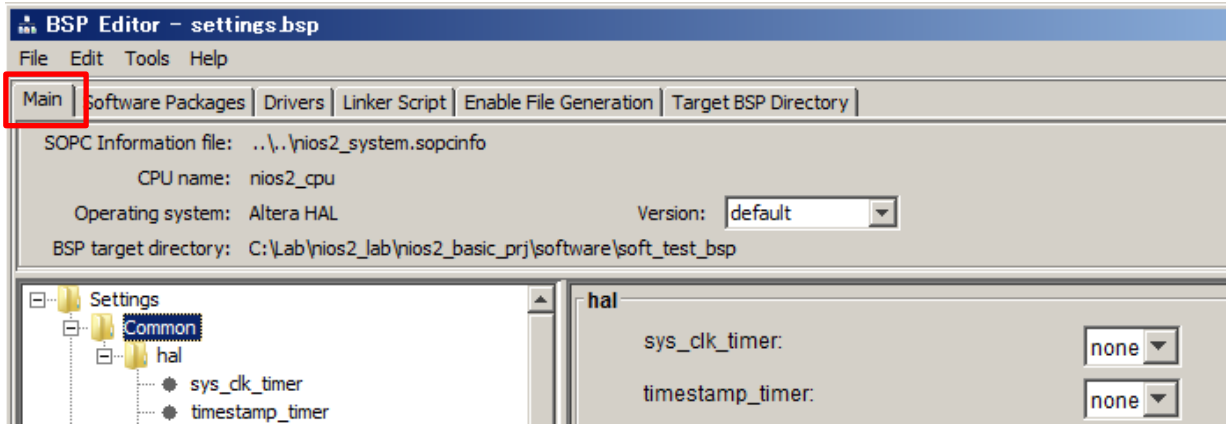

図 3-5. BSP Editor オプション設定④

3-3-1-1. enable_gprof

このオプションにチェックを入れた場合、gprof プロファイラを有効にしてコンパイルされます。この機能を有効にした場合、public.mk の ALT_CPPFLAGS に DALT_PROVIDE_GMON が追加されて、ALT_CFLAGS に -pg が追加されます。

3-3-1-2. enable_reduced_device_drivers

このオプションにチェックを入れた場合、高速/高機能モードとシンプル/低サイズ・モードの両モードが用意されているデバイス・ドライバに対して、後者のモードを選択します。この場合、コード・サイズやデータ・サイズが小さくなります。この機能は、すべてのドライバに適用されるとは限りません。altera_avalon_cfi_flash、altera_avalon_epcs_flash_controller, altera_avalon_lcd_16207 等のデバイス・ドライバについては、この機能はサポートされていません。このオプションにチェックを入れた場合、altera_avalon_uart および altera_avalon_jtag_uart のドライバは、割込み処理からポーリング処理にモードが切り替わります。なお、このとき、public.mk の ALT_CPPFLAGS に -DALT_USE_SMALL_DRIVERS が追加されます。

3-3-1-3. enable_sim_optimize

このオプションにチェックを入れた場合、キャッシュの初期化、.bss セクションのクリア、長い遅延ループを省略して、高速 HDL シミュレーション用途でコンパイルされます。この機能を有効にすると、public.mk の ALT_CPPFLAGS に -DALT_SIM_OPTIMIZE が追加されます。

なお、このオプションにチェックを入れた状態でビルドしたコードは、実際のアプリケーションには使用できません。

3-3-1-4. enable_small_c_library

このオプションにチェックを入れた場合、縮小版の newlib（組込み向けの C ライブラリ）を使用できます。newlib を使用すれば、コード・サイズやデータ・サイズが小さくなります。その際、printf() の浮動小数点サポート、stdin 入力ルーチン、I/O バッファが機能削減されます。この small c library は Micrium MicroC/OS-II との互換性はありません。この機能を有効にすると、public.mk の ALT_LDFLAGS に -msmallc が追加されます。

3-3-1-5. stderr

プルダウン・メニューで指定した標準エラー出力デバイスにスレーブ・ディスクリプタを設定します。この設定は、system.h のマクロ名 ALT_STDERR を定義します。

3-3-1-6. stdin

プルダウン・メニューで指定した標準入力デバイスにスレーブ・ディスクリプタを設定します。この設定は、system.h のマクロ名 ALT_STDIN を定義します。

3-3-1-7. stdout

プルダウン・メニューで指定した標準出力デバイスにスレーブ・ディスクリプタを設定します。この設定は、system.h のマクロ名 ALT_STDOUT を定義します。

3-3-1-8. sys_clk_timer

プルダウン・メニューで指定したシステム・クロック・タイマ・デバイスにスレーブ・ディスクリプタを設定します。このデバイスは周期的な割込み（RTOS を使用する際には標準的に必要です。）を提供します。この設定は、system.h のマクロ名 ALT_SYS_CLK に値を定義します。

3-3-1-9. timestamp_timer

プルダウン・メニューで指定したタイムスタンプ・タイマ・デバイスにスレーブ・ディスクリプタを設定します。このデバイスは高精度な時間測定に使用されます。この設定は、system.h のマクロ名 ALT_TIMESTAMP_CLK に値を定義します。

3-3-1-10. enable_exception_stack

このオプションにチェックを入れた場合、通常のスタックとは別のメモリ領域に例外処理用のスタック領域を確保します。この機能を有効にすると、linker.h にマクロ名 ALT_EXCEPTION_STACK を定義して、linker.x に exception_stack で呼び出されるメモリ領域を追加します。linker.x ではシンボル _alt_exception_stack_pointer と _alt_exception_stack_limit を定義します。この機能を使用する際、exception_stack_size と exception_stack_memory_region_name の設定も有効にする必要があります。MicroC/OS-II を使用する場合、この機能を無効にする必要があります。外部割込みコントローラを使用しないとき、例外処理用のスタックは、割込み等の例外処理のパフォーマンスを改善する目的で使用されます。

3-3-1-11. enable_ineterrupt _stack

このオプションにチェックを入れた場合、通常のスタックとは別のメモリ領域に割込み処理用のスタック領域を確保します。

この機能を有効にした場合、linker.h にマクロ名 ALT_INTERRUPT_STACK を定義して、linker.x に interrupt_stack で呼び出されるメモリ領域を追加します。

linker.x ではシンボル _alt_interrupt_stack_pointer と _alt_interrupt_stack_limit を定義します。この機能を使用する際、interrupt_stack_size と interrupt_stack_memory_region_name の設定も有効にする必要があります。MicroC/OS-II を使用する場合、この機能を無効にする必要があります。外部割込みコントローラが排他的に使用される場合にのみ有効になります。外部割込みコントローラを使用しないとき、例外処理用のスタックは、割込みおよびその他の例外処理のパフォーマンスを改善する目的で使用されます。

