お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、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に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。お客様が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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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その総株主の議決権の過半数を直接または間接に保有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。

RENESAS

アプリケーション・ノート

V850E/ME2, V850E2/ME3

32ビット・シングルチップ・マイクロコンピュータ

ハードウエア編

V850E/ME2：
　μPD703111A
V850E2/ME3：
　μPD703500

資料番号　U16794JJ3V0AN00（第3版）
発行年月　March 2006 N　CP(K)

〔メ　モ〕

目次要約

CMOSデバイスの一般的注意事項

① 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デバイス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。入力レベルが固定な場合はもちろん，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等が入らないようご使用ください。

② 未使用入力の処理

CMOSデバイスの未使用端子の入力レベルは固定してください。

未使用端子入力については，CMOSデバイスの入力に何も接続しない状態で動作させるのではなく，プルアップかプルダウンによって入力レベルを固定してください。また，未使用の入出力端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介してV_{DD}またはGNDに接続することが有効です。

資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。

③ 静電気対策

MOSデバイス取り扱いの際は静電気防止を心がけてください。

MOSデバイス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，または導電性の緩衝材，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，MOSデバイス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。

④ 初期化以前の状態

電源投入時，MOSデバイスの初期状態は不定です。

電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。

リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。

⑤ 電源投入切断順序

内部動作および外部インタフェースで異なる電源を使用するデバイスの場合，原則として内部電源を投入した後に外部電源を投入してください。切断の際には，原則として外部電源を切断した後に内部電源を切断してください。逆の電源投入切断順により，内部素子に過電圧が印加され，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「電源投入切断シーケンス」についての記載のある製品については，その内容を守ってください。

⑥ 電源OFF時における入力信号

当該デバイスの電源がOFF状態の時に，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OFF時における入力信号」についての記載のある製品については，その内容を守ってください。

- 本資料に記載されている内容は2006年3月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品の使用に関連し発生した第三者の特許権，著作権その他の知的財産権の侵害等に関し，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，半導体製品の動作例，応用例を説明するものです。お客様の機器の設計において，回路，ソフトウエアおよびこれらに関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お客様または第三者に生じた損害に関し，当社は，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「特定水準」に分類しております。また，各品質水準は，以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認ください。

  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

  特別水準：輸送機器（自動車，電車，船舶等），交通用信号機器，防災・防犯装置，各種安全装置，生命維持を目的として設計されていない医療機器

  特定水準：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

  当社製品のデータ・シート，データ・ブック等の資料で特に品質水準の表示がない場合は，標準水準製品であることを表します。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNECエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいう。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E　02.11

はじめに

対 象 者　**このアプリケーション・ノートは，**V850E/ME2, V850E2/ME3**の機能を理解し，それらを使用した応用システムを設計するユーザを対象とします。対象製品を次に示します。**

- V850E/ME2 ：μPD703111A
- V850E2/ME3：μPD703500

目　　的　**このアプリケーション・ノートでは**V850E/ME2**を用いたシステムの例として「**CPU**ボード（**TB-V850E/ME2**）」を取り上げ，その構成をユーザに理解していただくことを目的としています。なお，**V850E2/ME3**を用いたシステムでも基本的には同じ回路構成となります。**

構　　成　**このアプリケーション・ノートは大きく分けて次の内容で構成しています。**

●イントロダクション
●バス・インタフェース接続回路例
●内蔵周辺機能の接続回路例
●アプリケーション例

読 み 方　**このマニュアルの読者には，電気，論理回路，およびマイクロコンピュータに関する一般知識を必要とします。**

V850E/ME2, V850E2/ME3**のハードウエア機能，および電気的特性を知りたいとき**
→別冊のV850E/ME2　**ユーザーズ・マニュアル　ハードウエア編，**V850E2/ME3　**ユーザーズ・マニュアル　ハードウエア編を参照してください。**

V850E/ME2, V850E2/ME3**の命令機能を知りたいとき**
→別冊のV850E1　**ユーザーズ・マニュアル　アーキテクチャ編，**V850E2　**ユーザーズ・マニュアル　アーキテクチャ編を参照してください。**

本文欄外の★印は，本版で改訂された主な箇所を示しています。
この"★"をPDF**上でコピーして「検索する文字列」に指定することによって，改版箇所を容易に検索できます。**

凡　例　**データ表記の重み**　：**左が上位桁，右が下位桁**

アクティブ・ロウの表記　：$\overline{xxx}$（**端子，信号名称に上線**）**または**

／xxx（**信号名称の前に“／”記号**）

メモリ･マップのアドレス　：**上部－上位，下部－下位**

注　：**本文中に付けた注の説明**

注意　：**気を付けて読んでいただきたい内容**

備考　：**本文の補足説明**

数の表記　：2**進数**　…　xxxxまたはxxxxB

10**進数**　…　xxxx

16**進数**　…　xxxxH

2**のべき数を示す接頭語（アドレス空間，メモリ容量）**　：K（**キロ**）　…2^{10} = 1024

M（**メガ**）　…2^{20} = 1024^2

G（**ギガ**）　…2^{30} = 1024^3

関連資料　**関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。**
あらかじめご了承ください。

V850E/ME2, V850E2/ME3**に関する資料**

資　料　名	資料番号
V850E1　**ユーザーズ・マニュアル　アーキテクチャ編**	U14559J
V850E/ME2　**ユーザーズ・マニュアル　ハードウエア編**	U16031J
V850E/ME2　**アプリケーション・ノート　ハードウエア編**	**このマニュアル**
V850E/ME2　**アプリケーション・ノート**　USB**ファンクション・ドライバ編**	U17069J
V850E2　**ユーザーズ・マニュアル　アーキテクチャ編**	U17135J
V850E2/ME3　**ユーザーズ・マニュアル　ハードウエア編**	U17126J
V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2　**アプリケーション・ノート**　PCI**ホスト・ブリッジ・マクロ編**	U17121J

開発ツールに関する資料（ユーザーズ・マニュアル）

対応品名	資　料　名		資料番号
V850E/ME2**のみ**	CA850（Ver.3.00） **（C**コンパイラ・パッケージ**）**	**操作編**	U17293J
		C**言語編**	U17291J
		アセンブリー言語編	U17292J
		リンク・ディレクティブ編	U17294J
	PM+（Ver.6.00）**（プロジェクト・マネージャ）**		U17178J
	ID850（Ver.3.00）**（統合デバッガ）**	**操作編**	U17358J
	ID850NW（Ver.3.00, 3.10） **（統合デバッガ）**	**操作編**	U17369J
	TW850（Ver.2.00）**（性能解析チューニング・ツール）**		U17241J
V850E/ME2， V850E2/ME3	RX850（Ver.3.20） **（リアルタイム**OS**）**	**基礎編**	U13430J
		インストレーション編	U17419J
		テクニカル編	U13431J
		タスク・デバッガ編	U17420J
	RX850 Pro（Ver.3.20） **（リアルタイム**OS**）**	**基礎編**	U13773J
		インストレーション編	U17421J
		テクニカル編	U13772J
		タスク・デバッガ編	U17422J
	RX-NET（Ver.3.15）（TCP/IP**ライブラリ**）		U15083J
	AZ850（Ver.3.30）**（システム・パフォーマンス・アナライザ）**		U17423J

目　次

第1章　イントロダクション

V850E/ME2, V850E2/ME3は，NECエレクトロニクスのシングルチップ・マイクロコンピュータ「V850シリーズ」の1製品です。

1.1 概　　説

★　V850E/ME2はV850シリーズの「V850E1 CPU」を，V850E2/ME3は「V850E1 CPU」の性能をさらに強化した「V850E2 CPU」を使用し，キャッシュ，データRAM，命令RAM，および，各種メモリ・コントローラ，DMAコントローラ，タイマ／カウンタ，シリアル・インタフェース，USBファンクション・コントローラ（USBF），A/Dコンバータなどの周辺機能を内蔵し，大容量データ処理と高度なリアルタイム制御を実現した32ビット・シングルチップ・マイクロコンピュータです。

表1－1にV850E/ME2とV850E2/ME3の製品間の違いを示します。

★　**表1－1　V850E/ME2とV850E2/ME3の製品間の違い（1/2）**

項　　目		V850E/ME2	V850E2/ME3
命令数		80	89
最小命令実行時間		6.5 ns（内部150 MHz動作時）	5.0 ns（内部200 MHz動作時）
ブート・アドレス		0x100000H番地（外部メモリ）	0x000000H番地（外部メモリ）
メモリ空間	リニア・アドレス空間	256 Mバイト （プログラム：64 Mバイト／データ：256 Mバイト）	512 Mバイト （プログラム：512 Mバイト／データ：512 Mバイト）
	メモリ・ブロック分割機能	2 M, 4 M, 6 M, 8 M, 64 Mバイト／ブロック	2 M, 64 Mバイト／ブロック
外部バス・インタフェース	外部バス分周機能	内部システム・クロック（f_{CLK}）に対するBUSCLKの分周比： 1/1, 1/2, 1/3, 1/4（66 MHz MAX.）	VBCLK（f_{VBCLK}）に対するBUSCLKの分周比： 1/1, 1/2, 1/4（66 MHz MAX.）
	先読み機能	あり	なし
	エンディアン機能	あり	なし
命令RAM	容量	128 Kバイト（64 Kバイト×2バンク）	168 Kバイト（168 Kバイト×1バンク）
	リード／ライト・モードの切り替え	IRAMMレジスタ	IRWEレジスタ
データRAM		16 Kバイト	32 Kバイト
命令キャッシュ	容量	8 Kバイト （2ウエイ・セット・アソシアティブ）	8 Kバイト （4ウエイ・セット・アソシアティブ）
	ウエイ・ロック機能	あり（1ウエイのみロック可能）	なし
データ・キャッシュ		なし	あり（8 Kバイト（4ウエイ・セット・アソシアティブ））
割り込み／例外	リセット解除後のマスク機能	あり（NRSレジスタでNMI入力を確認）	なし

備考　詳細は，各製品のユーザーズ・マニュアル　ハードウエア編を参照してください。

★ **表1－1　V850E/ME2とV850E2/ME3の製品間の違い（2/2）**

項　目			V850E/ME2	V850E2/ME3
DMAコントローラ	最大転送回数		65536（2^{16}）回	16777216（2^{24}）回
	シングルステップ転送モードの設定		DADCnレジスタ	DADCn, DSMCレジスタ
	DRSTレジスタ		なし	あり
	DMAチャネルの優先順位		固定モード	固定モード／ラウンドロビン・モード
	$\overline{\text{DMARQn}}$信号のマスク機能		3クロック固定	0-15クロックから選択
	$\overline{\text{DMAAKn}}$信号アクティブ幅拡張機能		4, 6, 7クロックから選択	0-15クロックから選択
タイマ	TMC0-TMC5		あり	あり
	TMD0-TMD3		あり	あり
	TMENC10, TMENC11		あり（ノイズ除去クロック数：0, 2, 3, 5）	あり（ノイズ除去クロック数：0, 2, 3, 7）
シリアル・インタフェース	UARTB0, UARTB1		あり	あり
	CSI30, CSI31		あり（転送速度：最大5.5 Mbps）	あり（転送速度：最大6.25 Mbps）
	USB ファンクション・コントローラ		あり	あり
クロック･ジェネレータ	SSCG出力クロック		8逓倍固定	20逓倍固定
	メモリ・コントローラ		BUSCLK動作	VBCLK動作 （VBCSEL端子で分周比（1/2, 1/3）を設定）
	OSTSレジスタ		あり	なし
パワー・セーブ機能			HALT／IDLE／ソフトウエアSTOPモード	HALT／IDLEモード
ピン配置	ピン番号	1	JIT1端子	EV_{SS}端子
		2	JIT0端子	EV_{SS}端子
		156	PLLSEL端子	VBCSEL端子
パッケージ			176ピン・プラスチックLQFP （パッケージ厚：1.4 mm）	176ピン・プラスチックQFP （パッケージ厚：2.7 mm）

備考1.　詳細は，各製品のユーザーズ・マニュアル　ハードウエア編を参照してください。

2.　n = 0-3

1. 2　V850E/ME2

1. 2. 1　特　　徴

★　○命令数　80

○最小命令実行時間　10 ns/7.5 ns/6.7 ns（内部100 MHz/133 MHz/150 MHz動作時）

○汎用レジスタ　32ビット×32本

○命令セット　V850E1 CPU
符号付き乗算（16ビット×16ビット→32ビット，または32ビット×32ビット→64ビット）：1-2クロック
飽和演算命令（オーバフロー／アンダフロー検出機能付き）
32ビット・シフト命令：1クロック
ビット操作命令
ロング／ショート形式を持つロード／ストア命令
符号付きロード命令

○メモリ空間　256 Mバイト・リニア・アドレス空間（プログラム／データ共有）
チップ・セレクト出力機能：8空間
メモリ・ブロック分割機能：2 M, 4 M, 6 M, 8 M, 64 Mバイト／ブロック
プログラマブル・ウエイト機能
アイドル・ステート挿入機能

○外部バス・インタフェース
32ビット・データ・バス（アドレス／データ分離型バス）
32/16/8ビット・バス・サイジング機能
外部バス分周機能：1/1,1/2,1/3,1/4（66 MHz（MAX.））
バス・ホールド機能
外部ウエイト機能
アドレス・セットアップ・ウエイト機能
エンディアン制御機能

○内蔵メモリ　命令RAM：128 Kバイト
データRAM：16 Kバイト

○命令キャッシュ　8 Kバイト2ウエイ・セット・アソシアティブ

○割り込み／例外　外部割り込み　：40本（NMI含む）
内部割り込み　：59要因
例外　：2要因
8レベルの優先順位指定可能

○メモリ・アクセス制御

DRAMコントローラ（SDRAMに対応）
ページROMコントローラ
先読み／ライト・バッファ機能

○DMAコントローラ

4チャネル構成
転送単位　　：8ビット／16ビット／32ビット
最大転送回数：65,536（2^{16}）回
転送タイプ　：フライバイ（1サイクル）転送／2サイクル転送
転送モード　：シングル転送／シングルステップ転送／ブロック転送
転送対象　　：メモリ←→メモリ，メモリ←→I/O
転送要求　　：外部要求／内蔵周辺I/O／ソフトウエア
DMA転送終了（ターミナル・カウント）出力信号
ネクスト・アドレス設定機能

○I/Oライン　　入力ポート　：1
入出力ポート：77

○タイマ機能

16ビット・タイマ／イベント・カウンタ：6 ch（2 chはキャプチャ動作なし）
　16ビット・タイマ：6本
　16ビット・キャプチャ／コンペア・レジスタ：12本
16ビット・インターバル・タイマ：4 ch
16ビット2相エンコーダ入力用アップ/ダウン・カウンタ／汎用タイマ：2 ch
　16ビット・キャプチャ／コンペア・レジスタ：4本
　16ビット・コンペア・レジスタ：4本

○シリアル・インタフェース

アシンクロナス・シリアル・インタフェースB（UARTB）
クロック同期式シリアル・インタフェース3（CSI3）
　CSI3/UARTB　：1 ch
　UARTB　　：1 ch
　CSI3　　：1 ch
　USBファンクション・コントローラ（USBF）：1 ch
　　フルスピード（12 Mbps）
　　エンドポイント　　コントロール転送：64バイト×2本
　　インタラプト転送：8バイト×2本
　　バルク転送（IN）：64バイト×2バンク×2本
　　バルク転送（OUT）：64バイト×2バンク×2本

○A/Dコンバータ　10ビット分解能A/Dコンバータ：8 ch

○PWM（Pulse Width Modulation）

16ビット分解能PWM：2 ch

○クロック・ジェネレータ

SSCGによる8逓倍機能

○パワー・セーブ機能

HALT／IDLE／ソフトウエアSTOPモード

★　○パッケージ　176ピン・プラスチックLQFP（ファインピッチ）（24×24）

○CMOS構造　完全スタティック回路

★　1. 2. 2　オーダ情報

品　名	パッケージ	最高動作周波数
μPD703111AGM-10-UEU	176ピン・プラスチックLQFP（ファインピッチ）（24×24）	100 MHz
μPD703111AGM-10-UEU-A	〃	〃
μPD703111AGM-13-UEU	〃	133 MHz
μPD703111AGM-13-UEU-A	〃	〃
μPD703111AGM-15-UEU	〃	150 MHz
μPD703111AGM-15-UEU-A	〃	〃

備考　オーダ名称末尾「-A」の製品は，鉛フリー製品です。

★　1. 2. 3　端子接続図

・176ピン・プラスチックLQFP（ファインピッチ）（24×24）

μPD703111AGM-10-UEU

μPD703111AGM-10-UEU-A

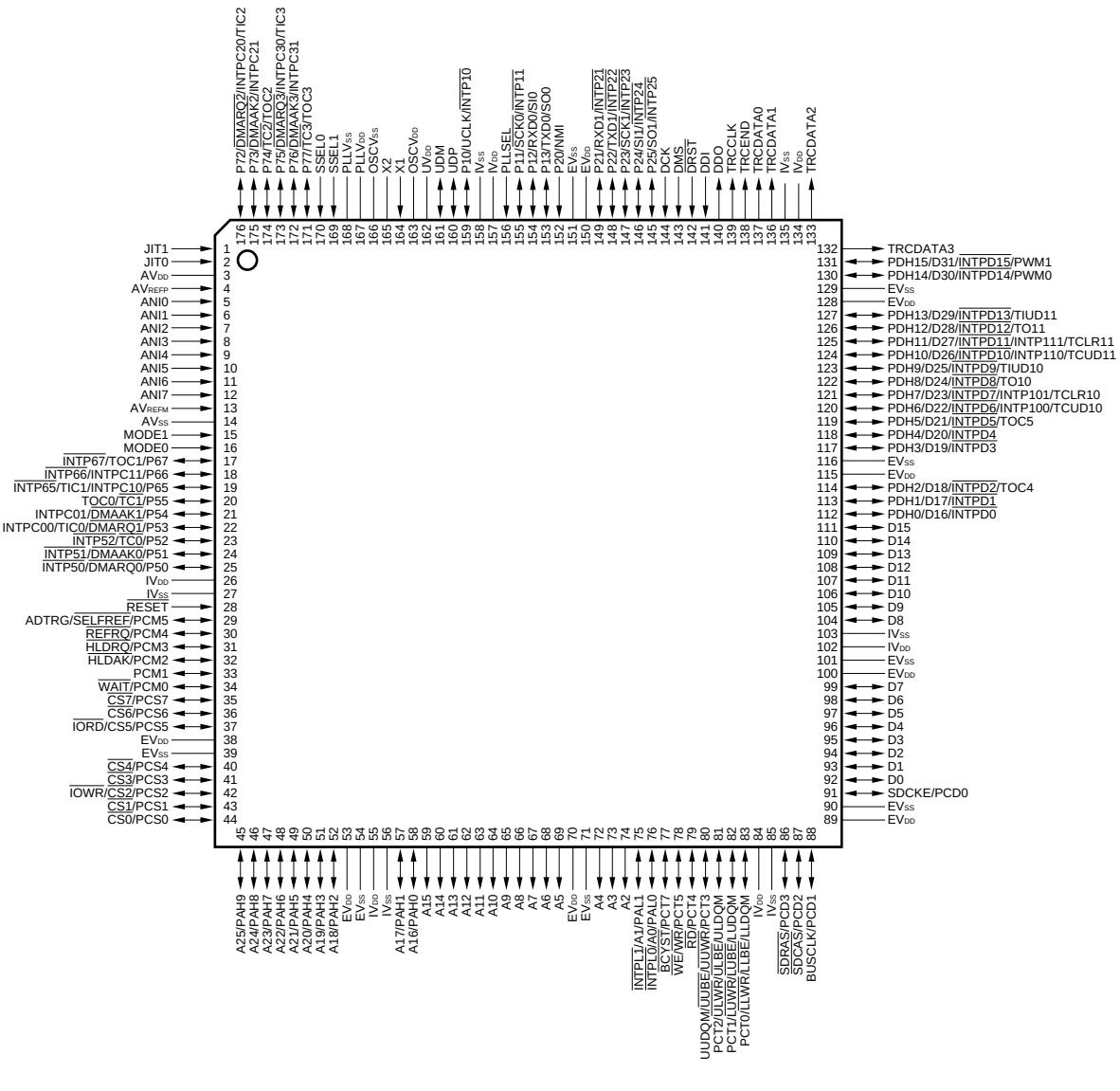
μPD703111AGM-13-UEU

μPD703111AGM-13-UEU-A

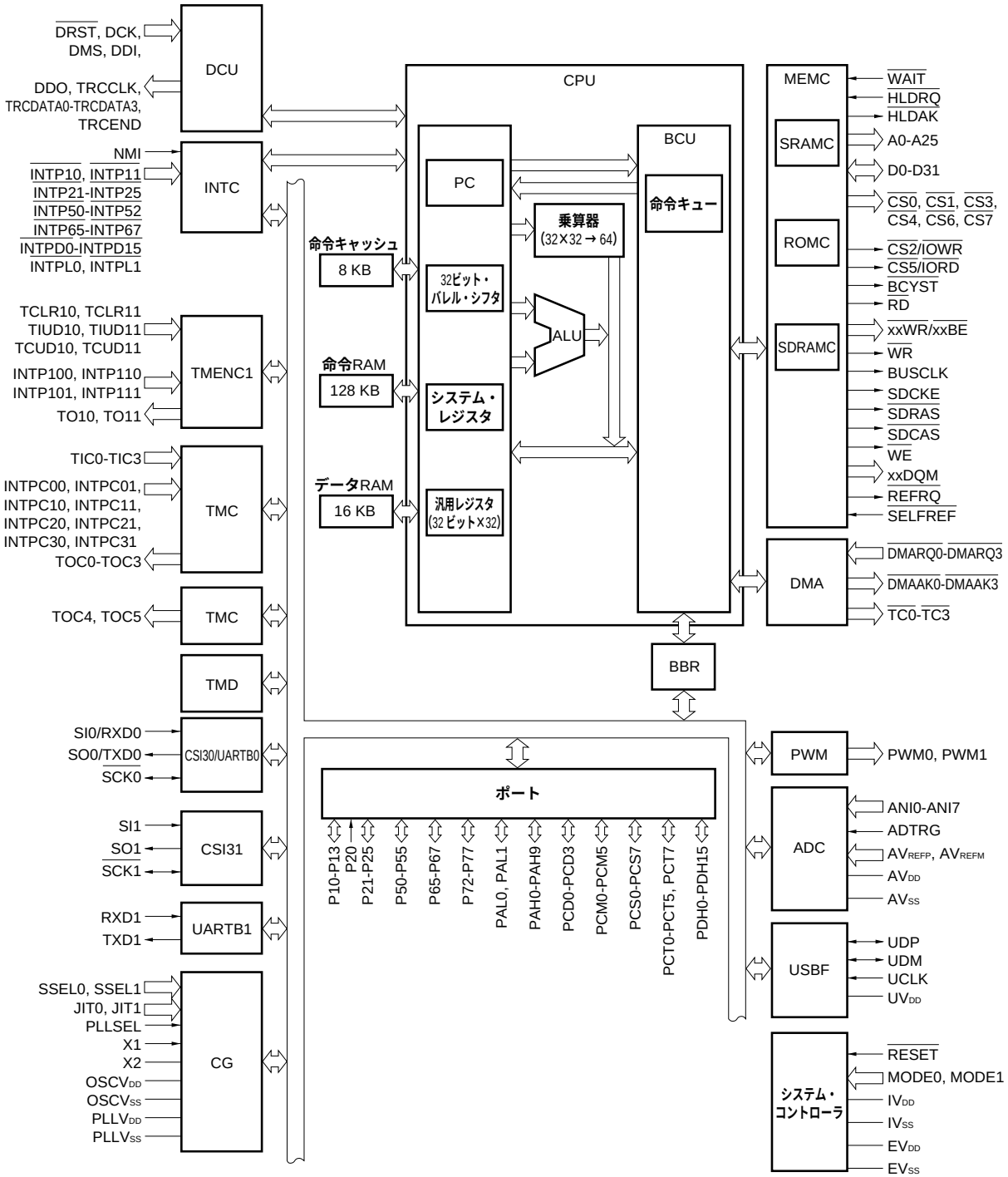
μPD703111AGM-15-UEU

μPD703111AGM-15-UEU-A

Top View



1. 2. 4　内部ブロック図



備考　xx：LL, LU, UL, UU

1. 2. 5　内蔵メモリ

V850E/ME2は，128 Kバイト（64 Kバイト×2バンク）の命令RAM，8 Kバイト（2ウエイ・セット・アソシアティブ）の命令キャッシュ，16 KバイトのデータRAMを内蔵しています。

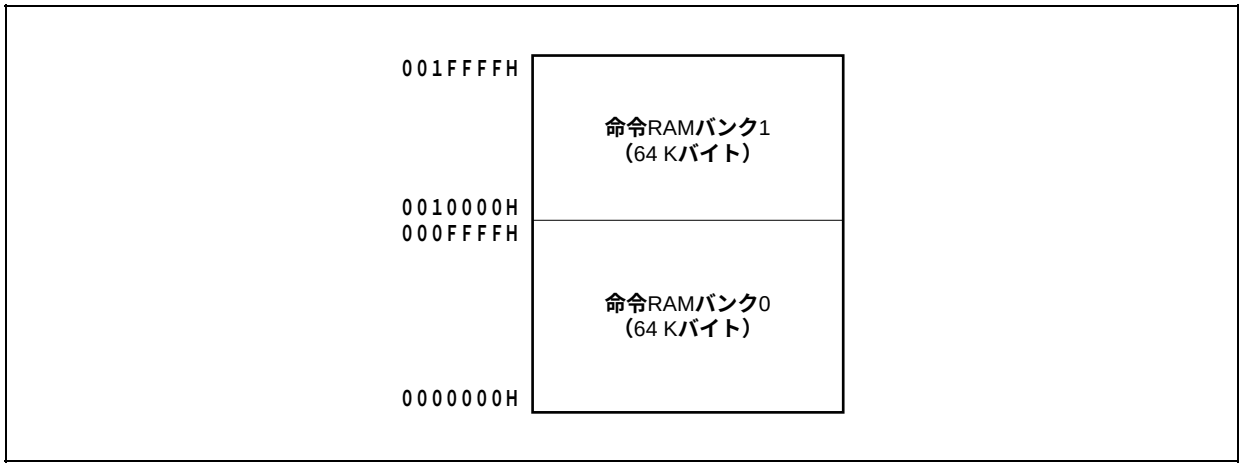
（1）内蔵命令RAM

内蔵命令RAMには，リード・モードとライト・モードの2つのモードがあり，内蔵命令RAMモード・レジスタ（IRAMM）で設定します。

リセット後はライト・モードに初期化されます。したがって，プログラムまたはDMAコントローラにより，内蔵命令RAMへ命令データを転送してからリード・モードを設定します。また，リード・モード時には，内部システム・クロック×1クロックで命令をフェッチします。

注意　内蔵命令RAM**バンク0には，リセット以外のすべての割り込み／例外ハンドラが配列されています。内蔵命令**RAM**バンク0へのライト動作が完了するまでは，すべての割り込み／例外を発生させないようにしてください。**

図1－1　内蔵命令RAM**のメモリ・マップ**



(2) 命令キャッシュ機能

$\overline{CS0}$-$\overline{CS2}$空間がキャッシャブル領域です。キャッシュ・コンフィギュレーション・レジスタ（BHC）により，各空間をキャッシュ可能領域にするか，キャッシュ不可能領域にするかを設定します。また，命令キャッシュ・コントロール・レジスタ（ICC）により，ウエイ0のキャッシュ・ロック状態，ウエイ0のオートフィル，ウエイ0, 1のタグ・クリアを設定します。

注意1. BHCレジスタへの書き込みはリセット後に行ってください。書き込み後は，値を変更しないでください。

2. 通常のシステムでは，$\overline{CS0}$-$\overline{CS2}$空間に配列したメモリは，キャッシュ領域で使用します。ブート・プログラムによりプログラムをダウンロードするようなシステムでは，ダウンロードが完了してからキャッシュ可能領域に設定します。ただし，BHCレジスタの設定を行う命令自身が存在する領域をキャッシュ不可領域→キャッシュ可能領域，またはキャッシュ可能領域→キャッシュ不可領域にすることはできません。キャッシュ可能領域に設定する場合は，キャッシュ不可領域または内蔵命令RAM領域で行ってください。

(3) 内蔵データRAM

内蔵データRAM領域は，FFFB000H-FFFEFFFH番地の16 Kバイトの領域に実装されています。命令コードは，配列できません。

1. 2. 6　先読み機能（リード・バッファ機能）

V850E/ME2は，CPUの処理を高速に行うために4ワード分（128ビット）の先読み機能用リード・バッファを内蔵しています。ライン・バッファ・コントロール・レジスタ0, 1（LBC0, LBC1）により，$\overline{CSn}$空間ごとに先読みタイミングを設定できます（n = 0-7）。

注意　一般的に連続したアドレスに配列されないユニット（I/Oデバイスなど）や，CPUと非同期に内容が変化するメモリ（他のバス・マスタからライトされるデュアル・ポート・メモリなど）は，先読み機能を禁止します。

1.2.7　初期設定端子

V850E/ME2は，さまざまな動作モードを決定する初期設定端子があります。

（1）MODE0, MODE1端子

データ・バスの動作モードを指定する入力端子です。MODE0, MODE1端子の指定は応用システムにおいて固定とし，動作中に変更した場合の動作は保証しません。

表1－2　データ・バスの設定

MODE1	MODE0	動作モード	
L	L	通常動作モード	32ビット・データ・バス
L	H		16ビット・データ・バス
上記以外		設定禁止	

備考　L：ロウ・レベル入力
　　H：ハイ・レベル入力

（2）PLLSEL, SSEL0, SSEL1端子

X1, X2端子への入力周波数（Fx）に応じて設定する入力端子です。Fx×8 = fx（メイン・クロック）の値に応じて，PLLSEL, SSEL0, SSEL1端子を設定してください。

表1－3　周波数一覧

逓倍数	PLLSEL端子	SSEL1端子	SSEL0端子	入力周波数（MHz）（目標値）	メイン・クロック（fx）周波数（MHz）
8	H	L	H	設定禁止	設定禁止
		H	L	10.00～10.19	80.00～81.59
		H	H	10.20～11.99	81.60～95.99
	L	L	L	12.00～14.39	96.00～115.19
		L	H	14.40～17.39	115.20～139.19
		H	L	17.40～18.75	139.20～150.00
		H	H	設定禁止	設定禁止

注意　f_{CLK}のMAX.値は100 MHz品の場合は100 MHz, 133 MHz品の場合は133 MHz, 150 MHz品の場合は150 MHzです。
f_{CLK}（MAX.）＜fxとなる構成で使用した場合の動作は保証しません。
fxは各製品の動作保証周波数を越えない値にしてください。

備考　L：ロウ・レベル入力
　　H：ハイ・レベル入力

(3) JIT0, JIT1端子

SSCG出力の周波数変調率（f_{DIT}）を指定する入力端子です。JIT0, JIT1端子の設定により，SSCGCレジスタのADJON, ADJ2-ADJ0ビットの初期値（リセット後）は次のようになります。

表1－4　SSCGC.ADJON, ADJ2-ADJ0ビットの初期値

JIT1端子	JIT0端子	初期値			
		ADJONビット	ADJ2ビット	ADJ1ビット	ADJ0ビット
L	L	0	0	0	0
L	H	1	0	0	1
H	L	1	0	1	1
H	H	1	1	0	1

備考　L：ロウ・レベル入力

H：ハイ・レベル入力

★ 1. 3　V850E2/ME3

1. 3. 1　特　　徴

○命令数　　89

○最小命令実行時間　　5.0 ns（内部200 MHz動作時）

○汎用レジスタ　　32ビット×32本

○命令セット　　V850E2 CPU
符号付き乗算（16ビット×16ビット→32ビット，または32ビット×32ビット→64ビット）：1クロック
飽和演算命令（オーバフロー／アンダフロー検出機能付き）
32ビット・シフト命令：1クロック
ビット操作命令
ロング／ショート形式を持つロード／ストア命令
符号付きロード命令
積和演算関数（MAC）（32ビット×32ビット＋64ビット→64ビット）

○メモリ空間　　512 Mバイト・リニア・アドレス空間（プログラム／データ共有）
チップ・セレクト出力機能：8空間
メモリ・ブロック分割機能：2 M, 64 Mバイト／ブロック
プログラマブル・ウエイト機能
アイドル・ステート挿入機能

○外部バス・インタフェース
32ビット・データ・バス（アドレス／データ分離型バス）
32/16/8ビット・バス・サイジング機能
外部バス分周機能：1/1,1/2,1/4（66 MHz MAX.）
バス・ホールド機能
外部ウエイト機能
アドレス・セットアップ・ウエイト機能

○内蔵メモリ　　命令RAM：168 KB
データRAM：32 KB

○命令キャッシュ　　8 Kバイト4ウエイ・セット・アソシアティブ

○データ・キャッシュ　　8 Kバイト4ウエイ・セット・アソシアティブ

○割り込み／例外　外部割り込み　：40本（NMI含む）
内部割り込み　：59要因
例外　：2要因
8レベルの優先順位指定可能

○メモリ・アクセス制御
DRAMコントローラ（SDRAMに対応）
ページROMコントローラ

○DMAコントローラ
4チャネル構成
転送単位　：8ビット／16ビット／32ビット
最大転送回数　：16,777,216（224）回
転送タイプ　：フライバイ（1サイクル）転送／2サイクル転送
転送モード　：シングル転送／シングルステップ転送／ブロック転送
転送対象　：メモリ←→メモリ，メモリ←→I/O
転送要求　：外部要求／内蔵周辺I/O／ソフトウエア
DMA転送終了（ターミナル・カウント）出力信号
ネクスト・アドレス設定機能
DMAチャネルの優先順位：優先順位固定モード／ラウンドロビン・モード

○I/Oライン　入力ポート　：1
入出力ポート：77

○タイマ機能
16ビット・タイマ／イベント・カウンタ：6 ch（2 chはキャプチャ動作なし）
16ビット・タイマ：6本
16ビット・キャプチャ／コンペア・レジスタ：12本
16ビット・インターバル・タイマ：4 ch
16ビット2相エンコーダ入力用アップ／ダウン・カウンタ／タイマ：2 ch
16ビット・キャプチャ／コンペア・レジスタ：4本
16ビット・コンペア・レジスタ：4本

○シリアル・インタフェース
アシンクロナス・シリアル・インタフェースB（UARTB）
クロック同期式シリアル・インタフェース3（CSI3）
CSI3/UARTB　：1 ch
UARTB　：1 ch
CSI3　：1 ch
USBファンクション・コントローラ（USBF）：1 ch
フルスピード（12 Mbps）
エンドポイント　コントロール転送：64バイト×2本
インタラプト転送：8バイト×2本

バルク転送（IN）：64バイト×2バンク×2本
バルク転送（OUT）：64バイト×2バンク×2本

○A/Dコンバータ　10ビット分解能A/Dコンバータ：8 ch

○PWM（Pulse Width Modulation）
16ビット分解能PWM：2 ch

○クロック・ジェネレータ
SSCGによる20逓倍機能

○パワー・セーブ機能
HALT／IDLEモード

○パッケージ　176ピン・プラスチックQFP（ファインピッチ）（24×24）

○CMOS構造　完全スタティック回路

1. 3. 2　オーダ情報

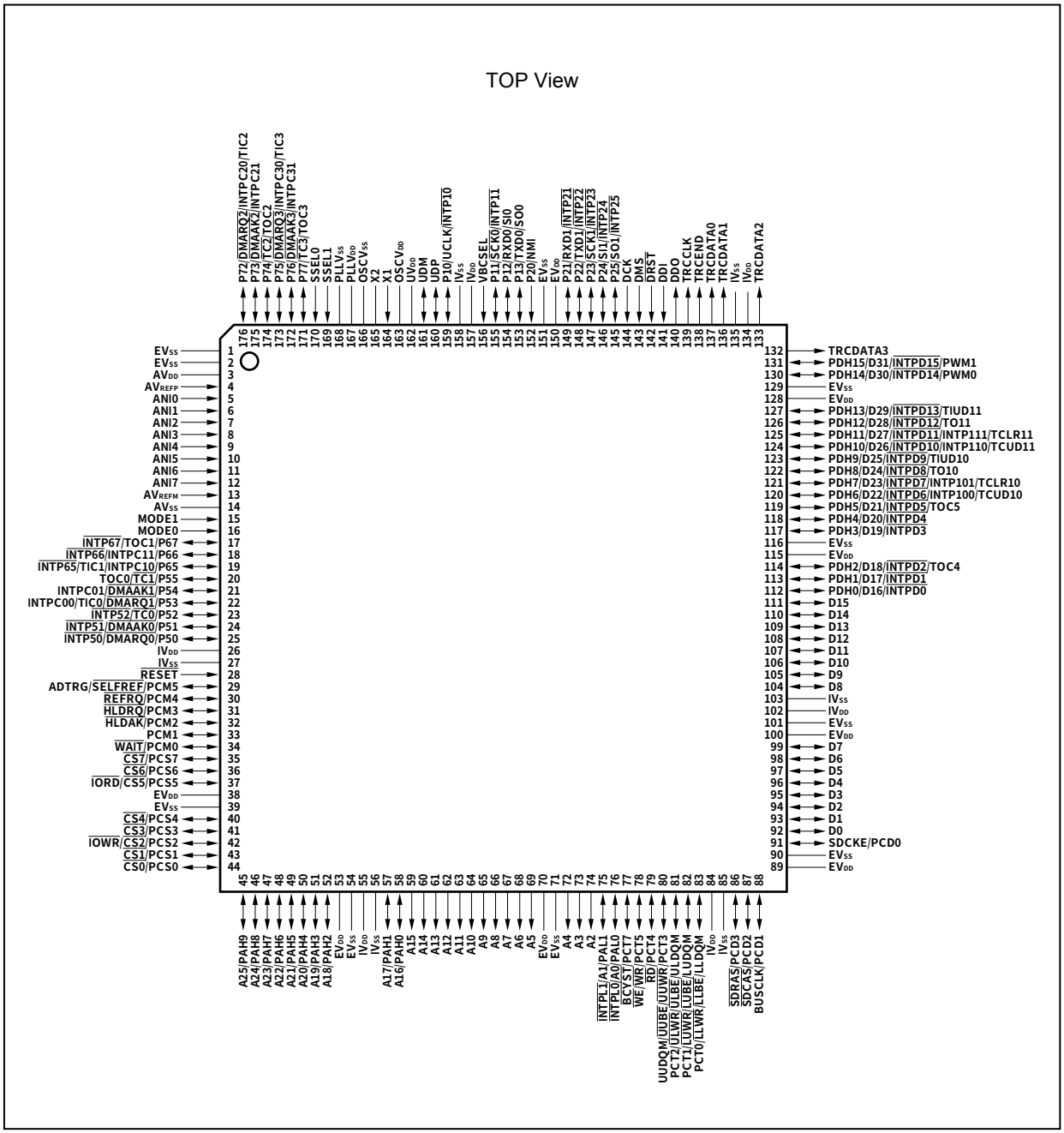
品　名	パッケージ	最高動作周波数
μPD703500GM-JEU-A	176ピン・プラスチックQFP（ファインピッチ）（24×24）	200 MHz