3-3-1-12. exception_stack_memory_region_name

プルダウン・メニューでメモリ領域名を指定して、exception_stack のメモリ領域を別の領域に確保します。この機能は、enable_exception_stack の機能を有効にした場合にのみ適用されます。

3-3-1-13. exception_stack_size

例外処理用のスタック・サイズをバイト単位で指定します。この機能は、enable_exceptin_stack を有効にした場合にのみ適用されます。

3-3-1-14. ineterrupt_stack_memory_region_name

プルダウン・メニューでメモリ領域名を指定して、interrupt_stack のメモリ領域を別の領域に確保します。この機能は、enable_interrupt_stack の機能を有効にした場合にのみ適用されます。

3-3-1-15. interrupt_stack_size

割込み処理用のスタック・サイズをバイト単位で指定します。この機能は、enable_interrupt_stack を有効にした場合にのみ適用されます。

3-3-1-16. bsp_cflags_debug

C/C++ コンパイラのデバッグ・レベルを指定します。-g を指定すると GNU デバッガを使用してデバッグできます。-g を省略した場合、ELF からデバッグ・シンボルが削除されます。この設定は、Makefile に BSP_CFLAGS_DEBUG の値を定義します。

3-3-1-17. bsp_cflags_optimization

C/C++ コンパイラの最適化レベルを指定します。(-O0 = 最適化は未実施。-O2 = 標準的な最適化が実施。)デバッグ用コードが埋め込まれる -O0 の使用を推奨します。この設定は、Makefile に BSP_CFLAGS_OPTIMIZATION の値を定義します。

3-3-1-18. custom_newlib_flags ※上級者向け

スペースで区切られたカスタムの newlib コンパイラ・フラグを指定して、カスタム版の newlib をビルドします。ビルド後、<bsp root>/newlib ディレクトリに配置されて、このライブラリや BSP を利用したアプリケーションのみに適用されます。

3-3-1-19. enable_c_plus_plus ※上級者向け

このオプションにチェックを入れた場合、C++ 言語のサポートを有効にしますが、C++ コンストラクタのサポートが追加されるのでコード・サイズやデータ・サイズが増えます。多重継承や例外処理のような機能はサポートされません。このオプションのチェックを外した場合、public.mk の ALT_CPPFLAGS に -DALT_NO_C_PLUS_PLUS が追加されて、コード・サイズやデータ・サイズが小さくなります。

3-3-1-20. enable_clean_exit ※上級者向け

アプリケーションを exit() 関数で強制終了する際、C++ デストラクタが呼び出される等により、ファイル・ディスクリプタをクローズします。このオプションのチェックを外すと、このような clean exit の処理が行われないので、コード・サイズやデータ・サイズが小さくなります。このオプションのチェックを外した場合、public.mk の ALT_CPPFLAGS に -DALT_NO_CLEAN_EXIT が追加されて、ALT_LDFLAGS に -Wl, --defsym, exit=_exit が追加されます。

3-3-1-21. enable_exit ※上級者向け

このオプションにチェックを入れた場合、exit() 関数のサポートが追加されます。(main() ルーチンは return 文や exit() を呼び出して終了します。)この機能を有効にすると、コード・サイズやデータ・サイズが増えます。このオプションのチェックを外した場合、public.mk の ALT_CPPFLAGS に -DALT_NO_EXIT を追加して、コード・サイズを小さくします。

3-3-1-22. enable_instruction_related_exceptions_api ※上級者向け

このオプションにチェックを入れた場合、インストラクション要因(Trap 例外など)による例外処理をサービスするハンドラを登録する API を有効にします。Nios II プロセッサの様々な機能オプション(MMU、MPU 他)を設定すると、それぞれの例外タイプを生成します。この設定を有効にする場合、例外エントリのコード・サイズが増えます。

3-3-1-23. enable_lightweight_device_driver_api ※上級者向け

このオプションにチェックを入れた場合、Lightweight device driver API によって、ファイル・ディスクリプタにデバイス名を割り付ける機能(例 /dev/uart0)が削除されて、ドライバのコード・サイズやデータ・サイズが小さくなります。それに伴い、open(), close(), lseek() ルーチンが呼び出されたときには動作不全に陥りますが、read(), write(), fstat(), ioctl(), isatty() ルーチンが呼び出されたときには標準入出力デバイスに対して正常に動作します。この機能が有効の場合、public.mk の ALT_CPPFLAGS に -DALT_USE_DIRECT_DRIVERS が追加されます。このとき、ALTERA ホスト・ファイル・システムとリード専用 ZIP ファイル・システムは使用できません。

3-3-1-24. enable_mul_div_emulation ※上級者向け

このオプションにチェックを入れた場合、Nios II/Economy ではサポートしていない multiply 命令と divide 命令をエミュレートするコードが追加されます。チェックを外した場合、public.mk の ALT_CPPFLAGS に -DALT_NO_INSTRUCTION_EMULATION が追加されます。

3-3-1-25. enable_runtime_stack_checking ※上級者向け

このオプションにチェックを入れた場合、スタック・オーバーフローのチェック機能を有効にします。その場合、ヒープ領域やスタティック領域に割り付けられたデータがスタック領域と衝突した場合、break 例外を発生するコードを埋め込みます。この場合、public.mk の ALT_CPPFLAGS に -DALT_STACK_CHECK と -mstack-check が追加されます。

3-3-1-26. enable_sopc_sysid_check ※上級者向け

このオプションにチェックを入れた場合、システム ID チェックを有効にします。Qsys の System ID コンポーネントが対応する CPU に接続されている場合、システム ID チェックが有効な状態で ELF ファイルのダウンロードが可能になります。この機能を無効にした場合、ダウンロードした ELF ファイルと、ターゲットのハードウェア・デザインとの整合性は保証されません。

その場合、public.mk の SOPC_SYSID_FLAG に -accept-bad-sysid が追加されます。

3-3-1-27. log_flags ※上級者向け

Value 欄に入力する 1～4 の値は、public.mk の ALT_LOG_FLAGS に適用されます。この機能を使用するには、log_port が使用できる状態であることを前提とします。

3-3-1-28. log_port ※上級者向け

プルダウン・メニューで指定したキャラクタ・モード・デバイスにデバッグ・ログ用のディスクリプタを適用します。この設定は、system.h のマクロ名 ALT_LOG_PORT に値を定義します。

3-3-1-29. max_file_descriptors ※上級者向け

Value 欄には、ファイル・ディスクリプタ数が入力されます。enable_lightweight_device_driver_api が有効の場合、この設定は無視されます。enable_lightweight_device_driver_api が無効の場合、/dev/null、/dev/stdio、/dev/stdout、/dev/stderr の 4 つのファイル・ディスクリプタが必要になるので、Value 欄には 4 以上の値を入力します。 この設定は、system.h のマクロ名 ALT_MAX_FD に定数の値を定義します。