備考　オーダ名称末尾「-A」の製品は，鉛フリー製品です。

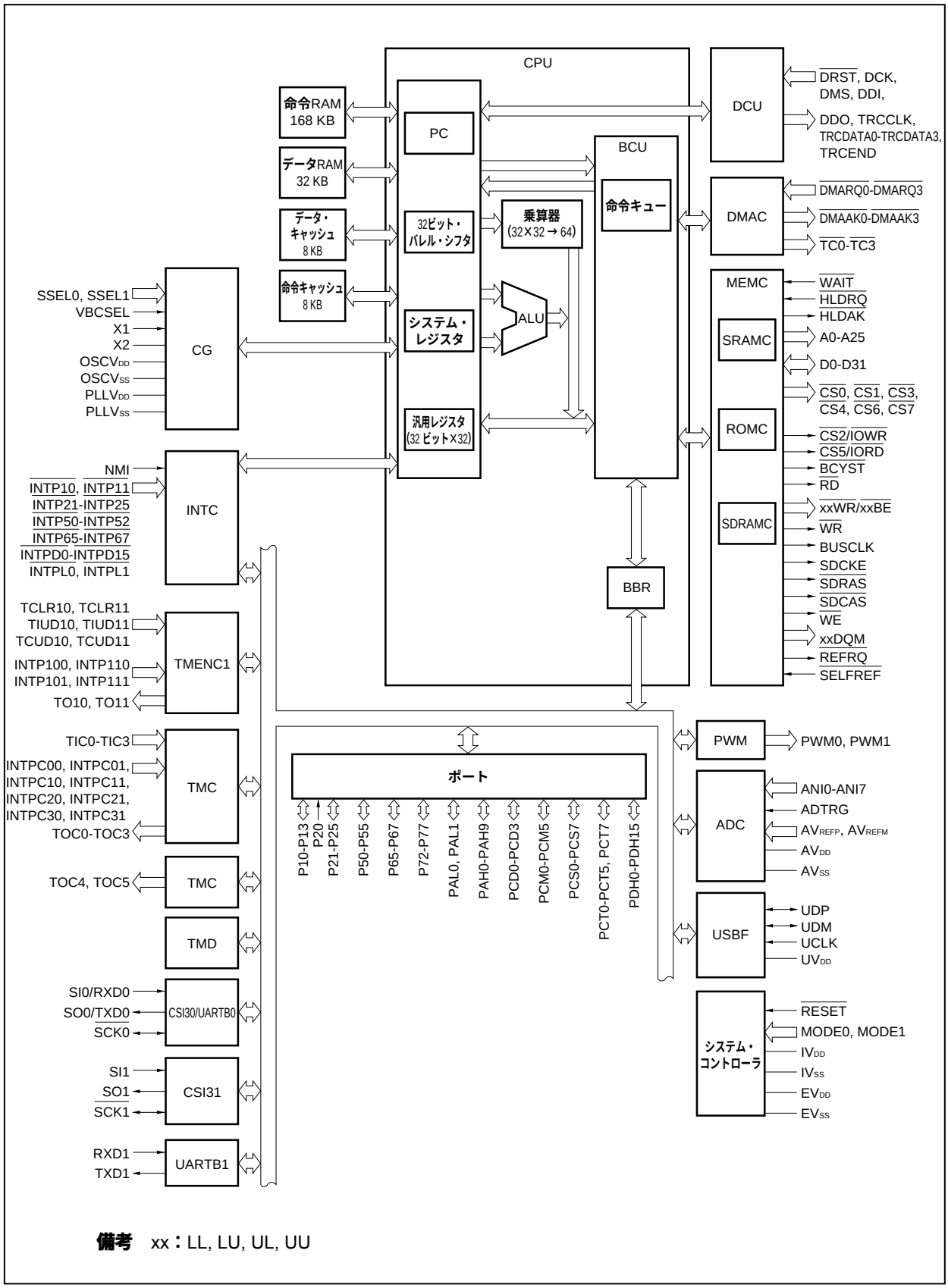
1. 3. 3　端子接続図

・176ピン・プラスチックQFP（ファインピッチ）（24×24）

μPD703500GM-JEU-A



1. 3. 4　内部ブロック図



備考　xx：LL, LU, UL, UU

1.3.5　内蔵メモリ

V850E2/ME3は，168 Kバイト（168 Kバイト×1バンク）の命令RAM，8 Kバイト（4ウエイ・セット・アソシアティブ）の命令キャッシュ，8 Kバイト（4ウエイ・セット・アソシアティブ）のデータ・キャッシュ，32 Kバイトのデータ RAMを内蔵しています。

（1）内蔵命令RAM

内蔵命令RAMは，00000000H-00029FFFHに実装されています。

V850E2/ME3は，リセット解除後，外部メモリ上の00000000H番地から命令の実行を開始します。この外部メモリ上のプログラムにより，内蔵命令RAMにプログラムを転送し，命令RAMコントロール・レジスタ（IRC）をセット（1）することで，内蔵命令RAMが有効になります。

内蔵命令RAMへのプログラム転送には，内蔵のDMA機能を使用します。詳細については，V850E2/ME3 ユーザーズ・マニュアル　ハードウエア編を参照してください。また，内蔵命令RAMには，リード・アクセス／ライト・アクセス可能モードとリード・アクセスのみ可能なモード（ライト・アクセス不可）があり，命令RAMモード・レジスタ（IRWE）で設定します。

（2）命令キャッシュ，データ・キャッシュ機能

命令キャッシュ，データ・キャッシュともに，すべての$\overline{CS}$空間がキャッシャブル領域です。キャッシュ領域指定レジスタ（BHC）により，各$\overline{CS}$空間をキャッシュ可能領域にするか，キャッシュ不可領域にするかを設定します。また，キャッシュ操作指定レジスタ（COPR）により，各ウエイのキャッシュへのオペレーションであるSync if dirty, Way Clear, Fill, Clearを設定できます。

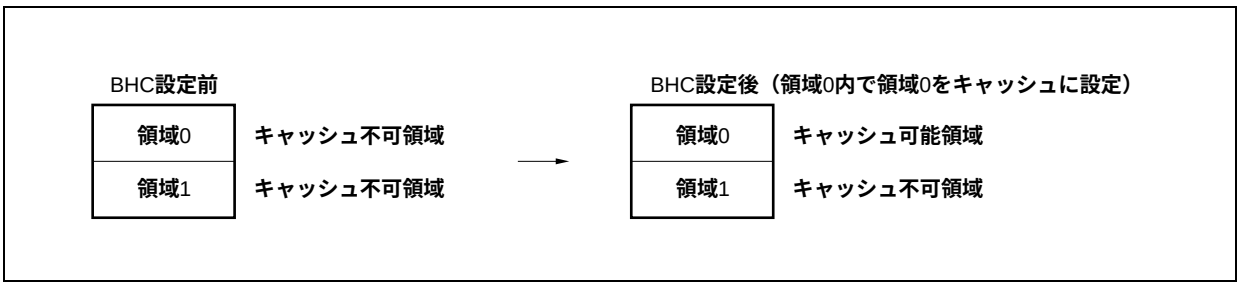
注意1. 内蔵命令RAM領域，内蔵データRAM領域および内蔵周辺I/O領域は，BHCレジスタの設定にかかわらず，「キャッシュ不可」となります。

2. 命令キャッシュ／データ・キャッシュを使用する場合は，バス・トランザクション制御レジスタ（BTSC）を設定してから，BHCレジスタを設定してください。

3. BHCレジスタの設定を行う命令自身が存在する領域を，キャッシュ不可領域→キャッシュ可能領域，またはキャッシュ可能領域→キャッシュ不可領域にすることはできません（変更した場合の動作は保証しません）。この場合，領域1に分岐してから領域1内の命令で領域0をキャッシュ可能領域に設定し，必要に応じて，再度，領域0に分岐してください。
   なお，内蔵命令RAM領域上であれば，いずれの$\overline{CS}$空間に対してもキャッシュ不可，キャッシュ可能の設定ができます。

（設定禁止例）

・領域0にBHCレジスタを設定する命令が存在する場合



注意4. BHC**レジスタは，既存製品である**V850E/ME2**のキャッシュ・コンフィギュレーション・レジスタ（**BHC**）と配置されているアドレスが同一ですが，レジスタの名称，各ビットの名称，およびビット配列が異なります。**

（3**）内蔵データ**RAM

内蔵データRAM**領域は，**1FFF7000H-1FFFEFFFH**番地の**32 K**バイトの領域に実装されています。命令コードは，配列できません。**

1. 3. 6　初期設定端子

V850E2/ME3は，さまざまな動作モードを決定する初期設定端子があります。

（1）MODE0, MODE1端子

データ・バスの動作モードを指定する入力端子です。MODE0, MODE1端子の指定は応用システムにおいて固定とし，動作中に変更した場合の動作は保証しません。

表1－5　データ・バスの設定

MODE1	MODE0	動作モード	
L	L	通常動作モード	32ビット・データ・バス
L	H		16ビット・データ・バス
上記以外		設定禁止	

備考　L：ロウ・レベル入力
　　H：ハイ・レベル入力

（2）SSEL0, SSEL1端子

X1, X2端子への入力周波数（Fx）に応じて設定する入力端子です。Fx×20 = fx（メイン・クロック）の値に応じて，SSEL0, SSEL1端子を設定してください。

表1－6　周波数一覧

逓倍数	SSEL1端子	SSEL0端子	入力周波数（MHz）（目標値）	メイン・クロック（fx）周波数（MHz）
20	0	0	6.00～7.24	120.00～144.80
	0	1	7.25～8.49	145.00～169.80
	1	0	8.50～10.00	170.00～200.00
	1	1	5.00～5.99	100.00～119.80

備考　L：ロウ・レベル入力
　　H：ハイ・レベル入力

（3）VBCSEL端子

内部システム・クロック（f_{CLK}）に対するVBCLKの分周比を指定する入力端子です。

表1－7　VBCSEL端子による分周比の設定

VBCSEL端子	分周比（f_{VBCLK}）	f_{VBCLK}出力周波数（MHz）（f_{CLK} = 200 MHz動作時）
L	f_{CLK}/2	100 MHz
H	f_{CLK}/3	66 MHz

注意　VBCSEL端子の指定は応用システムにおいて固定とし，動作中に変更した場合の動作は保証しません。

備考　L：ロウ・レベル入力
　　H：ハイ・レベル入力

★ 第2章　バス・インタフェース接続回路例

V850E/ME2, V850E2/ME3のバス・インタフェースに接続するメモリやI/Oの回路例と関連レジスタの設定について説明します。

RESETハンドラ・アドレスは，V850E/ME2ではブロック0の00100000H番地に，V850E2/ME3では00000000H番地に割り付けられており，ディフォールト時の状態は$\overline{\text{CS0}}$空間となります。このため，V850E/ME2, V850E2/ME3は，$\overline{\text{CS0}}$空間に対する命令フェッチから開始されます。したがって，$\overline{\text{CS0}}$空間にはROM（PROM／マスクROM／ページROM／フラッシュ・メモリ）を配置する必要があります。

また，V850E/ME2では256 Mバイト・メモリ空間のうち，00000000H-03FFFFFFH番地の64 Mバイト空間はプログラム空間であり，04000000H-0FFFFFFFH番地の192 Mバイト空間（$\overline{\text{CS3}}$-$\overline{\text{CS7}}$が使用可能）はプログラム・メモリを配列できないのでデータ用メモリやI/Oデバイスを配列します。V850E2/ME3では512 Mバイト・メモリ空間のすべての領域をプログラム空間として使用できます。ディフォールトのデータ・バス幅は，MODE0, MODE1端子の設定で決定されます。

回路例と関連レジスタの設定は次に示す考え方で構成しています。

(1) V850E/ME2, V850E2/ME3のバス・クロックは64 MHz ，SSCG出力の周波数変調率は周波数固定（変調なし）で使用することとします。

<V850E/ME2の場合>

・バス・モード・コントロール・レジスタ（BMC）の設定：01H（内部システム・クロック（f_{CLK}）を128 MHzと仮定してf_{CLK}/2を設定）

・SSCGコントロール・レジスタ（SSCGC）のADJONビットの設定：0

<V850E2/ME3の場合>

・VBCSEL端子の設定：ハイ・レベル

・バス・モード・コントロール・レジスタ（BMC）の設定：00H（内部システム・クロック（f_{CLK}）を192 MHzと仮定してf_{CLK}/3を設定）

・SSCGコントロール・レジスタ（SSCGC）のADJONビットの設定：0

(2) 接続回路例はV850E/ME2, V850E2/ME3に直結可能な3 V系デバイスを接続しています。

備考　2. 9　5 Vデバイスとの接続では，5 V系デバイスとの接続を示しています。

(3) バス・サイクル・ピリオド・コントロール・レジスタ（BCP）の設定は00H, エンディアン・コンフィギュレーション・レジスタ（BEC）の設定は任意とします（BECレジスタはV850E/ME2のみ）。

・$\overline{\text{IORD}}$, $\overline{\text{IOWR}}$の動作：禁止

・エンディアンの設定：必要に応じて設定

備考　2. 10　DMAフライバイ転送時の接続では，$\overline{\text{IORD}}$, $\overline{\text{IOWR}}$の使用例を示しています。

(4) ライン・バッファ・コントロール・レジスタ0, 1（LBC0, LBC1）の設定は任意とします（LBC0, LBC1レジスタはV850E/ME2のみ）。

・先読み機能の設定：必要に応じて設定

備考　1. 2. 6　先読み機能（リード・バッファ機能）を参照してください。なお，V850E2/ME3には先読み機能はありません。

(5) PCTn/$\overline{\text{xxWR}}$/$\overline{\text{xxBE}}$/xxDQM端子は$\overline{\text{xxBE}}$/xxDQM端子として使用することとします。

・ポートCTファンクション・コントロール・レジスタ（PFCCT）の設定：0FH

備考　n = 0-3

xx：LL, LU, UL, UU

(6) 外部付加回路を経由して外部メモリや外部I/Oと接続する場合の付加回路による信号伝達遅延時間を2 ns（MIN.），7 ns（MAX.）として計算しています。また，プリント基板配線上の遅延時間は含まれていないので，実際の回路では必要に応じて数nsのマージンを考慮してください。

2. 1　8ビットSRAMとの接続

SRAM（μ PD444008L：512 K×8ビット）を接続する例（V850E/ME2：ブロック1，V850E2/ME3：サブエリア01に配列し，$\overline{CS}$信号は$\overline{CS2}$を使用する場合）を示します。

備考　$\overline{IORD}$, $\overline{IOWR}$を使用する場合，$\overline{CS2}$は使用できません。また，各種構成例は，2. 11　システム構成例とCSCnレジスタの設定を参照してください。

2. 1. 1　μ PD444008Lとの接続（8ビット・バス幅例）

SRAM（μ PD444008L-A8：512 K×8ビット）を1つ（8ビット・バス幅，512 Kバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- 接続デバイス：μ PD444008L-A8×1つ
- 使用$\overline{CS}$信号：$\overline{CS2}$

  外部メモリ空間の0200000H-027FFFFH（V850E/ME2：ブロック1，V850E2/ME3：サブエリア01）に配列することとします。

  0280000H-03FFFFFHはイメージとなります。

【接続の考え方と注意点】

8ビット・バス幅で構成するため，μ PD444008L-A8のデータ・バス（I/O0-I/O7）はV850E/ME2, V850E2/ME3のD0-D7に，μ PD444008L-A8のアドレス・バス（A0-A18）はV850E/ME2, V850E2/ME3のA0-A18に接続します。また，μ PD444008L-A8の$\overline{CS}$, $\overline{OE}$, $\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{CS2}$, $\overline{RD}$, $\overline{WR}$端子にそれぞれ接続します。

【レジスタの設定 】

- $\overline{CS2}$の使用エリアとデバイス・タイプ，バス幅：

  V850E/ME2：ブロック1，V850E2/ME3：サブエリア01，SRAM／外部I/O，8ビット幅
- ウエイト設定：0ウエイト
- アドレス・セットアップ・ウエイト：0ウエイト
- アイドル・ステート：1ステート

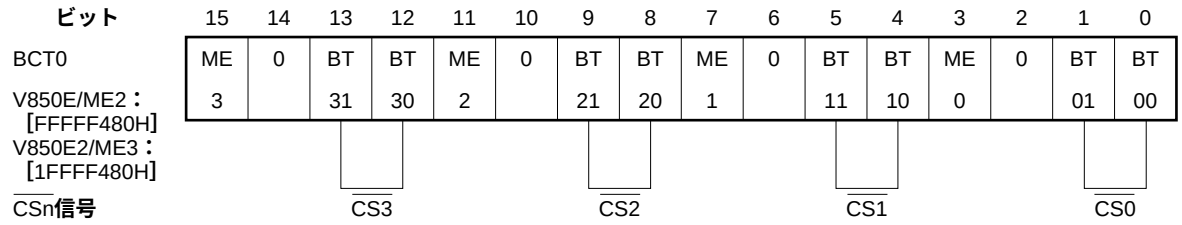
図2－1　チップ・エリア選択コントロール・レジスタ0（CSC0）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2：［FFFFF060H］ V850E2/ME3：［1FFFF060H］	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00

V850E/ME2：ブロック1，V850E2/ME3：サブエリア01アクセス時$\overline{CS2}$出力

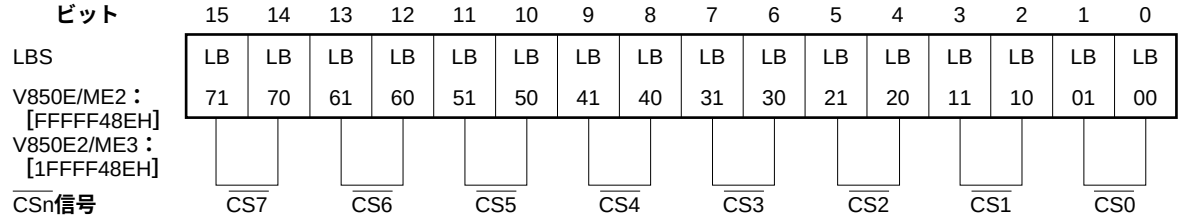
CSC0の設定値：xxxx0010xxxxxx0xB

図2－2　バス・サイクル・タイプ・コンフィギュレーション・レジスタ0（BCT0）の設定



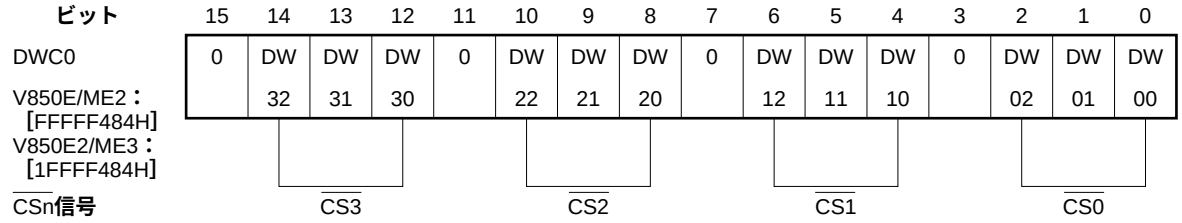
CS2空間をメモリ・コントローラ動作許可，SRAM／外部I/Oに指定
BCT0の設定値：x0xx1000x0xxx0xxB

図2－3　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定



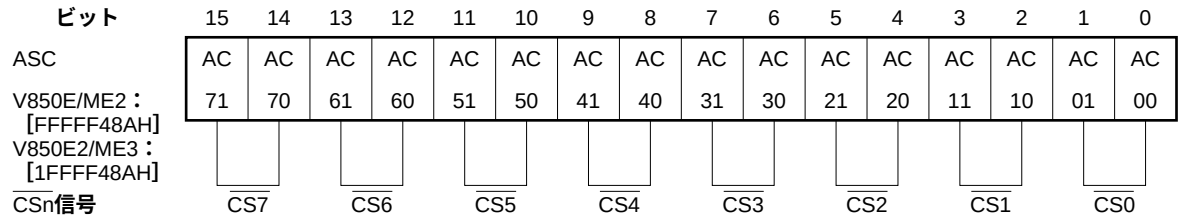
CS2空間をデータ・バス幅8ビットに指定
LBSの設定値：xxxxxxxxxx00xxxxB

図2－4　データ・ウエイト・コントロール・レジスタ0（DWC0）の設定



CS2空間のウエイト：0
DWC0の設定値：0xxx00000xxx0xxxB

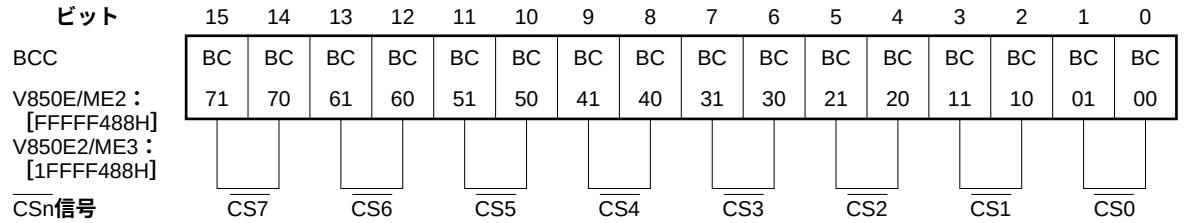
図2－5　アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）の設定



CS2空間のアドレス・セットアップ・ウエイト：0

ASCの設定値：xxxxxxxxxx00xxxxB

図2－6　バス・サイクル・コントロール・レジスタ（BCC）の設定



CS2空間のアイドル・ステート：1

BCCの設定値：xxxxxxxxxx01xxxxB

図2－7　μPD444008L-A8接続回路例

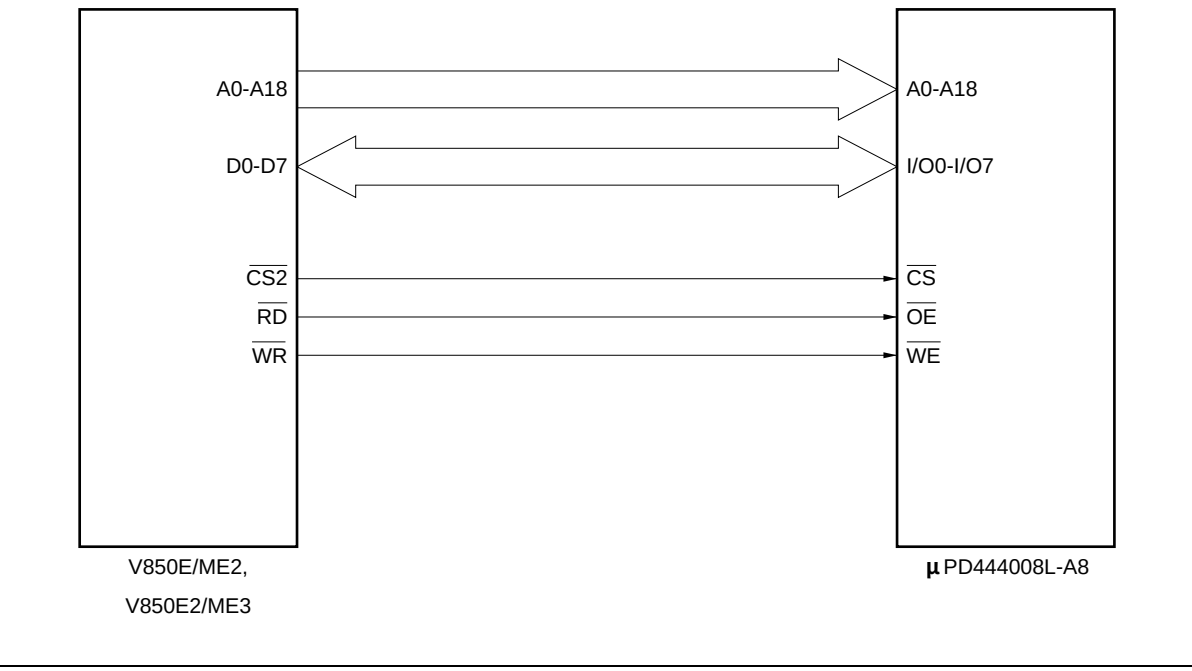
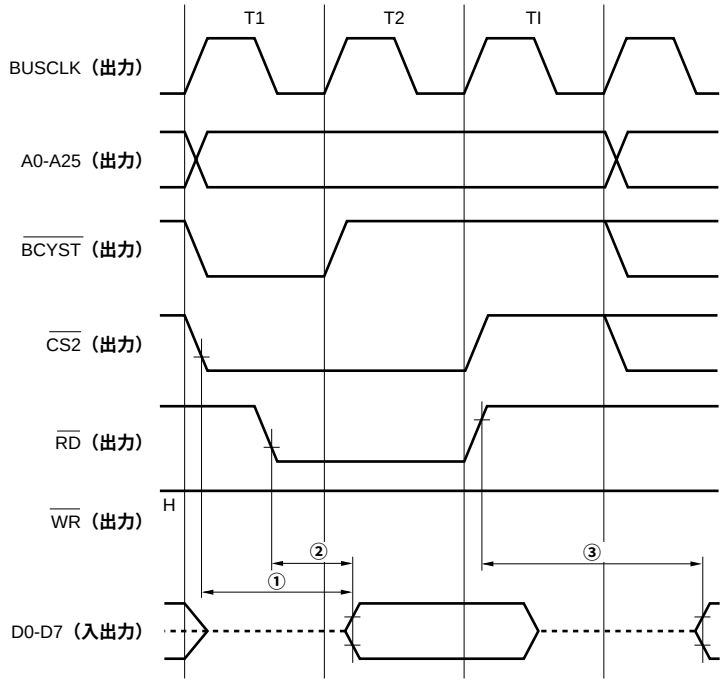


図2－8　μPD444008L-A8のリード動作



① μPD444008L-A8のアドレス，CSアクティブからの出力遅延時間：8 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ入力設定時間（対アドレス）の最大値

t_{SAID}（ns）　=（2＋w＋w_D＋w_{AS}）T－17
= 2×15.6－17；w = 0, w_D = 0, w_{AS} = 0, T = 15.6 ns
= 14.2 ns（＞8 ns）

② μPD444008L-A8のOEアクティブからの出力遅延時間：4 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ入力設定時間（対RD）の最大値

t_{SRDID}（ns）　=（1.5＋w＋w_D）T－17
= 1.5×15.6－17；w = 0, w_D = 0, T = 15.6 ns
= 6.4 ns（＞4 ns）

③ μPD444008L-A8のOEインアクティブからの出力フローティング遅延時間：4 ns（MIN.）
V850E/ME2, V850E2/ME3の電気的特性より，データ出力遅延時間（対RD↑）の最小値

t_{DRDOD}（ns）=（0.5＋i）T－6
= 1.5×15.6－6；i = 1, T = 15.6 ns
= 17.4 ns（＞4 ns）

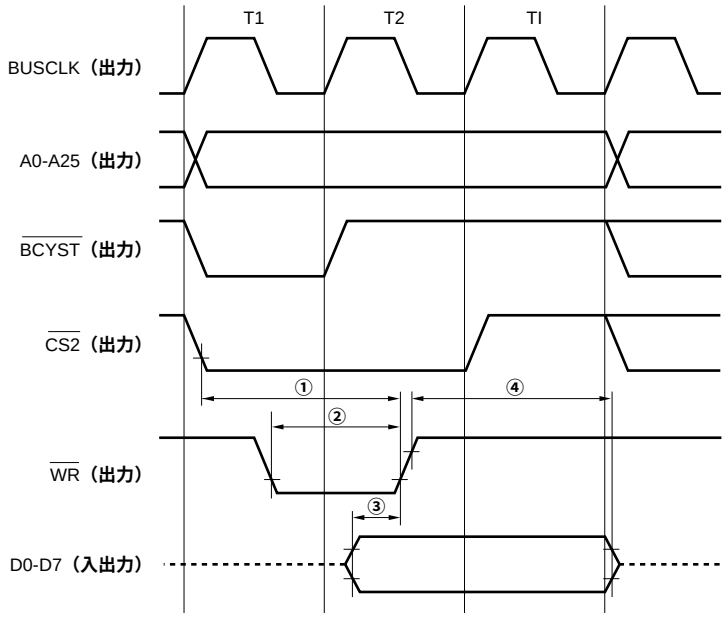
またRDハイ・レベル幅の最小値

t_{WRDH}（ns）　=（0.5＋w_{AS}＋i）T－6
= 1.5×15.6－6；w_{AS} = 0, i = 1, T = 15.6 ns
= 17.4 ns（＞4 ns）

備考1.　破線はハイ・インピーダンスを示しています。

2.　T　：t_{CYK}（BUSCLK出力周期）
w　：WAITによるウエイト数
w_D　：DWC0, DWC1レジスタによるウエイト数
w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数
i　：アイドル・ステート数

図2－9　μPD444008L-A8のライト動作



① μPD444008L-A8のアドレス，CSアクティブからWR立ち上がりまでの時間：6 ns（MIN.）
V850E/ME2, V850E2/ME3の電気的特性より，アドレス設定時間（対WR↑）の最小値
t_{SAWR}（ns）＝（1.5＋w＋w_D＋w_{AS}）T－10
＝ 1.5×15.6－10；w = 0, w_D = 0, w_{AS} = 0, T = 15.6 ns
＝ 13.4 ns（＞6 ns）

② μPD444008L-A8のWEアクティブ・パルス幅：6 ns（MIN.）
V850E/ME2, V850E2/ME3の電気的特性より，WRロウ・レベル幅の最小値
t_{WWRL}（ns）＝（1＋w＋w_D）T－5
＝ 1×15.6－5；w = 0, w_D = 0, T = 15.6 ns
＝ 10.6 ns（＞6 ns）

③ μPD444008L-A8のWR立ち上がりに対するデータ設定時間：4 ns（MIN.）
V850E/ME2, V850E2/ME3の電気的特性より，データ出力設定時間（対WR↑）の最小値
t_{SODWR}（ns）＝（1.5＋w_{AS}＋w＋w_D）T－5
＝ 1.5×15.6－5；w_{AS} = 0, w = 0, w_D = 0, T = 15.6 ns
＝ 18.4 ns（＞4 ns）

④ μPD444008L-A8のWR立ち上がりからのデータ保持時間：0 ns（MIN.）
V850E/ME2, V850E2/ME3の電気的特性より，データ出力保持時間（対WR↑）の最小値
t_{HWROD}（ns）＝（0.5＋i）T－5.5
＝ 1.5×15.6－5.5；i = 1, T = 15.6 ns
＝ 17.9 ns

備考1. 破線はハイ・インピーダンスを示しています。
2. T　：t_{CYK}（BUSCLK出力周期）
w　：WAITによるウエイト数
w_D　：DWC0, DWC1レジスタによるウエイト数
w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数
i　：アイドル・ステート数

2. 1. 2　μPD444008Lとの接続（16ビット・バス幅例）

SRAM（μPD444008L-A8：512 K×8ビット）を2つ（16ビット・バス幅，1 Mバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- 接続デバイス：μPD444008L-A8×2つ
- 使用$\overline{CS}$信号：$\overline{CS2}$

  外部メモリ空間の0200000H-02FFFFFH（V850E/ME2：ブロック1，V850E2/ME3：サブエリア01）に配列することとします。

  0300000H-03FFFFFHはイメージとなります。

【接続の考え方と注意点】

16ビット・バス幅で構成するため，μPD444008L-A8のアドレス・バス（A0-A18）はV850E/ME2, V850E2/ME3のA1-A19に接続します。V850E/ME2, V850E2/ME3のD0-D7に接続したμPD444008L-A8の$\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{WR}$信号と$\overline{LLBE}$信号で，V850E/ME2, V850E2/ME3のD8-D15に接続したμPD444008L-A8の$\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{WR}$信号と$\overline{LUBE}$信号で制御します。μPD444008L-A8の$\overline{CS}$, $\overline{OE}$端子はV850E/ME2, V850E2/ME3の$\overline{CS2}$, $\overline{RD}$端子にそれぞれ接続します。

【レジスタの設定】

LBSレジスタ（$\overline{CS}$空間のデータ・バス幅指定）の設定を除いて2. 1. 1　μPD444008Lとの接続（8ビット・バス幅例）と同等です。

図2－10　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2：[FFFFF48EH] V850E2/ME3：[1FFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
$\overline{CS}$n信号	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

$\overline{CS2}$空間をデータ・バス幅16ビットに指定

LBSの設定値：xxxxxxxxxx01xxxxB

図2－11　μPD444008L-A8接続回路例

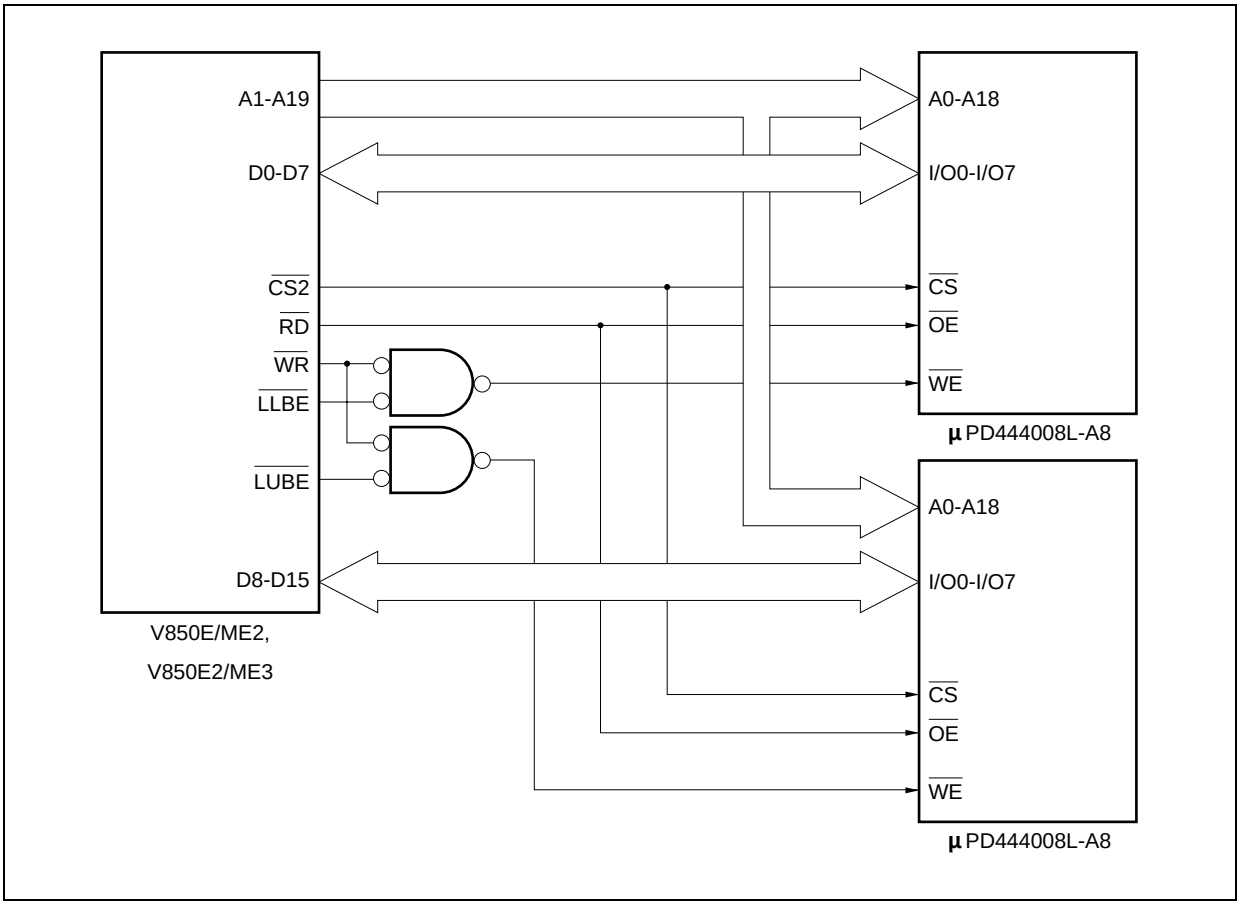
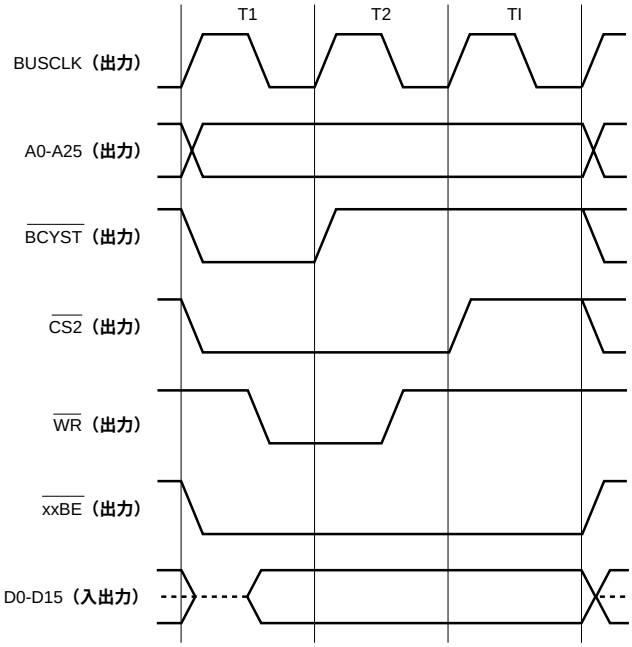


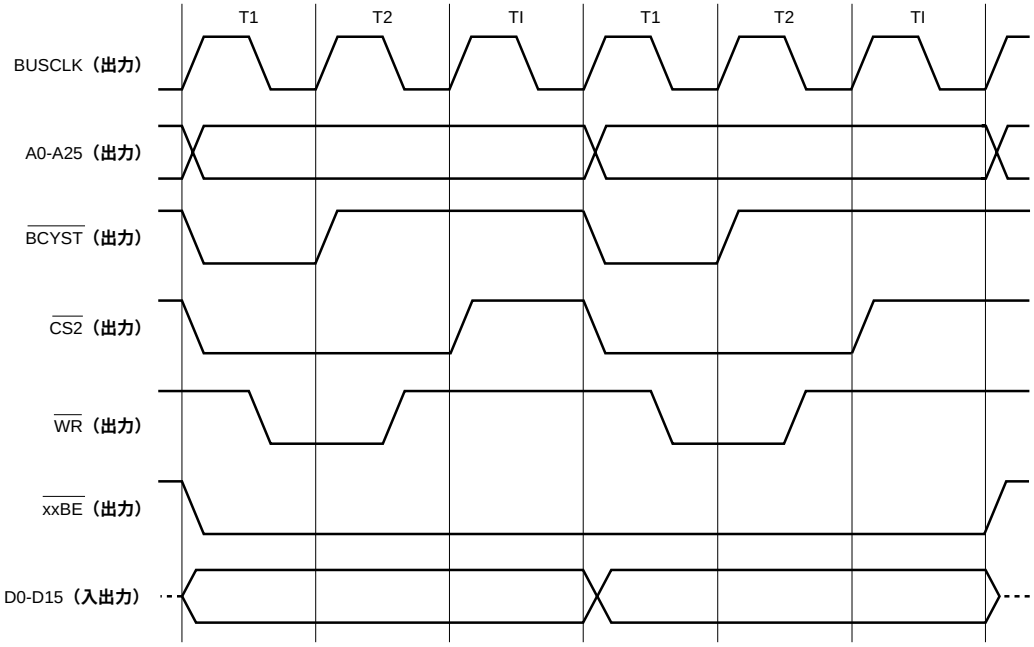
図2ー12　μPD444008L-A8のライト動作（W_D = 0, W_{AS} = 0, i = 1，16ビット・バス幅）



備考1.　**破線はハイ・インピーダンスを示します。**

2.　xx = LL, LU

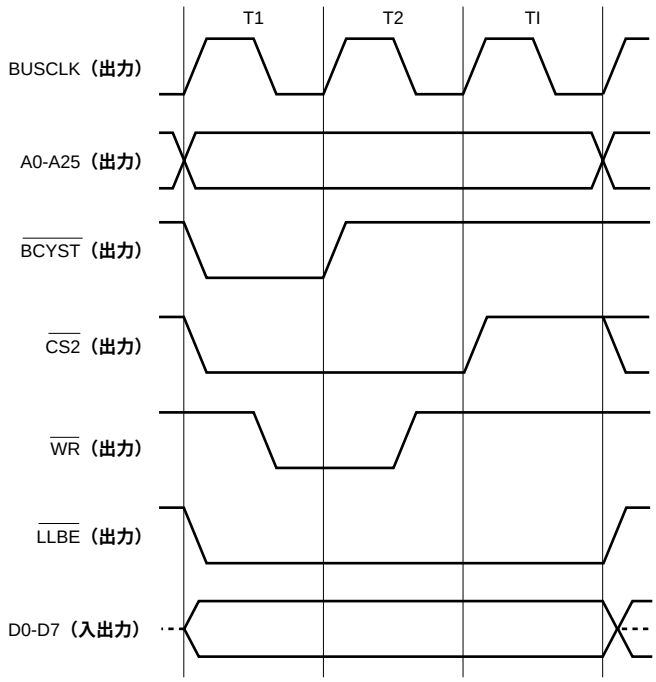
図2ー13　μPD444008L-A8のライト動作（W_D = 0, W_{AS} = 0, i = 1，16ビット・バス幅，ワード・アクセス）



備考1.　**破線はハイ・インピーダンスを示します。**

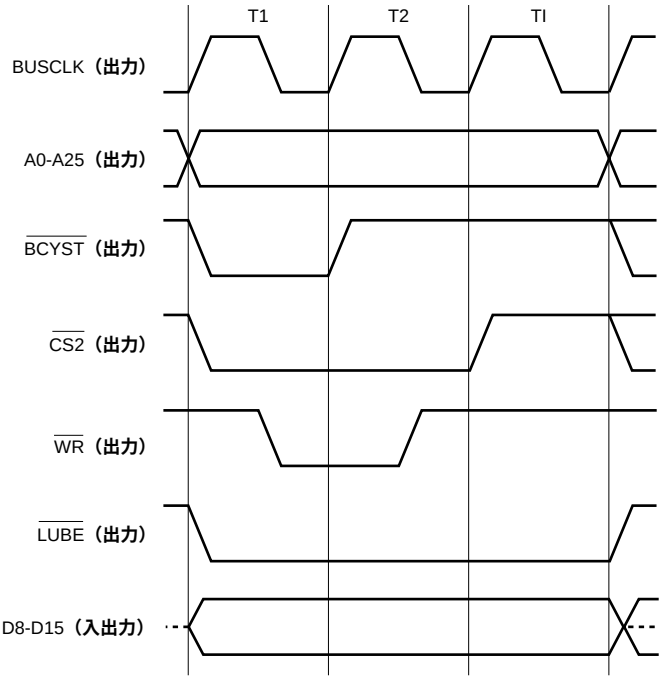
2.　xx = LL, LU

図2－14　μPD444008L-A8のライト動作（D0-D7に対するバイト・アクセス）



備考　破線は，ハイ・インピーダンスを示します。

図2－15　μPD444008L-A8のライト動作（D8-D15に対するバイト・アクセス）



備考　破線は，ハイ・インピーダンスを示します。

2. 1. 3　μPD444008Lとの接続（32ビット・バス幅例）

SRAM（μPD444008L-A8：512 K×8ビット）を4つ（32ビット・バス幅，2 Mバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- 接続デバイス：μPD444008L-A8×4つ
- 使用$\overline{CS}$信号：$\overline{CS2}$