3-3-1-30. allow_code_at_reset ※上級者向け

このオプションにチェックを入れた場合、初期化コードがリセット・アドレスに配置されます。この機能が有効な場合、linker.h のマクロ名 ALT_ALLOW_CODE_AT_RESET を定義します。外部のブートローダ（例：フラッシュ・ブートローダ）が存在する場合、このオプションのチェックを外します。

3-3-1-31. enable_alt_load ※上級者向け

このオプションにチェックを入れた場合、alt_load() を有効にします。alt_load() は、.text メモリから RAM にセクションをコピーします。この機能が有効の場合、.text メモリにロードされるように、linker.x でセクションの VMA/LMA を設定します。外部のブートローダ（例：フラッシュ・ブートローダ）が存在する場合、このオプションのチェックを外します。

3-3-1-32. enable_alt_load_copy_exceptions ※上級者向け

このオプションにチェックを入れた場合、alt_load() に、.exceptions セクションをコピーします。この機能が有効の場合、linker.h のマクロ名 ALT_LOAD_COPY_EXCEPTION を定義します。

3-3-1-33. enable_alt_load_copy_rodata ※上級者向け

このオプションにチェックを入れた場合、alt_load() に、.rodata セクションをコピーします。この機能が有効の場合、linker.h のマクロ名 ALT_LOAD_COPY_RODATA を定義します。

3-3-1-34. enable_alt_load_copy_rwdata ※上級者向け

このオプションにチェックを入れた場合、alt_load() に、.rwdata セクションをコピーします。この機能が有効の場合、linker.h のマクロ名 ALT_LOAD_COPY_RWDATA を定義します。

3-3-1-35. ar ※上級者向け

Value 欄のアーカイバー・コマンドは、ライブラリ・ファイルを作成します。

3-3-1-36. ar_post_process ※上級者向け

Value 欄のコマンドは、アーカイバー実行後に実行されます。

3-3-1-37. ar_pre_process ※上級者向け

Value 欄のコマンドは、アーカイバー実行前に実行されます。

3-3-1-38. as ※上級者向け

Value 欄のアセンブラ・コマンドを実行すると、C コンパイラ(CC)が .S ファイルを生成します。

3-3-1-39. as_post_process ※上級者向け

Value 欄のコマンドは、各アセンブリ・ファイルをコンパイルした後に実行されます。

3-3-1-40. as_pre_process ※上級者向け

Value 欄のコマンドは、各アセンブリ・ファイルをコンパイルする前に実行されます。

3-3-1-41. bsp_arflags ※上級者向け

Value 欄のフラグは、GNU Make が持つデフォルトの設定内容を使用しないように、標準の ARFLAGS ではなく、直接アーカイバーに渡されます。この設定は、Makefile に BSP_ARFLAGS の値を定義します。

3-3-1-42. bsp_asflags ※上級者向け

Value 欄のフラグは、アセンブラに渡されます。この設定は、Makefile に BSP_ASFLAGS の値を定義します。

3-3-1-43. bsp_cflags_defined_symbols ※上級者向け

Value 欄のシンタックスは、ソース・コードで #define を使用したマクロ定義と同じ効果があります。-DALT_DEBUG を設定すると、ソース・ファイルの #define ALT_DEBUG と同じ効果があります。さらに、-DFOO=1 を設定すると、ソース・ファイルの #define FOO 1 と等価になります。設定したマクロ定義は、BSP のすべての .S、.c および C++ ファイルに適用されます。この設定は、BSP Makefile に BSP_CFLAGS_DEFINED_SYMBOLS の値を定義します。

3-3-1-44. bsp_cflags_undefined_symbols ※上級者向け

Value 欄のシンタックスは、ソース・コードで #under を使用したマクロ定義の取り消しに似ています。マクロ名 FOO の取り消しを行う場合、シンタックス -u FOO を設定します。これは、ソース・ファイルの #undef FOO と等価になります。このシンタックスはマクロ名とは異なります。(例えば、-DFOO とは異なり、-u FOO のようにスペースが入ります。)マクロ定義の取り消しは、BSP のすべての .S、.c、C++ ファイルに適用されます。この設定は、BSP Makefile に BSP_CFLAGS_UNDEFINED_SYMBOLS の値を定義します。

3-3-1-45. bsp_cflags_user_flags ※上級者向け

Value 欄のフラグは、C、C++、.S ファイルをコンパイルする時、コンパイラに渡されます。この設定は、Makefile に BSP_CFLAGS_USER_FLAGS の値を定義します。

3-3-1-46. bsp_cflags_warnings ※上級者向け

Value 欄のフラグは、C/C++ コンパイラの警告レベルです。-Wall が一般的に使用されます。この設定は、Makefile の BSP_CFLAGS_WARNINGS の値を定義します。

3-3-1-47. bsp_cxx_flags ※上級者向け

Value 欄のフラグは、C++ コンパイラに渡されます。この設定は、Makefile の BSP_CXXFLAGS の値を定義します。

3-3-1-48. bsp_inc_dirs ※上級者向け

Value 欄には、インクルード・ディレクトリ名を指定します。複数のインクルード・ディレクトリ名をスペースで区切って入力します。これはヘッダ・ファイルをスキャンするときに使用します。ディレクトリは、トップレベルの BSP ディレクトリに対する相対ディレクトリです。ディレクトリからインクルード・ファイルを探す場合、make ファイルではサフィックス –I が使われますが、ここでは使用しません。この設定は、Makefile の BSP_INC_DIRS の値を定義します。