  外部メモリ空間の0200000H-03FFFFFH（V850E/ME2：ブロック1，V850E2/ME3：サブエリア01）に配列することとします。

【接続の考え方と注意点】

32ビット・バス幅で構成するため，μPD444008L-A8のアドレス・バス（A0-A18）はV850E/ME2, V850E2/ME3のA2-A20に接続します。V850E/ME2, V850E2/ME3のD0-D7に接続したμPD444008L-A8の$\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{WR}$信号と$\overline{LLBE}$信号で，V850E/ME2, V850E2/ME3のD8-D15に接続したμPD444008L-A8の$\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{WR}$信号と$\overline{LUBE}$信号で，V850E/ME2, V850E2/ME3のD16-D23に接続したμPD444008L-A8の$\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{WR}$信号と$\overline{ULBE}$信号で，V850E/ME2, V850E2/ME3のD24-D31に接続したμPD444008L-A8の$\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{WR}$信号と$\overline{UUBE}$信号で制御します。μPD444008L-A8の$\overline{CS}$, $\overline{OE}$端子はV850E/ME2, V850E2/ME3の$\overline{CS2}$, $\overline{RD}$端子にそれぞれ接続します。

【レジスタの設定】

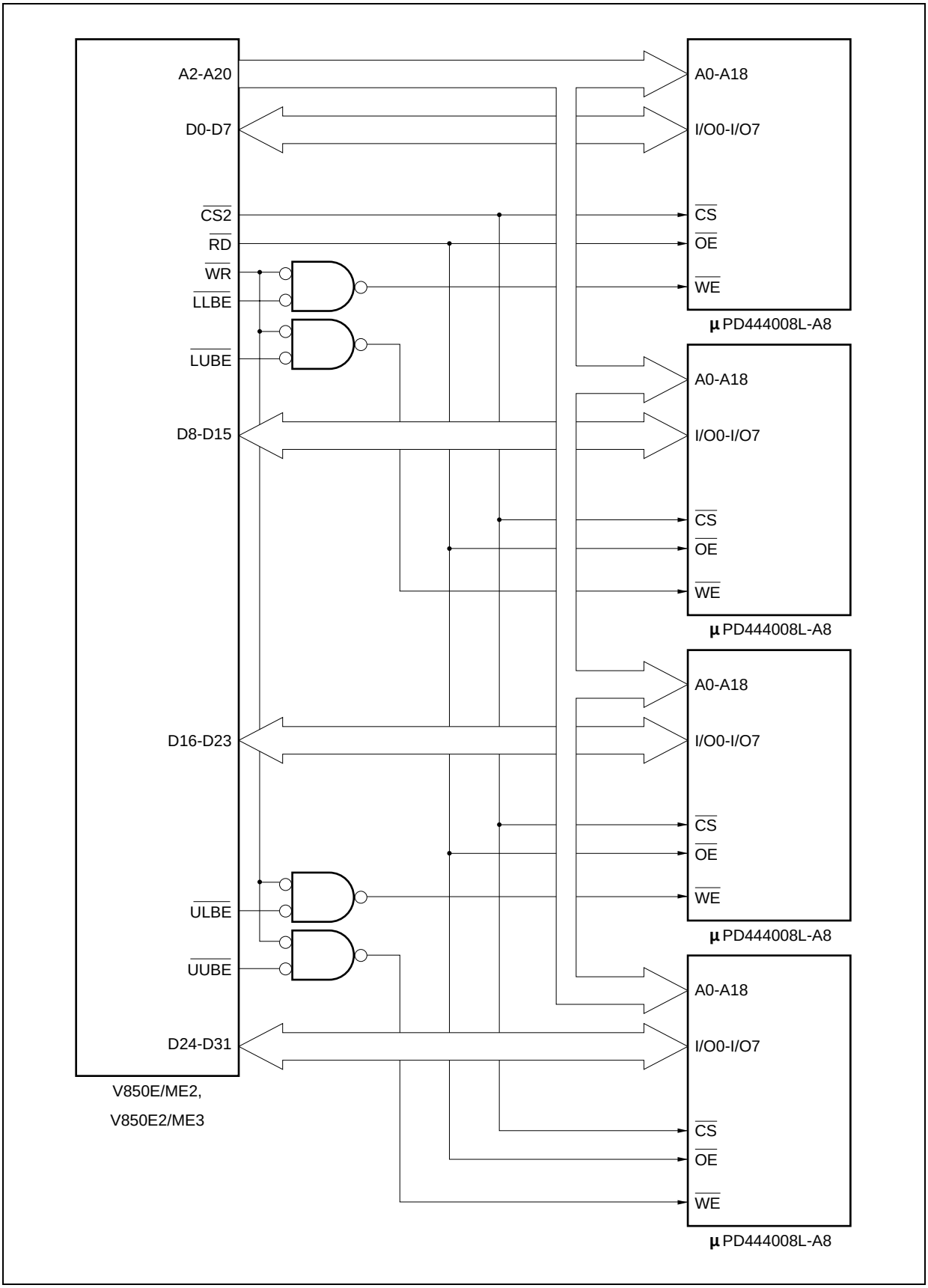
LBSレジスタ（$\overline{CS}$空間のデータ・バス幅指定）の設定を除いて2. 1. 1　μPD444008Lとの接続（8ビット・バス幅例）と同等です。

図2－16　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2：[FFFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3：[1FFFF48EH]																
$\overline{CS}$n信号		$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$

$\overline{CS2}$空間をデータ・バス幅32ビットに指定
LBSの設定値：xxxxxxxxxx10xxxxB

図2－17　μPD444008L-A8接続回路例



2. 2　16ビットSRAMとの接続

SRAM（μPD444016L：256 K×16ビット）を接続する例（V850E/ME2：ブロック4，V850E2/ME3：サブエリア04に配列し，$\overline{CS}$信号は$\overline{CS5}$を使用する場合）を示します。

備考　2. 2節では，レジスタの設定と接続回路例だけを説明します。また，動作タイミングについては，2. 1　8ビットSRAMとの接続と同等です。

2. 2. 1　μPD444016Lとの接続（16ビット・バス幅例）

SRAM（μPD444016L-A8：256 K×16ビット）を1つ（16ビット・バス幅，512 Kバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- 接続デバイス：μPD444016L-A8×1つ
- 使用$\overline{CS}$信号：$\overline{CS5}$

  V850E/ME2では外部メモリ空間のF800000H-F87FFFFH（ブロック4）に配列することとします。
  F880000H-F9FFFFFHはイメージとなります。
  V850E2/ME3では外部メモリ空間の1F800000H-1F87FFFFH（サブエリア04）に配列することとします。
  1F880000H-1F9FFFFFHはイメージとなります。

【接続の考え方と注意点】

16ビット・バス幅で構成するため，μPD444016L-A8のデータ・バス（I/O0-I/O15）はV850E/ME2, V850E2/ME3のD0-D15に，μPD444016L-A8のアドレス・バス（A0-A17）はV850E/ME2, V850E2/ME3のA1-A18に接続します。また，μPD444016L-A8の$\overline{CS}$, $\overline{OE}$, $\overline{WE}$, $\overline{LB}$, $\overline{UB}$端子はV850E/ME2, V850E2/ME3の$\overline{CS5}$, $\overline{RD}$, $\overline{WR}$, $\overline{LLBE}$, $\overline{LUBE}$端子にそれぞれ接続します。

【レジスタの設定 】

- $\overline{CS5}$の使用エリアとデバイス・タイプ，バス幅：

  V850E/ME2：ブロック4，V850E2/ME3：サブエリア04，SRAM／外部I/O，16ビット幅
- ウエイト設定：0ウエイト
- アドレス・セットアップ・ウエイト：0ウエイト
- アイドル・ステート：1ステート

図2－18　チップ・エリア選択コントロール・レジスタ1（CSC1）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC1	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2 [FFFFF062H]	43	42	41	40	53	52	51	50	63	62	61	60	73	72	71	70
V850E2/ME3 [1FFFF062H]																

V850E/ME2：ブロック4，V850E2/ME3：サブエリア04アクセス時$\overline{CS5}$出力
CSC1の設定値：xxxx1000xxxx0xxxB

図2－19　バス・サイクル・タイプ・コンフィギュレーション・レジスタ1（BCT1）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCT1	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT
V850E/ME2 [FFFFF482H]	7		71	70	6		61	60	5		51	50	4		41	40
V850E2/ME3 [1FFFF482H]																
$\overline{CS}$n信号			$\overline{CS7}$				$\overline{CS6}$				$\overline{CS5}$				$\overline{CS4}$	

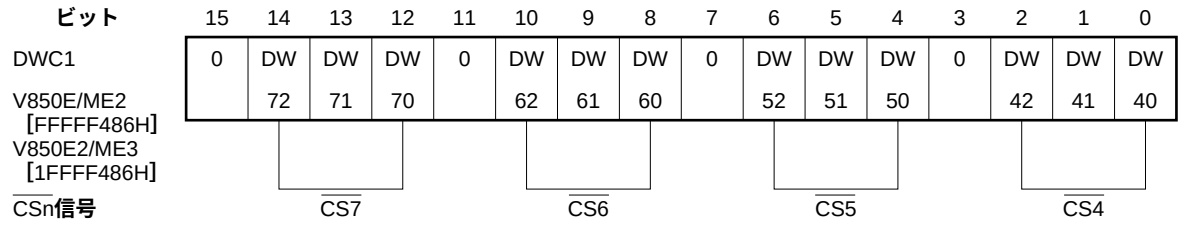
$\overline{CS5}$空間をメモリ・コントローラ動作許可，SRAM／外部I/Oに指定
BCT1の設定値：x0xxx0xx1000x0xxB

図2－20　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LBS	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB	LB
V850E/ME2： [FFFFF48EH]	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
V850E2/ME3： [1FFFF48EH]																
$\overline{CS}$n信号	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

$\overline{CS5}$空間をデータ・バス幅16ビットに指定
LBSの設定値：xxxx01xxxxxxxxxxB

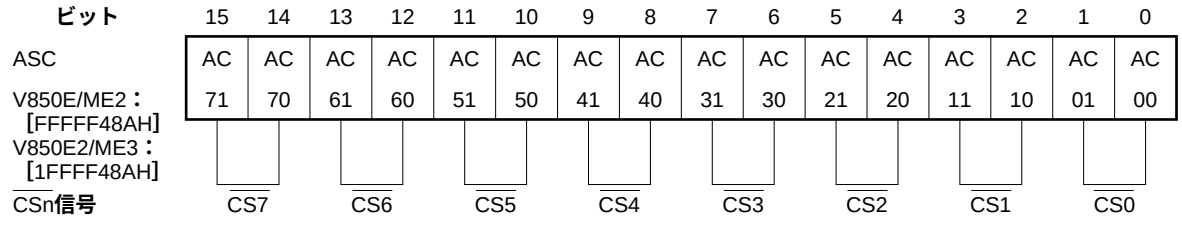
図2－21　データ・ウエイト・コントロール・レジスタ1（DWC1）の設定



$\overline{CS5}$**空間のウエイト：**0

DWC1**の設定値：**0xxx0xxx00000xxxB

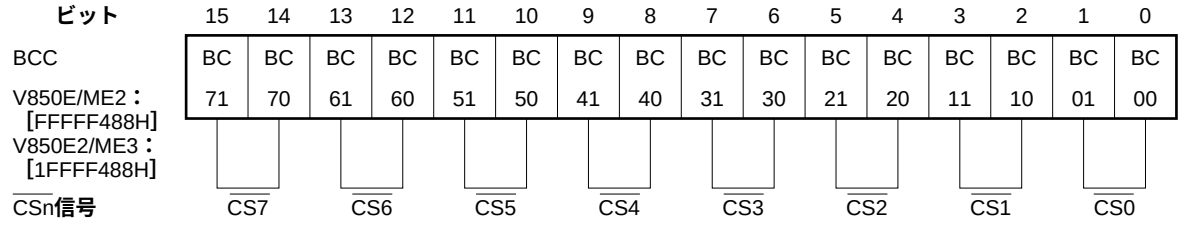
図2－22　アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）の設定



$\overline{CS5}$**空間のアドレス・セットアップ・ウエイト：**0

ASC**の設定値：**xxxx00xxxxxxxxxxB

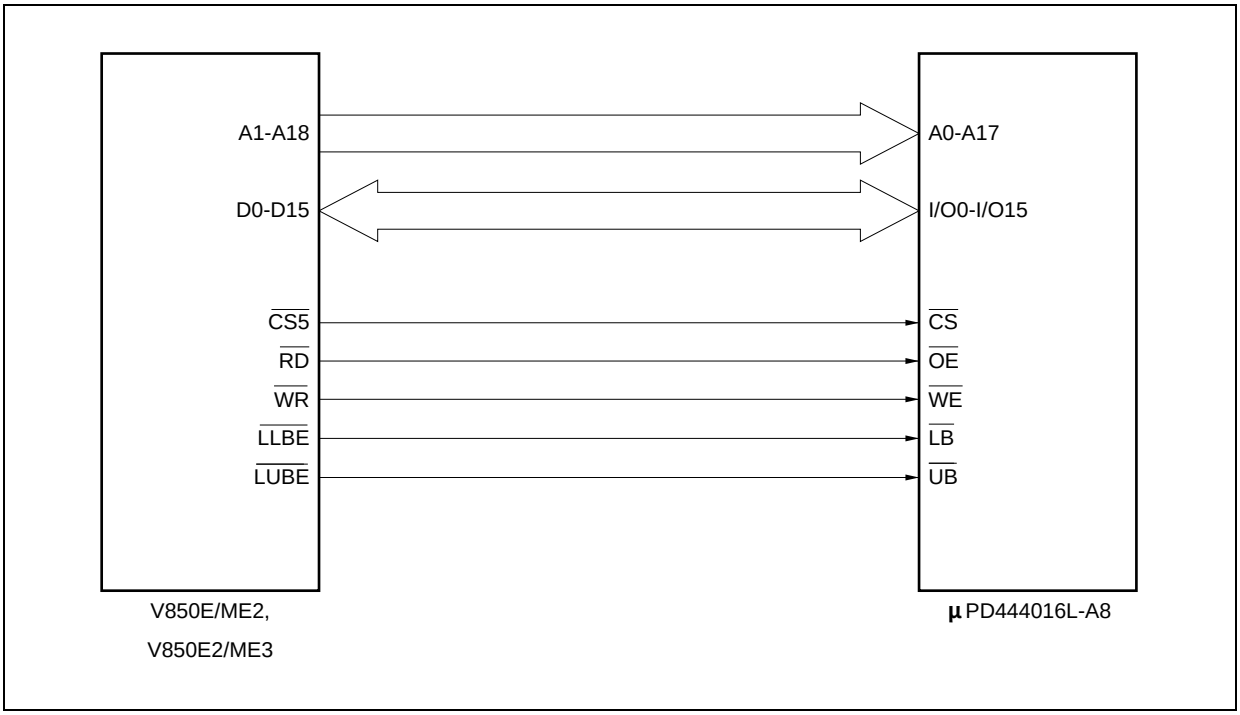
図2－23　バス・サイクル・コントロール・レジスタ（BCC）の設定



$\overline{CS5}$**空間のアイドル・ステート：**1

BCC**の設定値：**xxxx01xxxxxxxxxxB

図2－24　μPD444016L-A8接続回路例



2. 2. 2　μPD444016Lとの接続（32ビット・バス幅例）

SRAM（μPD444016L-A8：256 K×16ビット）を2つ（32ビット・バス幅，1 Mバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- 接続デバイス：μPD444016L-A8×2つ
- 使用$\overline{CS}$信号：$\overline{CS5}$

  V850E/ME2では外部メモリ空間のF800000H-F8FFFFFH（ブロック4）に配列することとします。

  F900000H- F9FFFFFHはイメージとなります。

  V850E2/ME3では外部メモリ空間の1F800000H-1F8FFFFFH（サブエリア04）に配列することとします。

  1F900000H-1F9FFFFFHはイメージとなります。

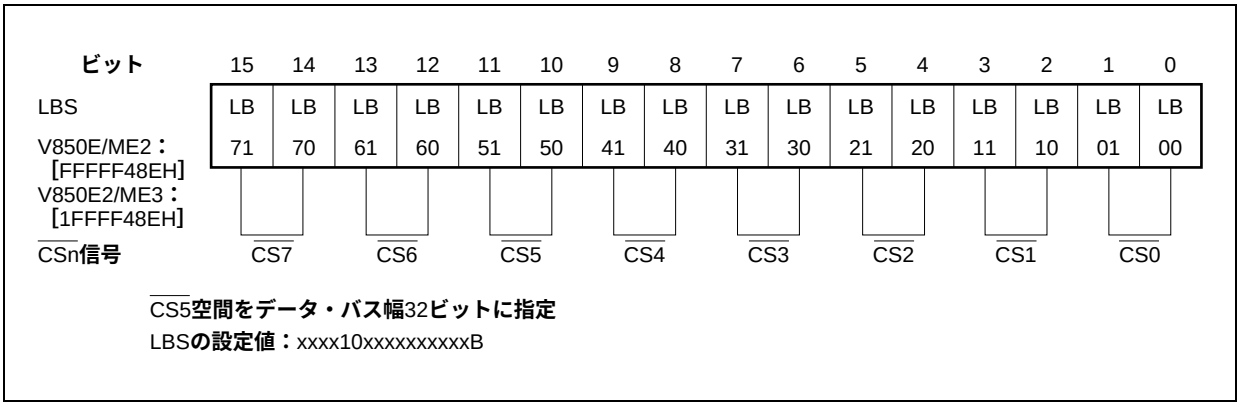
【接続の考え方と注意点】

32ビット・バス幅で構成するため，μPD444016L-A8のアドレス・バス（A0-A17）はV850E/ME2, V850E2/ME3のA2-A19に接続します。V850E/ME2, V850E2/ME3のD0-D15に接続したμPD444016L-A8の$\overline{LB}$, $\overline{UB}$端子はV850E/ME2, V850E2/ME3の$\overline{LLBE}$, $\overline{LUBE}$端子に，V850E/ME2, V850E2/ME3のD16-D31に接続したμPD444016L-A8の$\overline{LB}$, $\overline{UB}$端子はV850E/ME2, V850E2/ME3の$\overline{ULBE}$, $\overline{UUBE}$端子にそれぞれ接続します。また，μPD444016L-A8の$\overline{CS}$, $\overline{OE}$, $\overline{WE}$端子はV850E/ME2, V850E2/ME3の$\overline{CS5}$, $\overline{RD}$, $\overline{WR}$端子にそれぞれ接続します。

【レジスタの設定 】

LBSレジスタ（$\overline{CS}$空間のデータ・バス幅指定）の設定を除いて2. 2. 1　μPD444016Lとの接続（16ビット・バス幅例）と同等です。

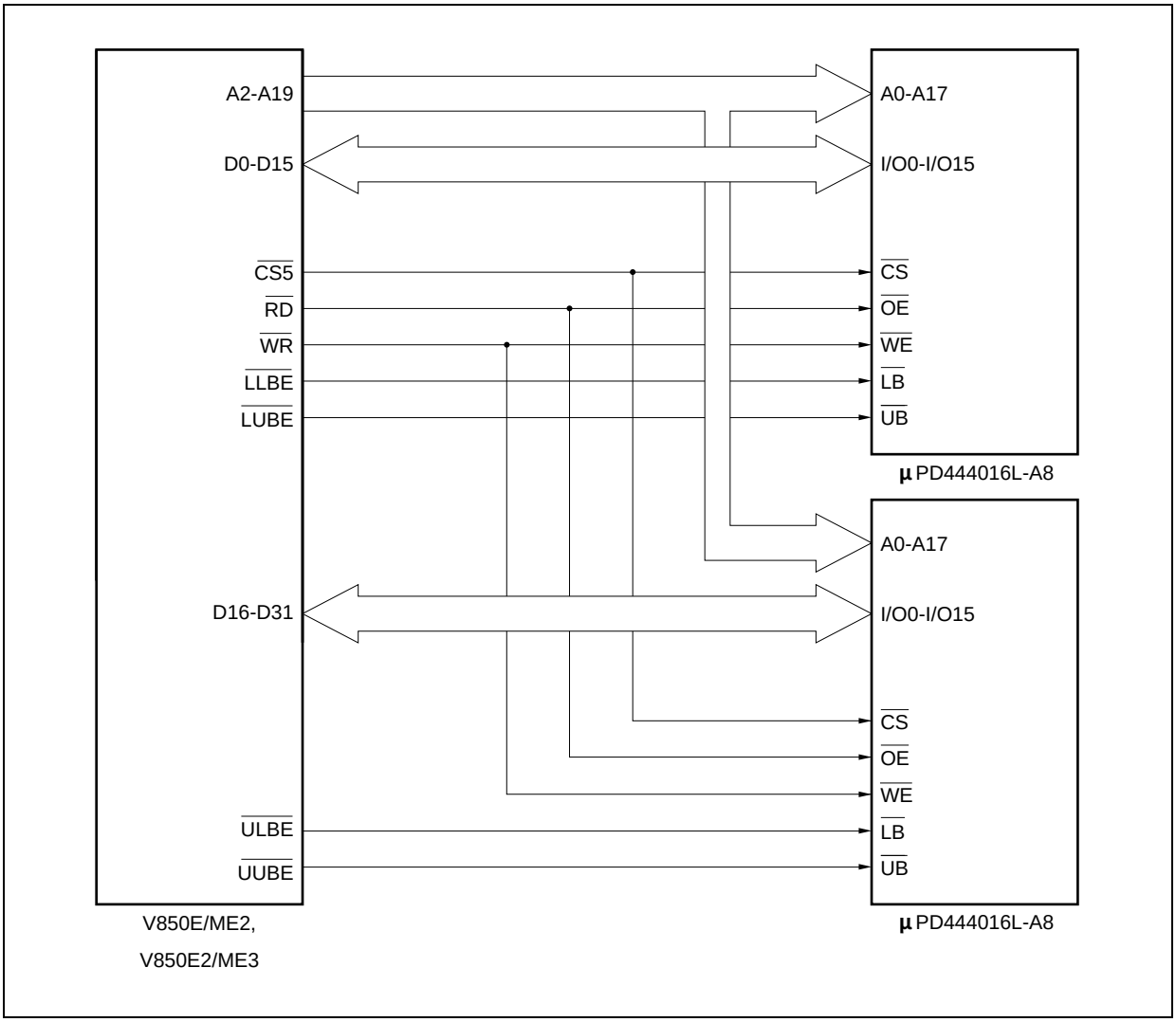
図2－25　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定