3-3-1-49. build_post_process ※上級者向け

Value 欄のコマンドは、BSP をビルドした後に実行されます。

3-3-1-50. build_pre_process ※上級者向け

Value 欄のコマンドは、BSP をビルドする前に実行されます。

3-3-1-51. cc ※上級者向け

Value 欄は、C コンパイラ・コマンドです。

3-3-1-52. cc_post_process ※上級者向け

Value 欄のコマンドは、各 .c/.S ファイルがコンパイルされた後に実行されます。

3-3-1-53. cc_pre_process ※上級者向け

Value 欄のコマンドは、各 .c/.S ファイルがコンパイルされる前に実行されます。

3-3-1-54. cxx ※上級者向け

Value 欄は、C++ コンパイラ・コマンドです。

3-3-1-55. cxx_post_process ※上級者向け

Value 欄のコマンドは、各 C++ ファイルがコンパイルされた後に実行されます。

3-3-1-56. cxx_pre_process ※上級者向け

Value 欄のコマンドは、各 C++ ファイルがコンパイルされる前に実行されます。

3-3-1-57. rm ※上級者向け

Value 欄のコマンドは、ターゲットを clean してファイルを削除する際に使用されます。

3-3-2. Software Packages タブ内のオプション設定

Software Packages タブを選択すると、ファイル・システムに関連したオプション設定が表示されます。

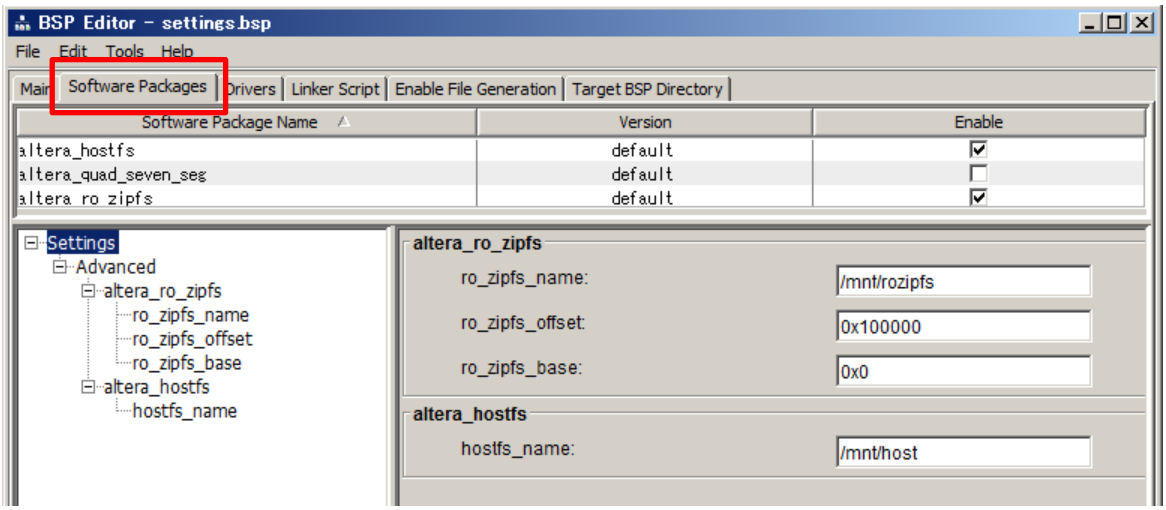

図 3-6. BSP Editor オプション設定⑤

Nios II では、「ホスト・ファイル・システム」と「zip ファイル・システム」をサポートしています。ホスト・ファイル・システムを実現させる場合、Nios II SBT で Debugger 画面を起動させて、Debug モードで実施する必要があります。

	ホスト・ファイル・システム	Zip ファイル・システム
ファイルの格納場所 （マウント・ポイント）	/mnt/host/	/mnt/rozips/
ファイルの格納媒体	ホスト PC 内のハードディスク。ダウンロード・ケーブルを経由してアクセス。	ターゲット・ボード上の CFI 準拠フラッシュ・メモリ
属性	リード/ ライト	リード専用

表 3-1. Nios II ファイル・システム

3-3-2-1. host_name ※上級者向け

Value 欄には、ホスト・ファイル・システムのマウント・ポイントを指定します。

3-3-2-2. ro_zips_base ※上級者向け

Value 欄には、リード専用 ZIP ファイル・システムで使用するフラッシュ・メモリ・デバイスのベース・アドレスを指定します。

3-3-2-3. ro_zips_name ※上級者向け

Value 欄には、リード専用 ZIP ファイル・システムのマウント・ポイントを指定します。

3-3-2-4. ro_zips_offset ※上級者向け

Value 欄には、リード専用 ZIP ファイル・システムで使用するフラッシュ・メモリ・デバイスのベース・アドレスを基点にしたオフセットを指定します。

3-3-3. Drivers タブ内のオプション設定

Drivers タブを選択すると、アルテラ提供の JTAG UART/ UART ペリフェラルのデバイス・ドライバに関連したオプション設定が表示されます。

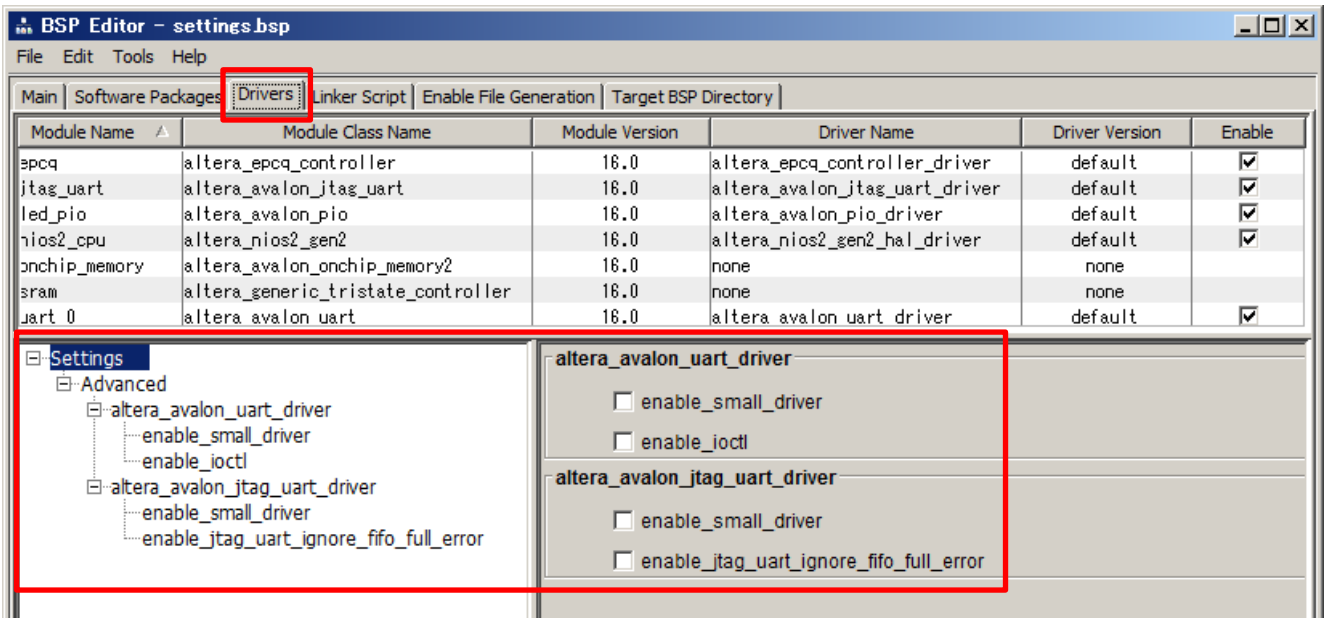

図 3-7. BSP Editor オプション設定⑥

3-3-3-1. enable_small_driver（altera_avalon_jtag_uart_driver ディレクトリ内）※上級者向け

このオプションにチェックを入れた場合、JTAG UART ペリフェラルのデバイス・ドライバに対して、コード・サイズやデータ・サイズが小さいポーリング・モードのドライバが適用されます。

3-3-3-2. enable_ioctl（altera_avalon_uart_driver ディレクトリ内）※上級者向け

このオプションにチェックを入れた場合、UART ペリフェラルのデバイス・ドライバの ioctl() のサポートを有効にします。この機能は small ドライバと互換性をもちません。UART ドライバに対する enable_small_driver や enable_reduced_device_drivers の設定のいずれかが無効の場合、ioctl() のサポート機能はコンパイルされません。

3-3-3-3. enable_small_driver（altera_avalon_uart_driver ディレクトリ内）※上級者向け

このオプションにチェックを入れた場合、UART ペリフェラルのデバイス・ドライバに対して、コード・サイズやデータ・サイズが小さいポーリング・モードのドライバが適用されます。

3-4. BSP Editor によるリンカ・スクリプトの設定例(Linker Script タブ)

Linker Script タブを選択すると、GUI によるパラメータ設定を使用して、リンカ・スクリプト linker.x に対する設定が行えます。

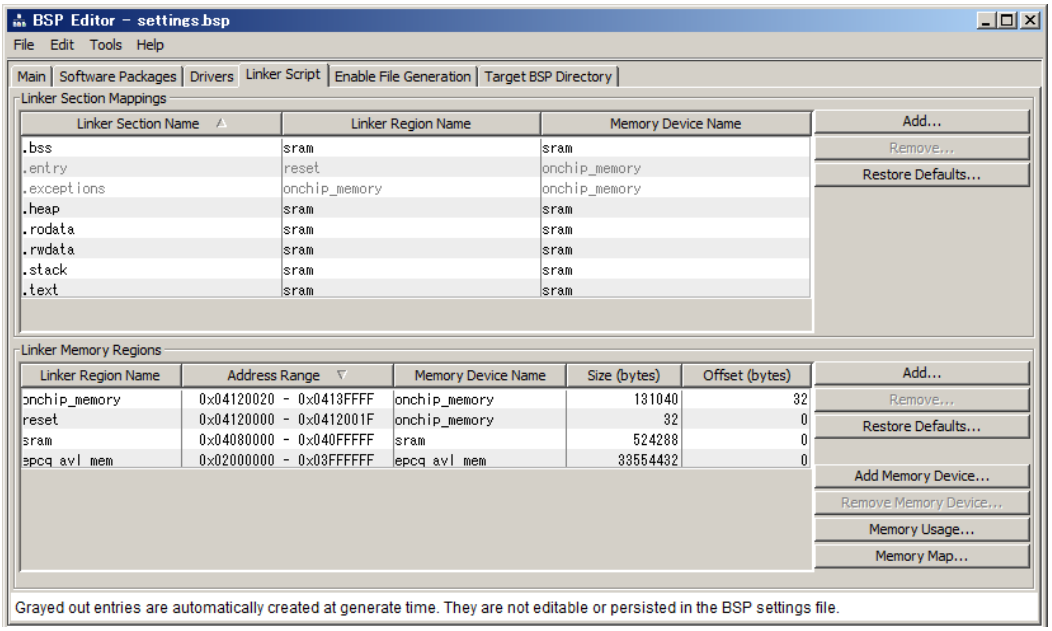

図 3-8. Linker Script タブ上のリンカ・スクリプト設定画面

右側の Memory Map ボタンをクリックすると、メモリ・マップを GUI で確認できます。

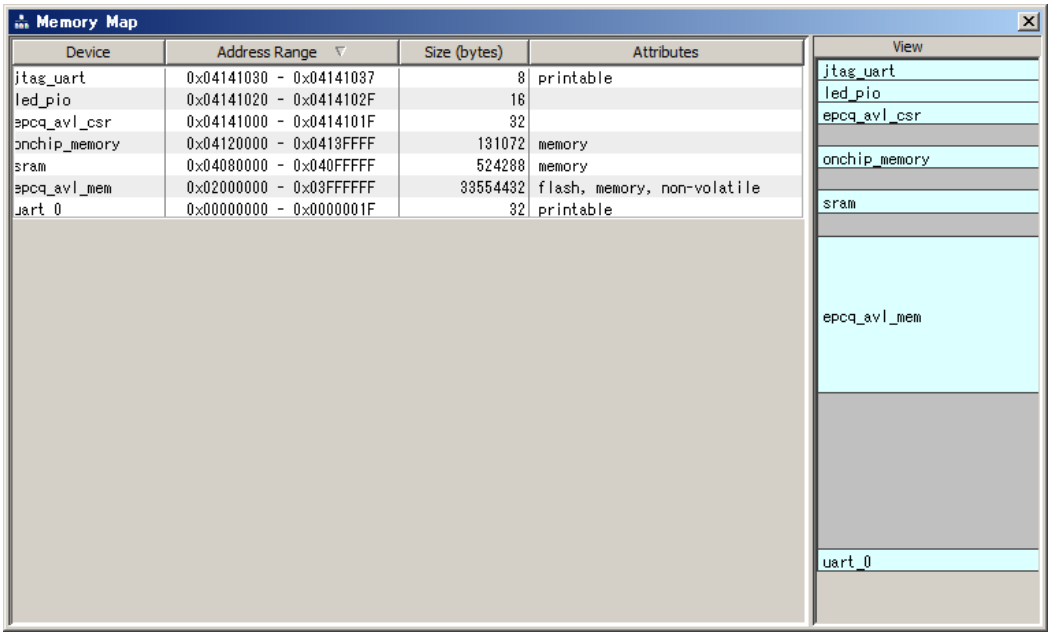

図 3-9. メモリ・マップの GUI 表示

以下に、カスタム例を紹介します。

※以降の BSP Editor の画面は、古いバージョンの画面です。最新バージョンでも同様の機能を実現できますので、その際の参考としてご覧ください。

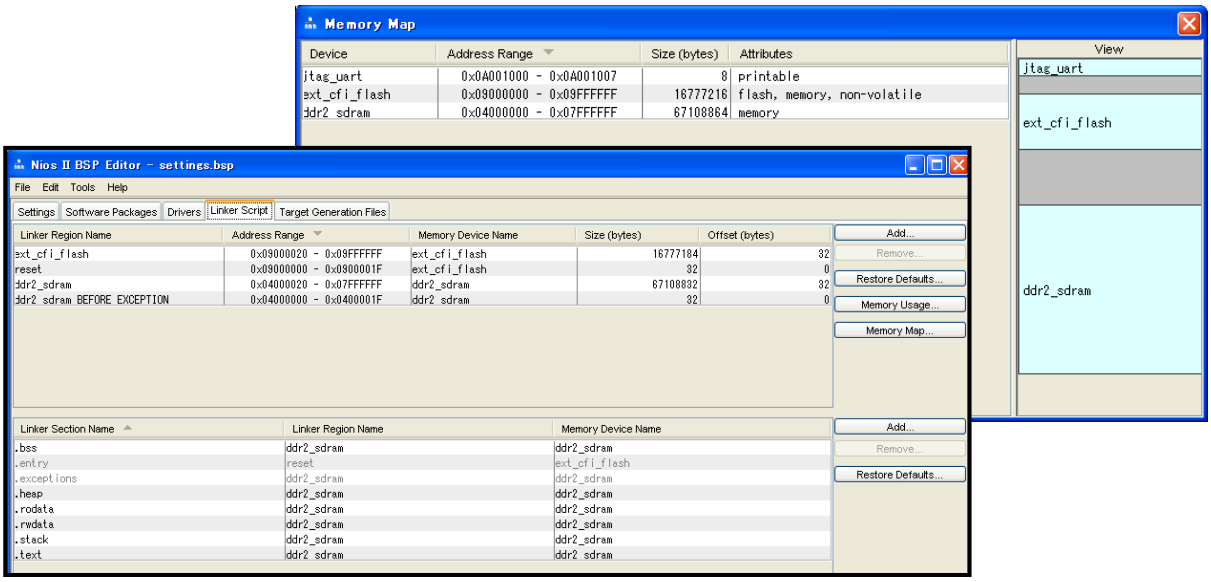

図 3-10. カスタム例（変更前）

図 3-10 では、Nios II プロセッサのワーク・メモリとして DR2-SDRAM を使用する構成です。ソフトウェアから見た場合、ワーク・メモリは前半の 32 バイトの領域（ddr2_sdram_BEFORE_EXCEPTION）と後半の 67108832 バイトの領域（ddr2_sdram）に分割されています。そして、後半の領域にプログラム・メモリ（.text）、リード専用メモリ（.rodata）、リード/ライト・メモリ（.rwdata）、BSS 領域（.bss）、ヒープ、スタックが配置されています。

後半の領域内に、Nios II 以外のマスタ・コントローラ（ハードウェア・ロジック）がアクセスする 16M バイトのメモリ領域を安全に確保する為に、メモリ・デバイスを、次のように、3 つの領域に分割できるように変更します。

① 32 バイト（領域名：ddr2_sdram_BEFORE_EXCEPTION）

② 50331616 バイト（領域名：ddr2_sdram）