$\overline{CS5}$空間をデータ・バス幅32ビットに指定

LBSの設定値：xxxx10xxxxxxxxxxB

図2－26　μPD444016L-A8接続回路例



2. 2. 3　SRAM接続時のウエイト／アイドル設定とバス・クロック一覧

表2－1　SRAM接続時のウエイト／アイドル設定とバス・クロック一覧

（a）μPD444016L-A8との接続

BUSCLK	データ・ウエイト	アドレス・セットアップ・ウエイト	アイドル・ステート	制限されるタイミング注
BUSCLK≦50 MHz	0	0	0	t_{WRDH}, t_{DRDOD}
50 MHz＜BUSCLK≦66 MHz	0	0	1	－

（b）μPD444016L-A10との接続

BUSCLK	データ・ウエイト	アドレス・セットアップ・ウエイト	アイドル・ステート	制限されるタイミング注
BUSCLK≦45 MHz	0	0	0	t_{WRDH}, t_{DRDOD}
45 MHz＜BUSCLK≦66 MHz	0	0	1	－

（c）μPD444016L-A12との接続

BUSCLK	データ・ウエイト	アドレス・セットアップ・ウエイト	アイドル・ステート	制限されるタイミング注
BUSCLK≦41 MHz	0	0	0	t_{WRDH}, t_{DRDOD}
41 MHz＜BUSCLK≦65 MHz	0	0	1	t_{SRDID}
65 MHz＜BUSCLK≦66 MHz	1	0	1	－

注　2. 1. 1　μPD444008Lとの接続（8ビット・バス幅例）を参照してください。

2. 3　PROMとの接続

PROM（M27V800：1 M×8ビット／512 K×16ビット）を1つ（16ビット・バス幅，1 Mバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- **接続デバイス：**M27V800-100×1つ
- **使用**$\overline{\text{CS}}$**信号：**$\overline{\text{CS0}}$

  V850E/ME2**では外部メモリ空間の**0100000H-01FFFFFH**（ブロック**0**）に，**V850E2/ME3**では外部メモリ空間の**0000000H-00FFFFFH**（サブエリア**00**）に配列することとします。**

【接続の考え方と注意点】

- M27V800-100**のアドレス・バス（**A0-A18**）は**V850E/ME2, V850E2/ME3**の**A1-A19**に接続します。**
- M27V800-100**の**$\overline{\text{CE}}$, $\overline{\text{OE}}$, O0-O15**端子は**V850E/ME2, V850E2/ME3**の**$\overline{\text{CS0}}$, $\overline{\text{RD}}$, D0-D15**端子に接続します。**
- M27V800-100**の**$\overline{\text{BYTE}}$**端子は16ビット・バス幅で使用するためプルアップします。**
- PROM**へのアクセスは**SRAM**のリード動作と同等です。**
- $\overline{\text{CS0}}$**空間のバス幅は**MODE0, MODE1**端子の状態で決定されます。**LBS**レジスタへの書き込み時は**$\overline{\text{CS0}}$**空間の設定値を変更しないようにしてください。**

【レジスタの設定】

- $\overline{\text{CS0}}$**の使用エリアとデバイス・タイプ：**

  V850E/ME2：**ブロック**0，V850E2/ME3：**サブエリア**00，SRAM／**外部**I/O
- **ウエイト設定：**6**ウエイト**[注]
- **アドレス・セットアップ・ウエイト：**0**ウエイト（**1**ウエイト**[注]**）**
- **アイドル・ステート：**3**ステート**

注　ウエイト設定 = 5**（または**4**），アドレス・セットアップ・ウエイト** = 1**（または**2**）としても同等です。**

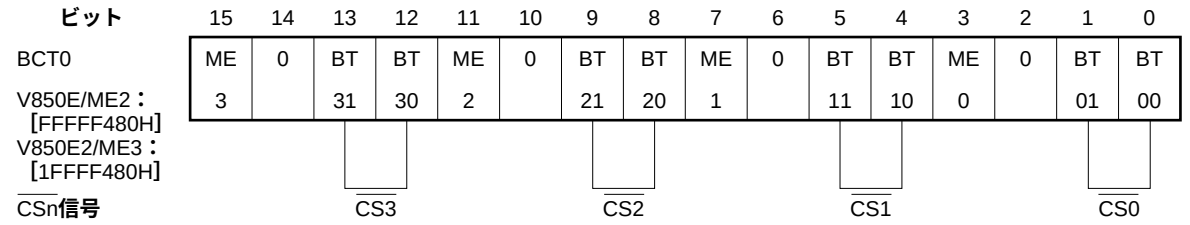
図2－27　**チップ・エリア選択コントロール・レジスタ**0**（**CSC0**）の設定**

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2：[FFFFF060H] V850E2/ME3：[1FFFF060H]	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00

V850E/ME2：**ブロック**0，V850E2/ME3：**サブエリア**00**アクセス時**$\overline{\text{CS0}}$**出力**

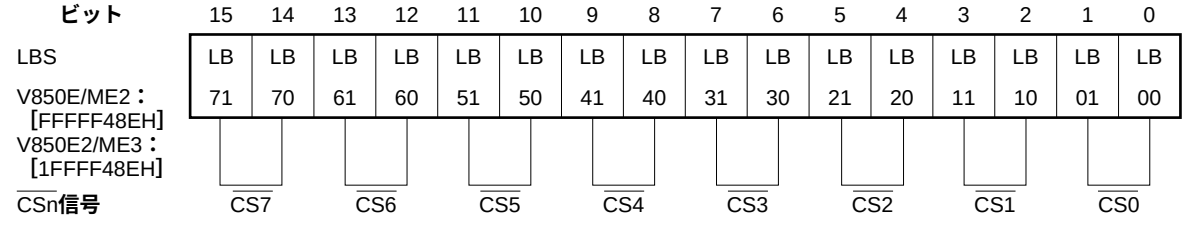
CSC0**の設定値：**xxxxxxx0xxxx0001B

図2－28　バス・サイクル・タイプ・コンフィギュレーション・レジスタ0（BCT0）の設定



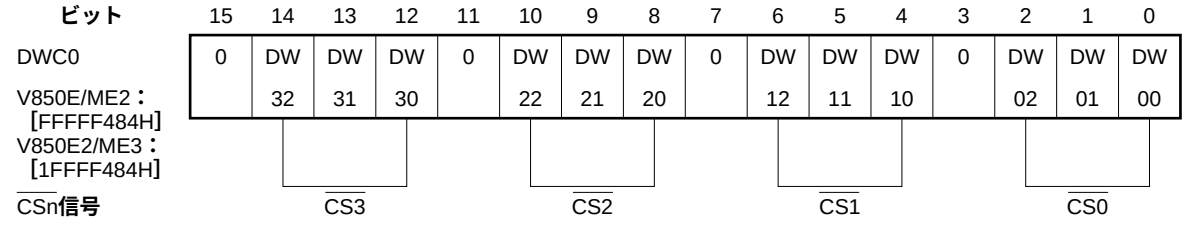
$\overline{CS0}$空間をメモリ・コントローラ動作許可，SRAM／外部I/Oに指定
BCT0の設定値：x0xxx0xxx0xx1000B

図2－29　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定



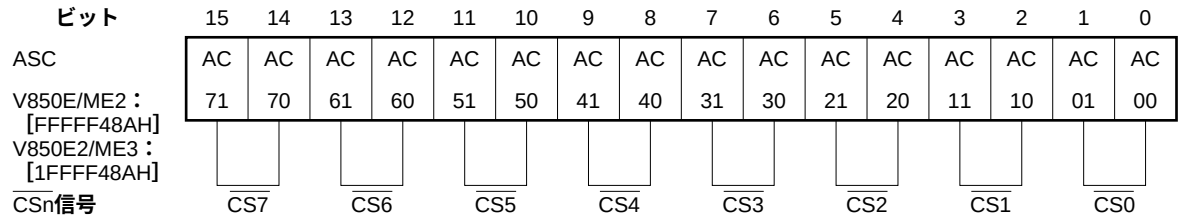
$\overline{CS0}$空間をデータ・バス幅16ビットに指定
LBSの設定値：xxxxxxxxxxxxxx01B

図2－30　データ・ウエイト・コントロール・レジスタ0（DWC0）の設定



$\overline{CS0}$空間のウエイト：6
DWC0の設定値：0xxx0xxx0xxx0110B

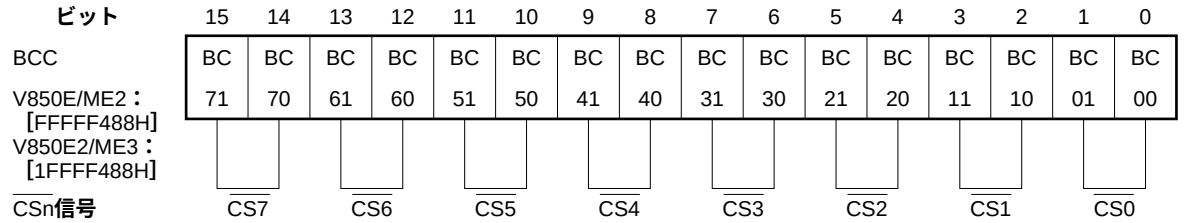
図2－31　アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）の設定



CS0空間のアドレス・セットアップ・ウエイト：0

ASCの設定値：xxxxxxxxxxxxxx00B

図2－32　バス・サイクル・コントロール・レジスタ（BCC）の設定



CS0空間のアイドル・ステート：3

BCCの設定値：xxxxxxxxxxxxxx11B

図2－33　M27V800-100接続回路例

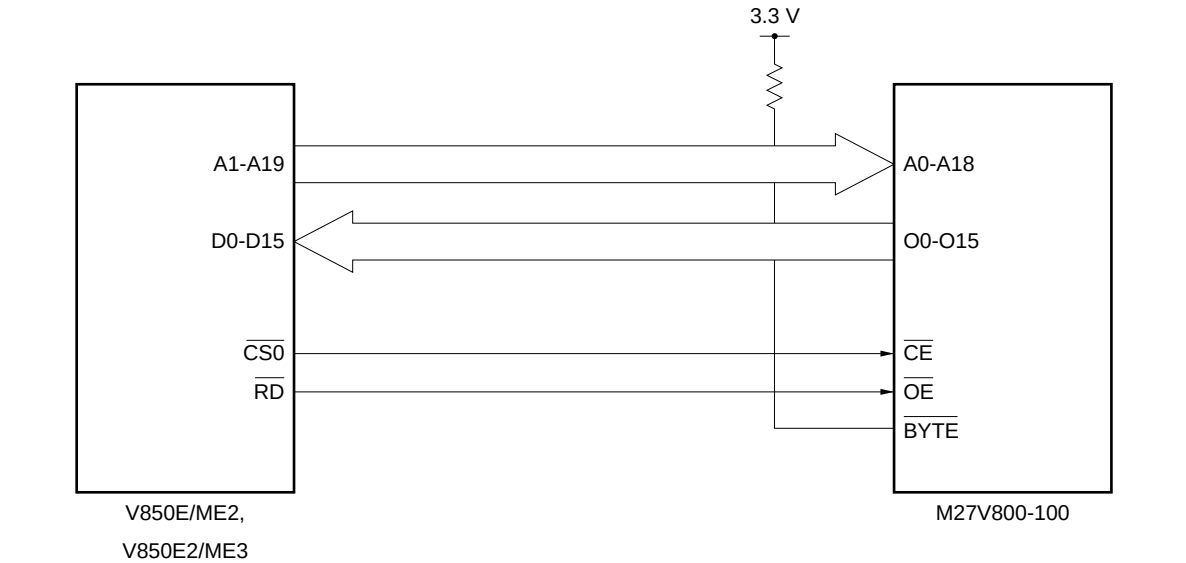
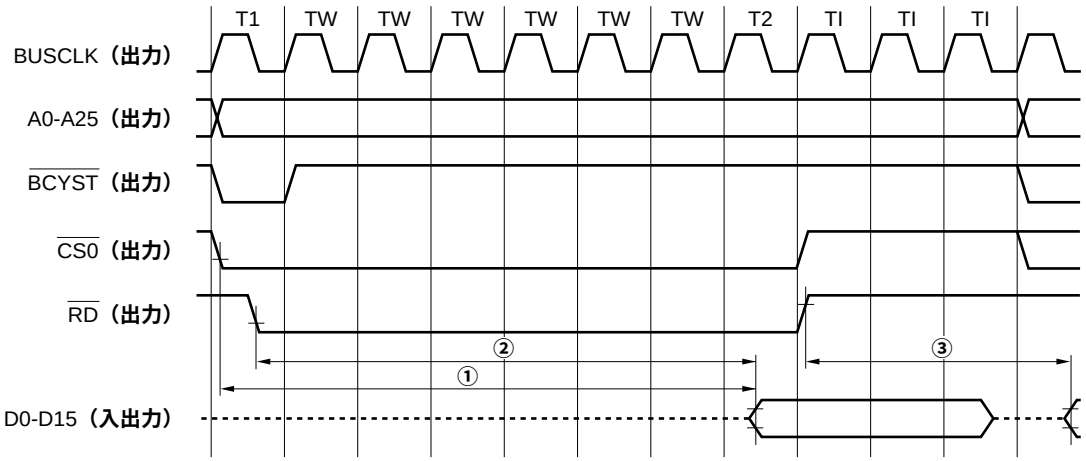


図2－34　M27V800-100のリード動作



① M27V800-100のアドレス，CEアクティブからの出力遅延時間：100 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ入力設定時間（対アドレス）の最大値

t_{SAID}（ns）　=（2＋w＋w_D＋w_{AS}）T－17
= 8×15.6－17；w = 0, w_D = 6, w_{AS} = 0, T = 15.6 ns
= 107.8 ns（＞100 ns）

② M27V800-100のOEアクティブからの出力遅延時間：50 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ入力設定時間（対RD）の最大値

t_{SRDID}（ns）　=（1.5＋w＋w_D）T－17
= 7.5×15.6－17；w = 0, w_D = 6, T = 15.6 ns
= 100 ns（＞50 ns）

③ M27V800-100のOEインアクティブからの出力フローティング遅延時間：45 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ出力遅延時間（対RD↑）の最小値

t_{DRDOD}（ns）=（0.5＋i）T－6
= 3.5×15.6－6；i = 3, T = 15.6 ns
= 48.6 ns（＞45 ns）

備考1.　破線はハイ・インピーダンスを示しています。

2.　T　：t_{CYK}（BUSCLK出力周期）
w　：WAITによるウエイト数
w_D　：DWC0, DWC1レジスタによるウエイト数
w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数
i　：アイドル・ステート数

2. 4　ページROMとの接続

ページROM（μPD23C16080BL：2 M×8ビット/1 M×16ビット）を1つ（16ビット・バス幅，2 Mバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- 接続デバイス：μPD23C16080BL×1つ
- 使用$\overline{\text{CS}}$信号：$\overline{\text{CS2}}$

  外部メモリ空間の0600000H-07FFFFFH（V850E/ME2：ブロック3，V850E2/ME3：サブエリア03）に配列することとします。

【接続の考え方と注意点】

- μPD23C16080BLの$\overline{\text{BYTE}}$端子は16ビット・バス幅で使用するためプルアップします。
- μPD23C16080BLのページ・サイズは16バイトのためPRCレジスタの設定は8×16ビットにします。
- オフページ時のウエイト数はDWC0レジスタで設定し，オンページ時のウエイト数とページ・サイズはPRCレジスタで設定します。

【レジスタの設定】

- $\overline{\text{CS2}}$の使用エリアとデバイス・タイプ：

  V850E/ME2：ブロック3，V850E2/ME3：サブエリア03，ページROM
- オフページ時のウエイト設定：5ウエイト
- オンページ時のウエイト設定：1ウエイト
- アドレス・セットアップ・ウエイト：0ウエイト
- アイドル・ステート：2ステート

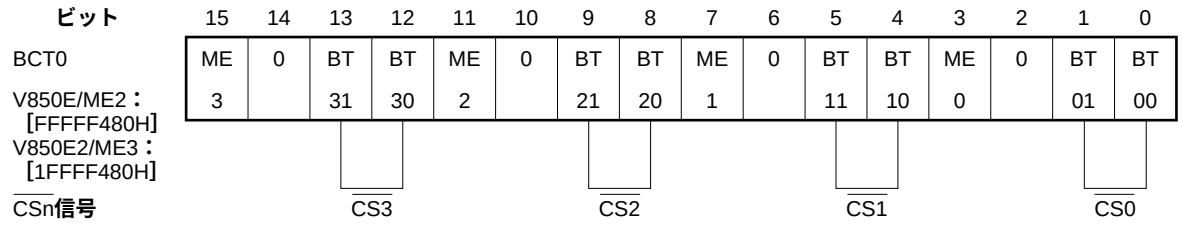
図2－35　チップ・エリア選択コントロール・レジスタ0（CSC0）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2：［FFFFF060H］ V850E2/ME3：［1FFFF060H］	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00

V850E/ME2：ブロック3，V850E2/ME3：サブエリア03アクセス時$\overline{\text{CS2}}$出力

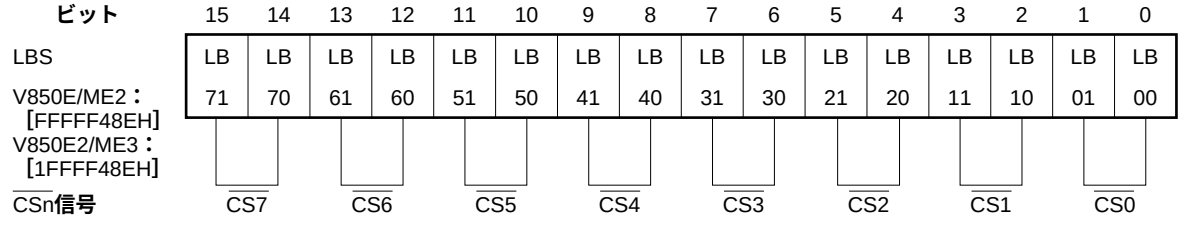
CSC0の設定値：xxxx1000xxxx0xxxB

図2－36　バス・サイクル・タイプ・コンフィギュレーション・レジスタ0（BCT0）の設定



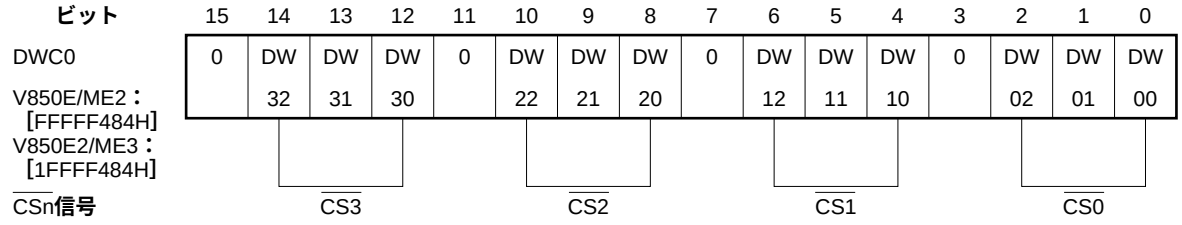
CS2空間をメモリ・コントローラ動作許可，ページROMに指定
BCT0の設定値：x0xx1001x0xxx0xxB