③ 16777216 バイト（領域名：ext_data_memory）

Linker Region Name 欄の ddr2_sdram をハイライトして、右側の Memory Usage ボタンをクリックします。

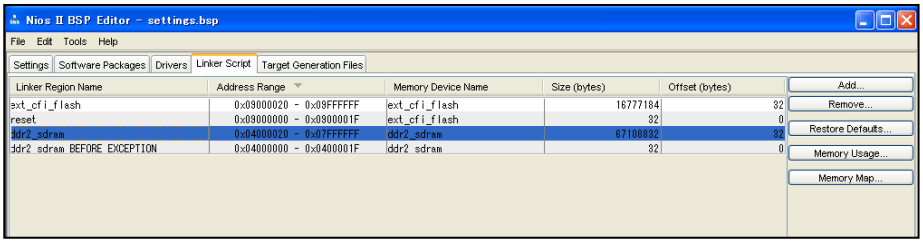

図 4-11. カスタム例①

図 3-11 でハイライトした領域がサイズ分だけ水色の枠で囲まれていることが確認できます。図 3-12 の場合、64M バイトの ddr2_sdram デバイスが 100% の使用率で 2 つの領域に分割されていることが把握できます。

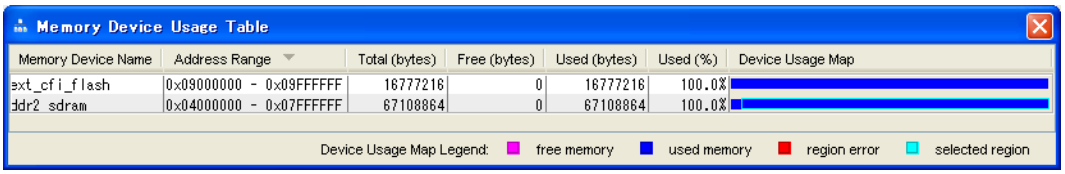

図 3-12. カスタム例②

図 3-12 の Memory Device Usage Table 画面をクローズ後、図 3-11 の画面に戻ったら、Remove ボタンをクリックして、ハイライトしている ddr2_sdram 領域を削除します。図 3-13 の画面が表示されたら、了解ボタンをクリックします。

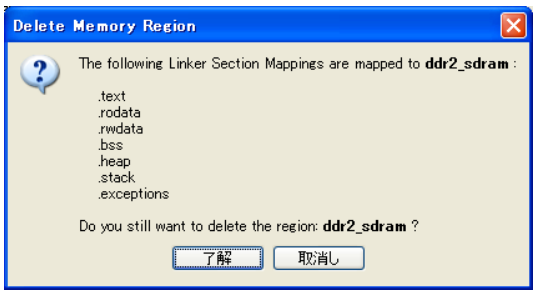

図 3-13. カスタム例③

削除すると、図 3-14 のようになります。右側の Memory Usage ボタンをクリックします。

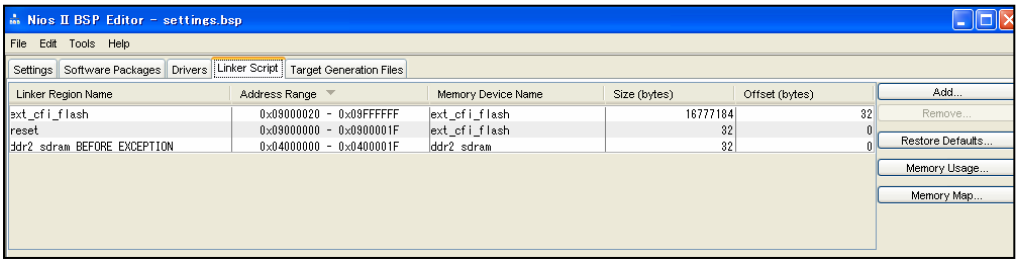

図 3-14. カスタム例③

削除したサイズ分だけ、未使用領域（free memory）として表示されていることが確認できます。

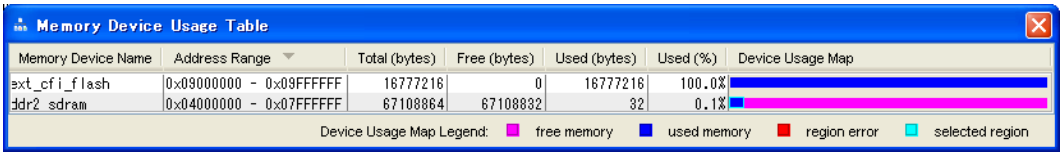

図 3-15. カスタム例④

次に Linker Region Name 欄の ddr2_sdram_BEFORE_EXCEPTION をハイライトした状態で、右クリックします。

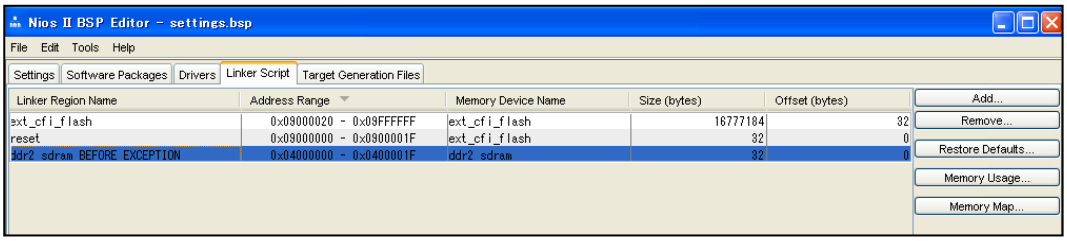

図 3-16. カスタム例⑤

図 3-17 のような Add Memory Region 画面が起動します。この画面で ddr2_sdram 領域の入れ直しを行います。Region Size 欄には、再登録するバイト数 5033161 を入力します。Memory Offset 欄には ddr2_sdram_BEFORE_EXCEPTION 領域に隣接するような値を設定します。