図2－37　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定



CS2空間をデータ・バス幅16ビットに指定
LBSの設定値：xxxxxxxxxx01xxxxB

図2－38　データ・ウエイト・コントロール・レジスタ0（DWC0）の設定



CS2空間のウエイト：5
DWC0の設定値：0xxx01010xxx0xxxB

図2－39　アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC	AC
V850E/ME2：［FFFFF48AH］ V850E2/ME3：［1FFFF48AH］	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
$\overline{\text{CS}}$n信号		$\overline{\text{CS7}}$		$\overline{\text{CS6}}$		$\overline{\text{CS5}}$		$\overline{\text{CS4}}$		$\overline{\text{CS3}}$		$\overline{\text{CS2}}$		$\overline{\text{CS1}}$		$\overline{\text{CS0}}$

$\overline{\text{CS2}}$**空間のアドレス・セットアップ・ウエイト：**0
ASC**の設定値：**xxxxxxxxxx00xxxxB

図2－40　バス・サイクル・コントロール・レジスタ（BCC）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC
V850E/ME2：［FFFFF488H］ V850E2/ME3：［1FFFF488H］	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
$\overline{\text{CS}}$n信号		$\overline{\text{CS7}}$		$\overline{\text{CS6}}$		$\overline{\text{CS5}}$		$\overline{\text{CS4}}$		$\overline{\text{CS3}}$		$\overline{\text{CS2}}$		$\overline{\text{CS1}}$		$\overline{\text{CS0}}$

$\overline{\text{CS2}}$**空間のアイドル・ステート：**2
BCC**の設定値：**xxxxxxxxxx10xxxxB

図2－41　ページROM**コンフィギュレーション・レジスタ（**PRC**）の設定**

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRC	0	PRW2	PRW1	PRW0	0	0	0	0	0	0	0	0	MA6	MA5	MA4	MA3
V850E/ME2［FFFFF49AH］ V850E2/ME3［1FFFF49AH］																

オンページ時のウエイト数：1
ページ・サイズ：8×16**ビット**
PRC**の設定値：**0001000000000001B

図2ー42　μPD23C16080BL接続回路例

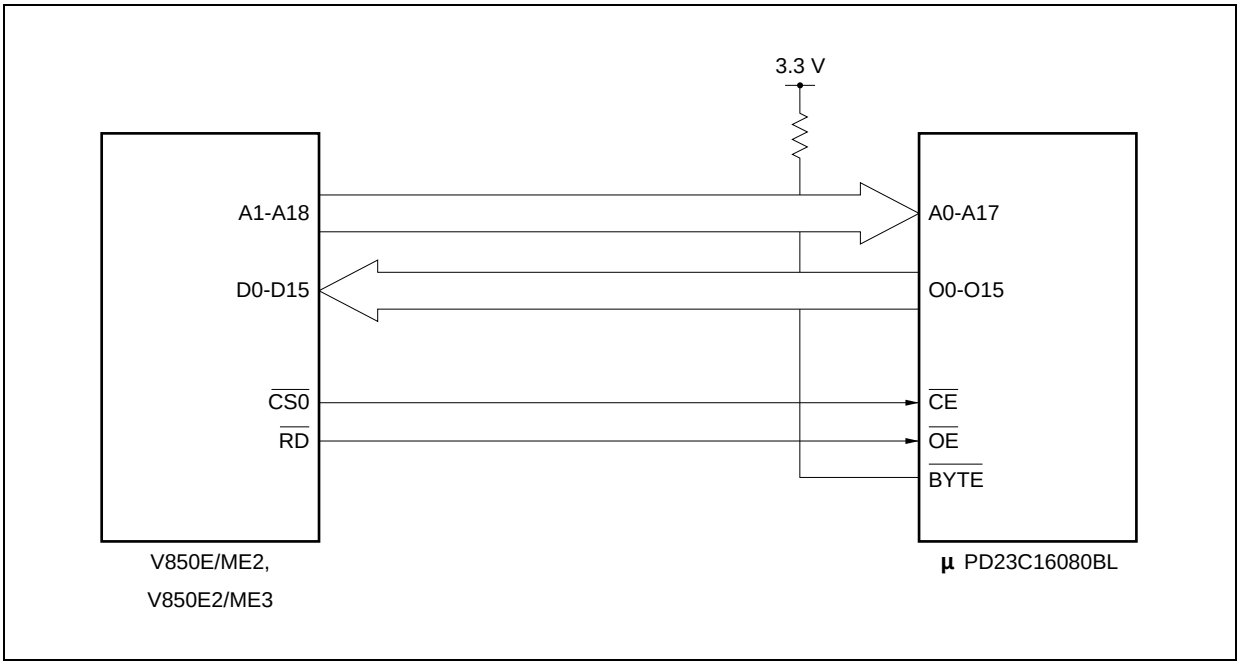
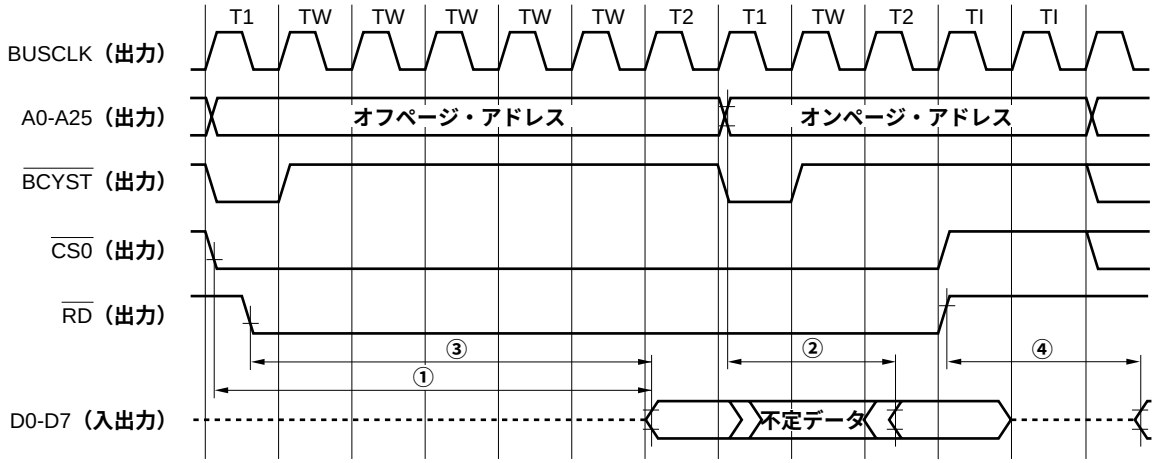


図2－43　μPD23C16080BLのリード動作



① μPD23C16080BLのオフページ時アドレス，CEアクティブからの出力遅延時間：85 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，オフページ・データ入力設定時間（対アドレス）の最大値

t_{SAID}（ns）＝（2＋w＋w_D＋w_{AS}）T－17
＝ 7×15.6－17；w = 0, w_D = 5, w_{AS} = 0, T = 15.6 ns
＝ 92.2 ns（＞85 ns）

② μPD23C16080BLのオンページ時アドレスからの出力遅延時間：25 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，オンページ・データ入力設定時間（対アドレス）の最大値

t_{SOAID}（ns）＝（2＋w＋w_{PR}）T－17
＝ 3×15.6－17；w = 0, w_{PR} = 1, T = 15.6 ns
＝ 29.8 ns（＞25 ns）

③ μPD23C16080BLのOEアクティブからの出力遅延時間：25 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，オフページ・データ入力設定時間（対RD）の最大値

t_{SRDID}（ns）＝（1.5＋w＋w_D）T－17
＝ 6.5×15.6－17；w = 0, w_D = 5, T = 15.6 ns
＝ 84.4 ns（＞25 ns）

④ μPD23C16080BLのOEインアクティブからの出力フローティング遅延時間：25 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ出力遅延時間（対RD↑）の最小値

t_{DRDOD}（ns）＝（0.5＋i）T－6
＝ 2.5×15.6－6；i = 2, T = 15.6 ns
＝ 33 ns（＞25 ns）

注意　ページ・ヒット時はRD信号をロウ・レベルに保って連続アクセスします。その間のアイドルは挿入されません。

備考1.　破線はハイ・インピーダンスを示しています。
2.　T　：t_{CYK}（BUSCLK出力周期）
w　：WAITによるウエイト数
w_D　：DWC0, DWC1レジスタによるウエイト数
w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数
w_{PR}　：PRCレジスタによるウエイト数
i　：アイドル・ステート数

2. 5　フラッシュ・メモリとの接続

フラッシュ・メモリ（MB29PL320：4 M×8ビット/2 M×16ビット）を1つ使用し，16ビット・バス幅，4 Mバイトの外部メモリ空間に接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- **接続デバイス：**MB29PL320TE80×1つ
- **使用**$\overline{\text{CS}}$**信号：**$\overline{\text{CS2}}$

  外部メモリ空間の0400000H-07FFFFFH（V850E/ME2：**ブロック**2, **ブロック**3，V850E2/ME3：**サブエリア**02, **サブエリア**03）**に配列することとします。**

【接続の考え方と注意点】

- MB29PL320TE80**のアドレス・バス**（A0-A20）**は**V850E/ME2, V850E2/ME3**の**A1-A21**に接続します。**
- MB29PL320TE80**の**D0-D15, $\overline{\text{CE}}$, $\overline{\text{OE}}$, $\overline{\text{WE}}$**端子は**V850E/ME2, V850E2/ME3**の**D0-D15, $\overline{\text{CS2}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$**端子に接続します。**
- MB29PL320TE80**の**RY/$\overline{\text{BY}}$**端子は**V850E/ME2, V850E2/ME3**のポート入力（**Pxx**）に接続することとします。**

備考1. RY/$\overline{\text{BY}}$**端子を**V850E/ME2, V850E2/ME3**のポート入力に接続することによりライト時／イレース時などの完了状態を監視することができます。**

2. Pxx = P10-P13, P20-P25, P50-P55, P65-P67, P72-P77, PAH6-PAH9, PAL0, PDH0-PDH15, PCD0-PCD3, PCM0-PCM5, PCS0, PCS1, PCS3-PCS7, PCT0-PCT3, PCT7

【レジスタの設定】

- $\overline{\text{CS2}}$**の使用エリアとデバイス・タイプ：**

  V850E/ME2：**ブロック**2, **ブロック**3，V850E2/ME3：**サブエリア**02, **サブエリア**03，SRAM／**外部**I/O
- **ウエイト設定：**2**ウエイト**[注]
- **アドレス・セットアップ・ウエイト：**3**ウエイト**[注]
- **アイドル・ステート：**2**ステート**

注　この接続回路例では，アドレス・セットアップ・ウエイト（t_{AS}**）の設定値を少なくして，ウエイト（**t_D**）の設定値を多くしても同等です。**

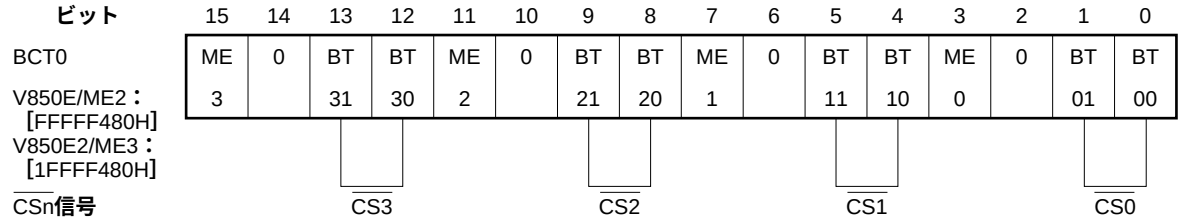
図2－44　**チップ・エリア選択コントロール・レジスタ**0（CSC0）**の設定**

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC0	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2：[FFFFF060H] V850E2/ME3：[1FFFF060H]	33	32	31	30	23	22	21	20	13	12	11	10	03	02	01	00

V850E/ME2：**ブロック**2, **ブロック**3，V850E2/ME3：**サブエリア**02, **サブエリア**03，**アクセス時**$\overline{\text{CS2}}$**出力**

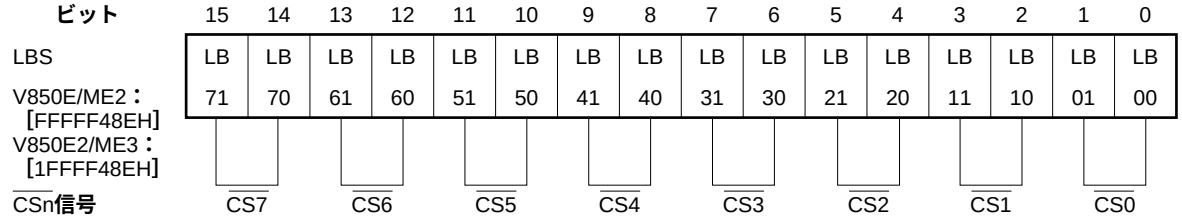
CSC0**の設定値：**xxxx1100xxxx00xxB

図2－45　バス・サイクル・タイプ・コンフィギュレーション・レジスタ0（BCT0）の設定



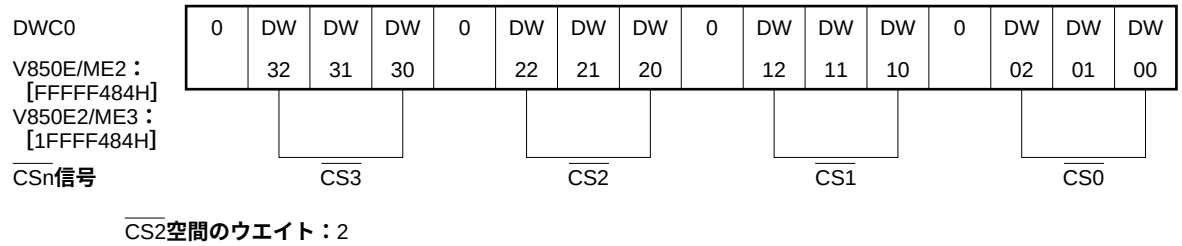
CS2空間をメモリ・コントローラ動作許可，SRAM／外部I/Oに指定
BCT0の設定値：x0xx1000x0xxx0xxB

図2－46　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定



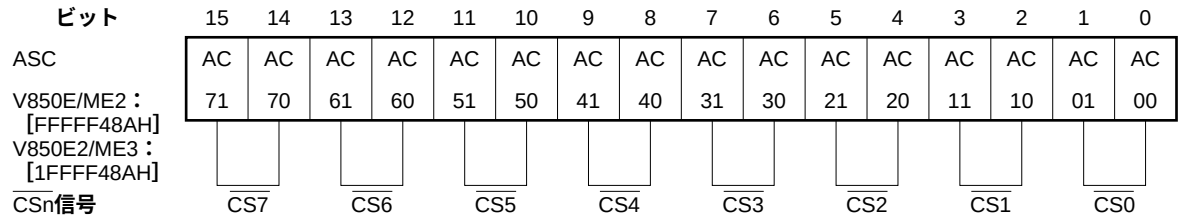
CS2空間をデータ・バス幅16ビットに指定
LBSの設定値：xxxxxxxxxx01xxxxB

図2－47　データ・ウエイト・コントロール・レジスタ0（DWC0）の設定



CS2空間のウエイト：2
DWC0の設定値：0xxx00100xxx0xxxB

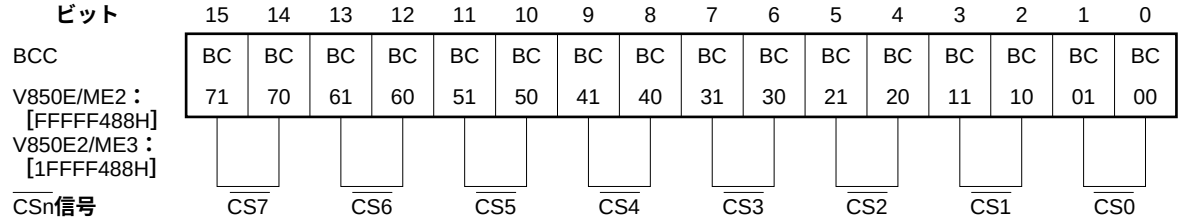
図2－48　アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）の設定



CS2空間のアドレス・セットアップ・ウエイト：3

ASCの設定値：xxxxxxxxxx11xxxxB

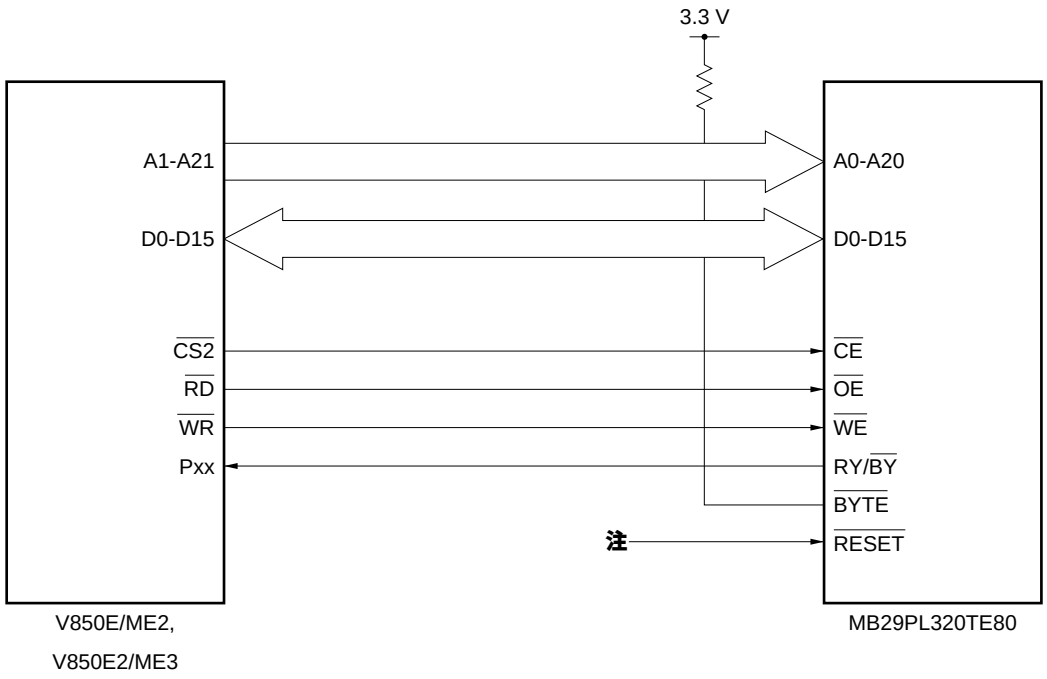
図2－49　バス・サイクル・コントロール・レジスタ（BCC）の設定



CS2空間のアイドル・ステート：2

BCCの設定値：xxxxxxxxxx10xxxxB

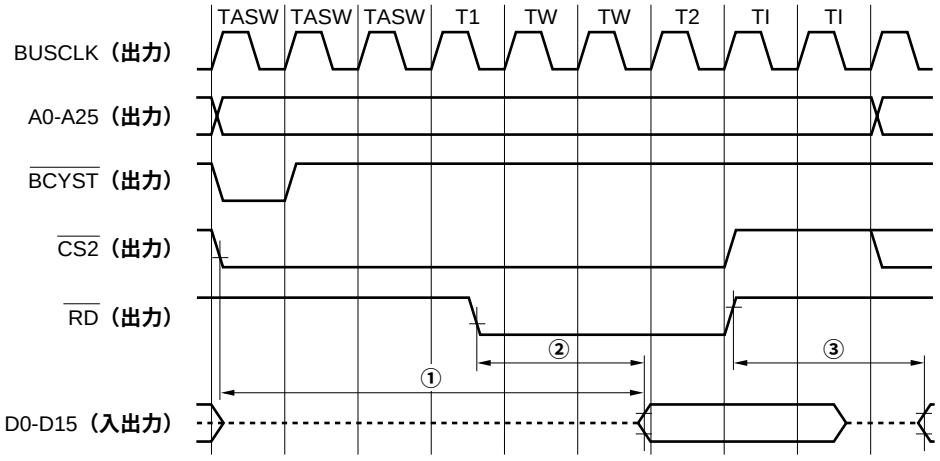
図2－50　MB29PL320TE80接続回路例



注　MB29PL320TE80の$\overline{\text{RESET}}$端子は，パワーオン・リセット信号など，システムで作成する信号を接続します。

備考　Pxx = P10-P13, P20-P25, P50-P55, P65-P67, P72-P77, PAH6-PAH9, PAL0, PDH0-PDH15, PCD0-PCD3, PCM0-PCM5, PCS0, PCS1, PCS3-PCS7, PCT0-PCT3, PCT7

図2－51　MB29PL320TE80のリード動作



① MB29PL320TE80のアドレス，CEアクティブからの出力遅延時間：80 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ入力設定時間（対アドレス）の最大値

t_{SAID}（ns）　=（2＋w＋w_D＋w_{AS}）T－17
= 7×15.6－17；w = 0, w_D = 2, w_{AS} = 3, T = 15.6 ns
= 92.2 ns（＞80 ns）

② MB29PL320TE80のOEアクティブからの出力遅延時間：30 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ入力設定時間（対RD）の最大値