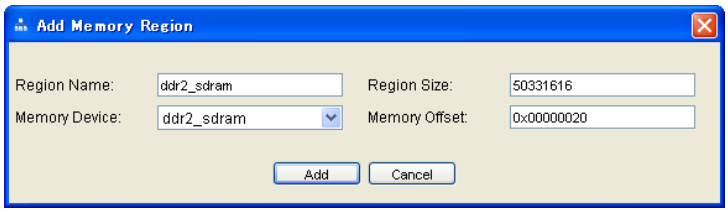

図 3-17. カスタム例⑥

図 3-18 のように、再設定した領域が反映されます。

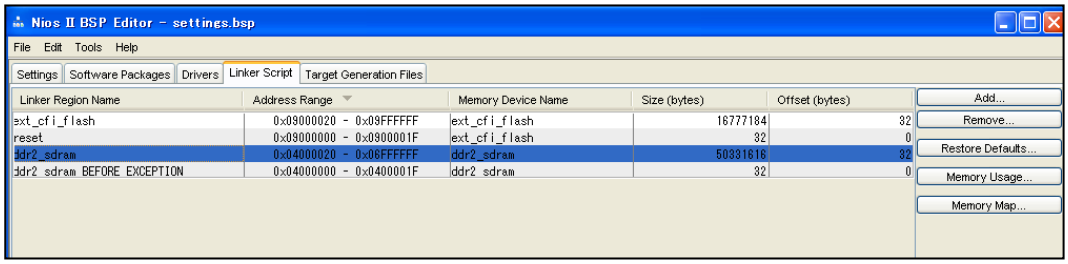

図 3-18. カスタム例⑦

Memory Usage ボタンをクリックして Memory Device Usage Table 画面を起動させます。図 3-19 では、再設定した領域がサイズ分だけ水色の枠で囲まれていて、この領域も含めてメモリ・デバイスの使用率が 75% を示していることが確認できます。この後、25% の未使用領域への新規設定を行います。

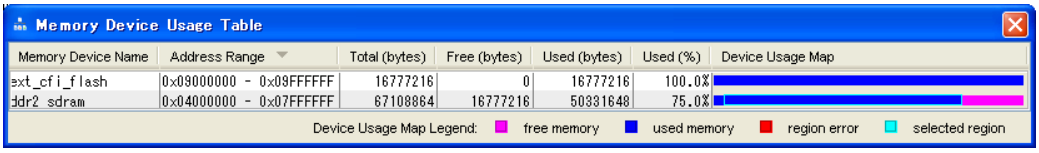

図 3-19. カスタム例⑧

Memory Device Usage Table 画面をクローズして、前述の図 3-18 のように、再設定した ddr2_sdram 領域をハイライトした状態から、右クリックより Add Memory Region 画面を起動させます。図 3-20 のように新規設定する領域名を Region Name 欄に指定します。Region Size 欄には、新規登録するバイト数 16777216 を入力します。Memory Offset 欄には再設定した ddr2_sdram に隣接するような値を設定します。

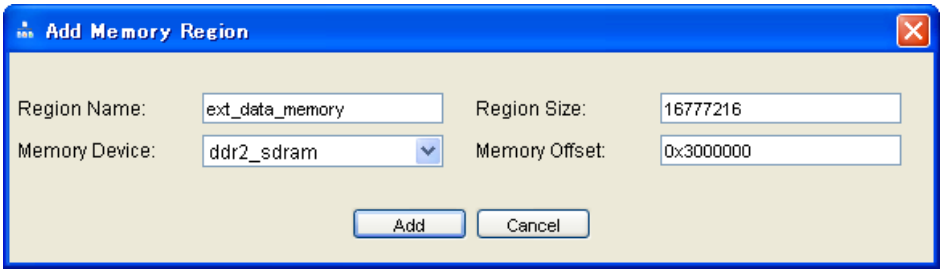

図 3-20. カスタム例⑨

図 3-21 のように、新規設定した領域 ext_data_memory が反映されていることが確認できます。また、新規設定した領域がサイズ分だけ水色の枠で囲まれていて、この領域も含めてメモリ・デバイスの使用率が 100% を示していることが確認できます。

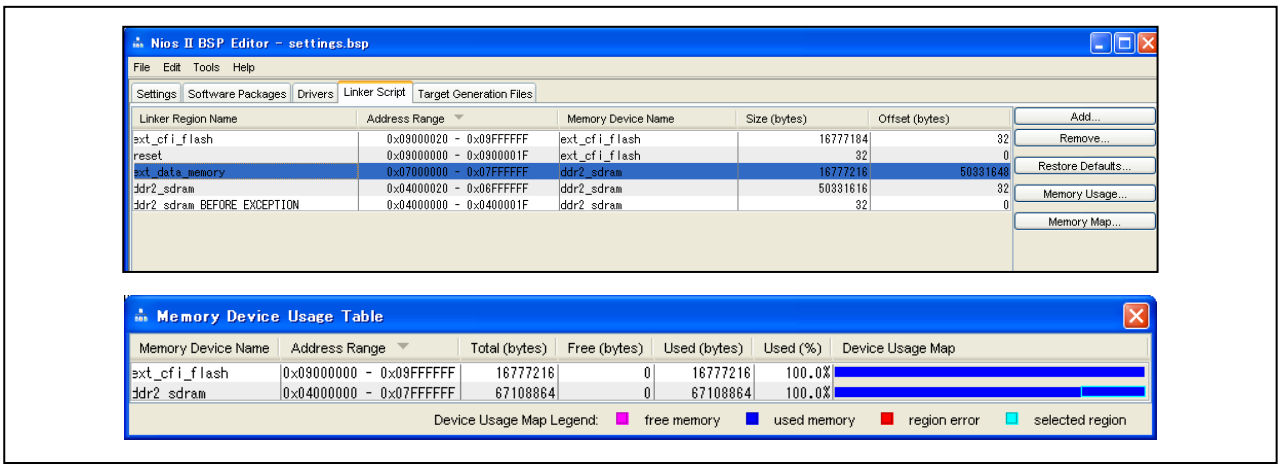

図 3-21. カスタム例⑩

次に、Linker Section Name 欄に、新規登録した領域に対して、新たにリンカ・セクションを定義します。右側の Add ボタンをクリックします。

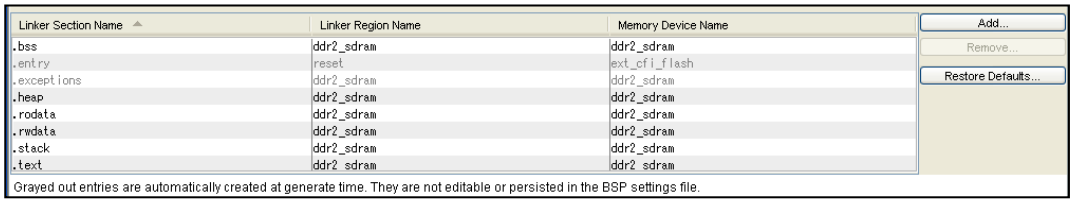

図 3-22. カスタム例⑪

Add Section Mapping 画面が起動します。図 3-23 のように、Section Name 欄には .ext_data_memory と入力して、Memory Region 欄のプルダウン・メニューから、上記の手順で新規登録した ext_data_memory 領域を指定します。

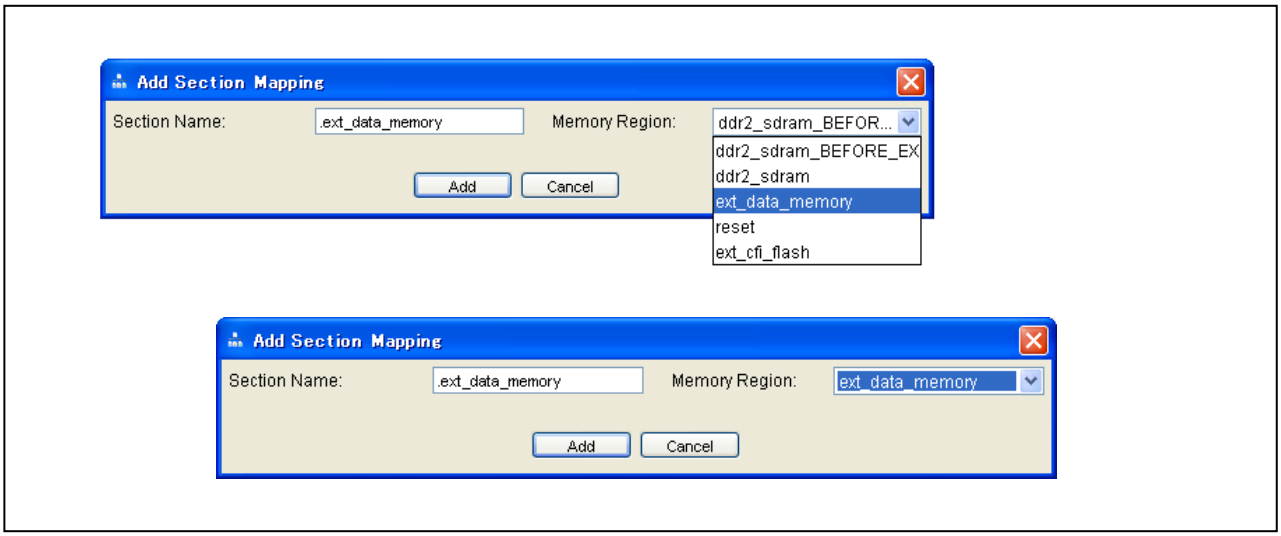

図 3-23. カスタム例⑫

最終的に、図 3-24 のように変更されます。マスタ・コントローラ（ハードウェア・ロジック）からは、安全に確保した領域名 ext_data_memory のアドレス範囲 0x07000000 ～ 0x07FFFFFF へのデータ・アクセスを行います。Nios II ソフトウェアからは、例えばアトリビュート機能を用いて、配列変数に ext_data_memory セクションを配置させることで、データ・アクセスを行います。

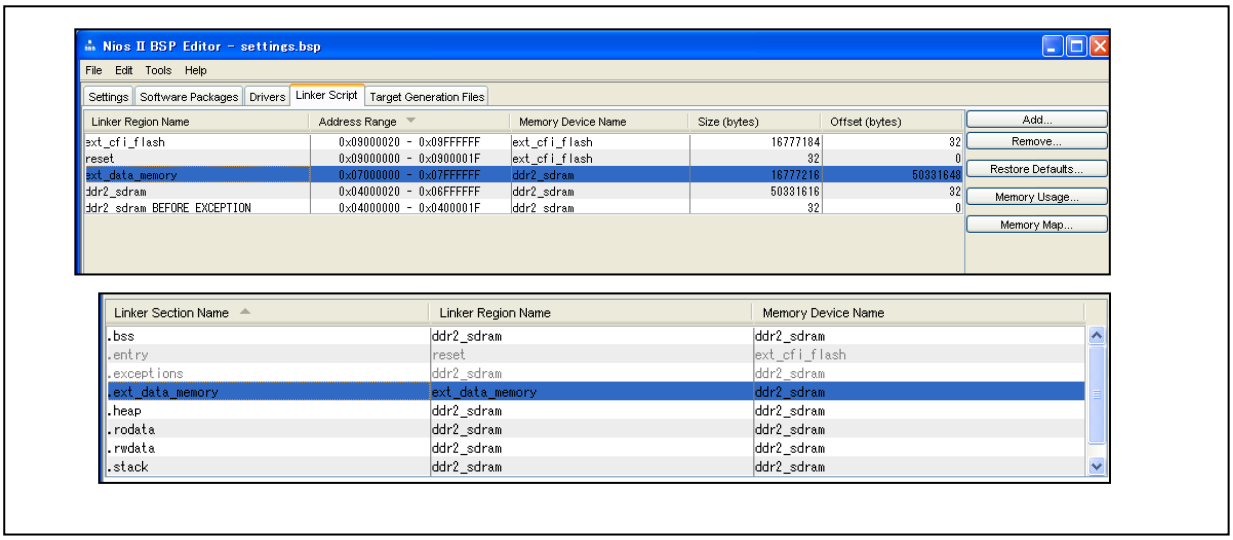

図 3-24. カスタム例（変更後）

4. おわりに

この資料では、多数のオプションについて解説していますが、目次を参照することによって、各オプションのインデックスとして利用できるようにしています。また、そのオプション機能が上級者向けかどうかもインデックスから把握できるようにしています。Nios II SBT を簡単に使いたいときには、オプション設定はデフォルトの状態で使用するのが得策です。

改版履歴

Revision	年月	概要
1	2016 年 8 月	初版