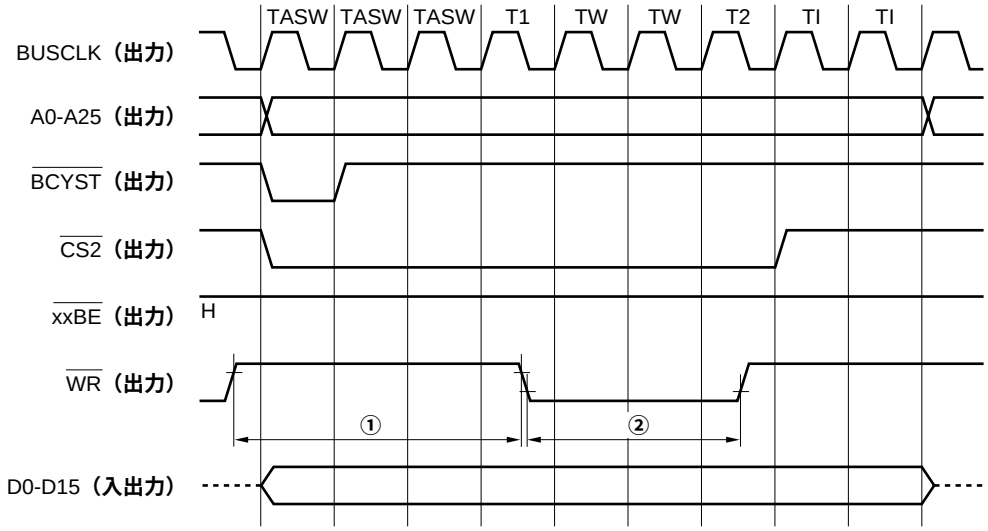
t_{SRDID}（ns）=（1.5＋w＋w_D）T－17
= 3.5×15.6－17；w = 0, w_D = 2, T = 15.6 ns
= 37.6 ns（＞30 ns）

③ MB29PL320TE80のOEインアクティブからの出力フローティング遅延時間：25 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ出力遅延時間（対RD↑）の最小値

t_{DRDOD}（ns）=（0.5＋i）T－6
= 2.5×15.6－6；i = 2, T = 15.6 ns
= 33 ns（＞25 ns）

備考1. 破線はハイ・インピーダンスを示しています。
2. T　：t_{CYK}（BUSCLK出力周期）
w　：WAITによるウエイト数
w_D　：DWC0, DWC1レジスタによるウエイト数
w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数
i　：アイドル・ステート数

図2－52　MB29PL320TE80のライト動作



① MB29PL320TE80の$\overline{WE}$ハイ・レベル・パルス幅：25 ns（MIN.）

V850E/ME2, V850E2/ME3の電気的特性より，$\overline{WR}$ハイ･レベル幅の最小値

t_{WWRH}（ns）＝（1＋i＋w_{AS}）T－5

＝ 6×15.6－5；i = 2, w_{AS} = 3, T = 15.6 ns

＝ 88.6 ns（＞25 ns）

② MB29PL320TE80の$\overline{WE}$アクティブ・レベル・パルス幅：35 ns（MIN.）

V850E/ME2, V850E2/ME3の電気的特性より，$\overline{WR}$ロウ･レベル幅の最小値

t_{WWRL}（ns）＝（1＋w＋w_D）T－5

＝ 3×15.6－5；w = 0, w_D = 2, T = 15.6 ns

＝ 41.8 ns（＞35 ns）

備考1. 破線はハイ・インピーダンスを示しています。

2. T　：t_{CYK}（BUSCLK出力周期）

w　：$\overline{WAIT}$によるウエイト数

w_D　：DWC0, DWC1レジスタによるウエイト数

w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数

i　：アイドル・ステート数

3. xx = LU, LL

2. 6　SDRAMとの接続

SDRAM（μPD45128163：2 M×16ビット×4バンク）を2つ（32 Mバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- **接続デバイス：**μPD45128163-A10×2つ
- **使用**$\overline{\text{CS}}$**信号：**$\overline{\text{CS3}}$

  外部メモリ空間の4000000H-5FFFFFFH**に配列することとします。**

  6000000H-7FFFFFFH**はイメージとなります。**

> **注意　この空間にプログラムは配列できません。プログラム・エリアとして使用する場合は**$\overline{\text{CS1}}$**空間に配列します。**

【接続の考え方と注意点】

- μ PD45128163-A10**の**$\overline{\text{CS}}$**端子は**V850E/ME2, V850E2/ME3**の**$\overline{\text{CS3}}$**端子に接続しますのでアクセス・タイミングは**SCR3**レジスタに設定します。**
- 32**ビット・バス幅で構成するため，**μ PD45128163-A10**のアドレス・バス（**A0-A11**）は**V850E/ME2, V850E2/ME3**の**A2-A13**に，**μPD45128163-A10**のデータ・バス（**DQ0-DQ15**）は**V850E/ME2, V850E2/ME3**の**D0-D15, D16-D31**にそれぞれ接続します。**

> **注意　この接続回路例では，データ・バスを**SDRAM**に直結していますが，データ・バスの負荷が大きい場合には**SDRAM**以外のメモリに接続するデータ・バスは双方向バッファなどで分断した信号を接続してください。双方向バッファは**$\overline{\text{CSn}}$**信号と**$\overline{\text{RD}}$**信号で制御します。**
>
> - **バッファのイネーブル端子　：**SDRAM**以外の**$\overline{\text{CS}}$**信号の論理和（**OR**）でイネーブル**
> - **バッファの方向制御　　：**$\overline{\text{RD}}$**信号がアクティブ・レベルの場合「メモリ→**V850E/ME2, V850E2/ME3**」**

- μPD45128163-A10**のアドレスはロウ・アドレスが**12**ビット，カラム・アドレスが**9**ビットなのでロウ・アドレス出力時に**V850E/ME2, V850E2/ME3**の**A2-A13**が次の状態になるようにアドレス・マルチプレクス幅（**9**ビット）を設定します。**

ロウ・アドレス

A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2
a22	–	–	–	–	–	–	–	–	–	a12	a11

- μ PD45128163-A10**のバンク・セレクト・アドレス（**BA0, BA1**）は，**V850E/ME2, V850E2/ME3**の上位アドレス（**32 M**バイト空間などで**A23, A24**が上位アドレス）を接続します。**
- μ PD45128163-A10**のリフレッシュ・サイクルは**4096**サイクル/**64 ms**なので，リフレッシュ・インターバルが**15.6 μs**以下になるように設定します。**

- μPD45128163-A10のAC特性は次のとおりです。したがって，SCR3レジスタはCASレーテンシ = 2，BCW = 2で使用し，アイドル・ステートは0を設定します。
- CASレーテンシ = 2での最大クロック：77 MHz（＞BUSCLK = 64 MHz）
- アクティブ・コマンドからリード／ライト・コマンドまでの最小時間：20 ns（＜BCW×2）
- クロック立ち上がりからのデータ・フロート遅延時間（MIN.）：7 ns

V850E/ME2, V850E2/ME3の電気的特性より，データ出力遅延時間（対BUSCLK↑）の最小値

t_{DSDOD}（ns）=（0.5＋i）T
= 0.5×15.6；i = 0, T = 15.6 ns
= 7.8 ns（＞7 ns）

- コマンド間サイクル時間：70 ns（＜15.6×5[注] ns）

注　アクティブ・コマンドBUSCLK数（1）＋コマンド間ウエイトBUSCLK数（1）＋レーテンシBUSCLK数（2）＋リード／ライト・コマンドBUSCLK数（1）の合計となります。

【レジスタの設定】

- $\overline{CS3}$のデバイス・タイプ：SDRAM
- $\overline{CS3}$空間のバス幅：32ビット
- アイドル・ステート：0ステート

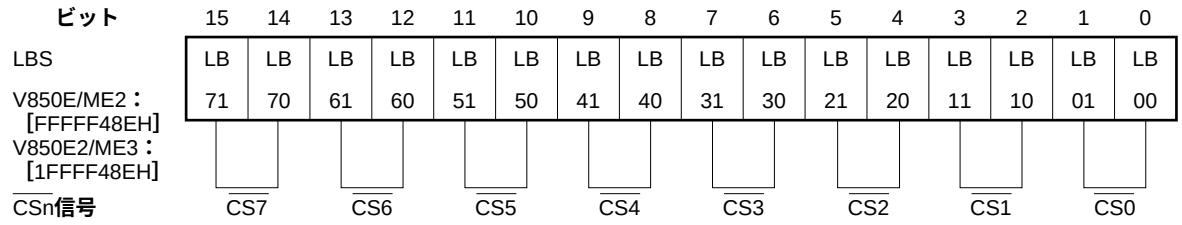
備考　SDRAMアクセスではDWC0, ASCレジスタの設定値は意味を持ちません。また，$\overline{CS3}$空間の領域は固定されているためCSC0レジスタの設定値は意味を持ちません。

図2－53　バス・サイクル・タイプ・コンフィギュレーション・レジスタ0（BCT0）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCT0	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT
V850E/ME2：［FFFFF480H］ V850E2/ME3：［1FFFF480H］	3		31	30	2		21	20	1		11	10	0		01	00
$\overline{CS}$n信号			$\overline{CS3}$				$\overline{CS2}$				$\overline{CS1}$				$\overline{CS0}$	

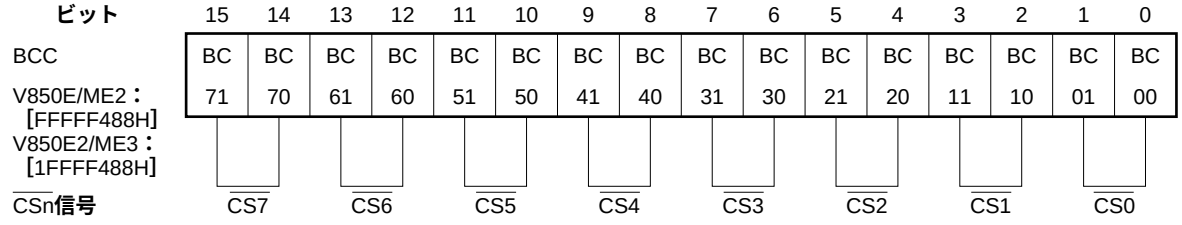
$\overline{CS3}$空間をメモリ・コントローラ動作許可，SDRAMに指定
BCT0の設定値：1011x0xxx0xxx0xxB

図2－54　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定



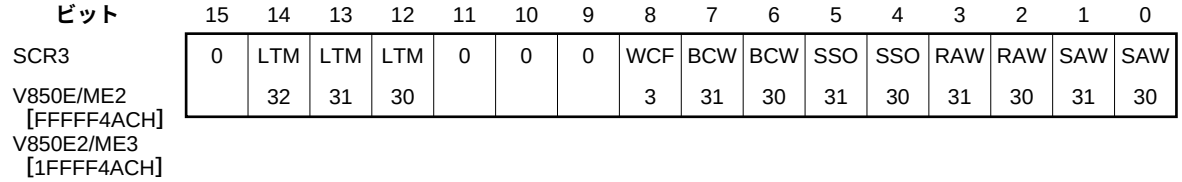
CS3空間をデータ・バス幅32ビットに指定
LBSの設定値：xxxxxxxx11xxxxxxB

図2－55　バス・サイクル・コントロール・レジスタ（BCC）の設定



CS3空間のアイドル・ステート：0
BCCの設定値：xxxxxxxx00xxxxxxB

図2－56　SDRAMコンフィギュレーション・レジスタ3（SCR3）の設定



LTM32-LTM30　：0, 1, 0（CASレーテンシ = 2）
BCW31, BCW30：1, 0（コマンド間ウエイト数：BCW = 2）
SSO31, SSO30　：1, 0（データ・バス幅 = 32ビット）
RAW31, RAW30：0, 1（ロウ・アドレス幅 = 12）
SAW31, SAW30：0, 1（アドレス・マルチプレクス幅 = 9ビット）
SCR3の設定値　：20A5H

備考　WCF3ビットはリードのみ可能です。

図2－57　SDRAMリフレッシュ・コントロール・レジスタ3（RFS3）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RFS3	REN	0	0	0	0	0	RCC	RCC	0	0	RIN	RIN	RIN	RIN	RIN	RIN
V850E/ME2 ［FFFFF4AEH］ V850E2/ME3 ［1FFFF4AEH］	3						31	30			35	34	33	32	31	30

REN3　　：1（リフレッシュ許可）
RCC31, RCC30　：0, 0（リフレッシュ・カウント・クロック：32/BUSCLK）
RIN35-RIN30　：0, 1, 1, 1, 1, 0（インターバル・ファクタ：31）
RFS3の設定値　：801EH

備考　32/BUSCLK = 0.5 μsなので上記設定でのリフレッシュ間隔は次のようになります。

リフレッシュ間隔 = 0.5×31

= 15.5 μs（＜15.6 μs）

図2－58　μPD45128163-A10接続回路例

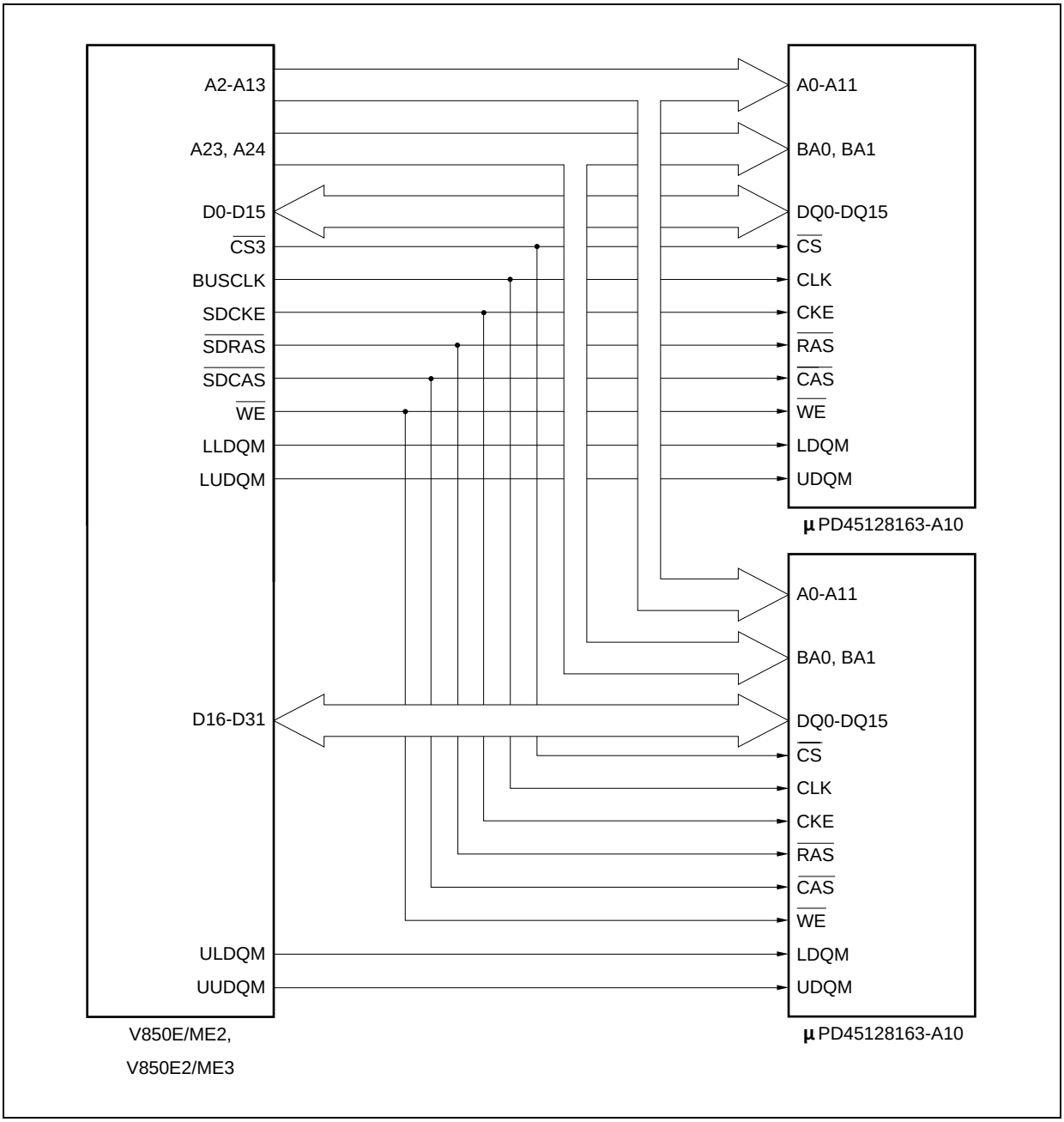
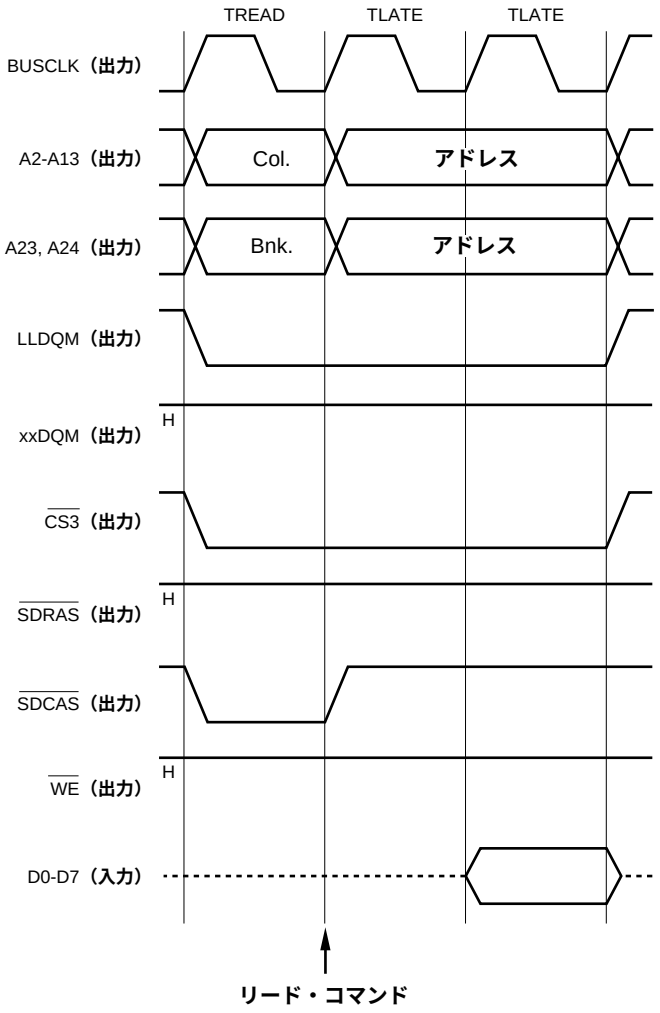


図2－59　μPD45128163-A10のリード動作（BCW = 2，レーテンシ = 2，アイドル・ステート挿入なし，オンページ・アクセス時，D0-D7に対するバイト・アクセス）

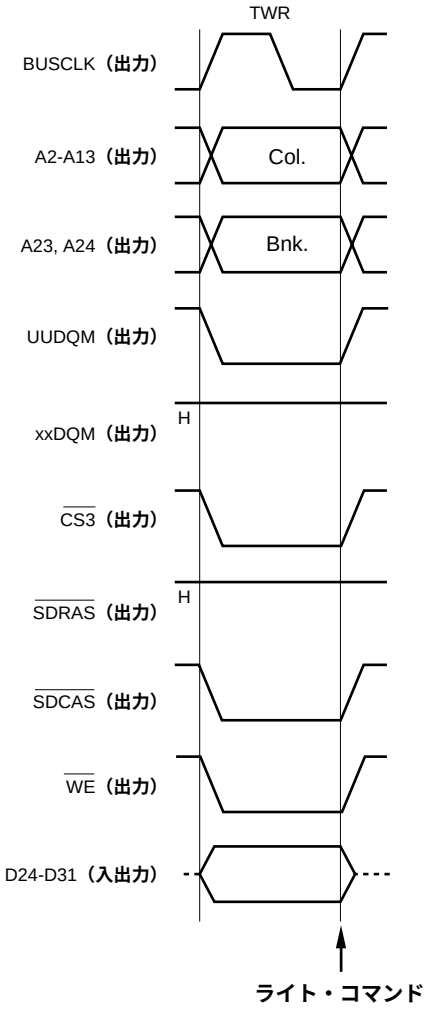


備考1.　**破線はハイ・インピーダンスを示します。**

2.　Bnk.：**バンク・アドレス**
　　Col.：**カラム・アドレス**

3.　xx = UU, UL, LU

図2－60　μPD45128163-A10のライト動作（BCW = 2，レーテンシ = 2，アイドル・ステート挿入なし，オンページ・アクセス時，D24-D31に対するバイト・アクセス）

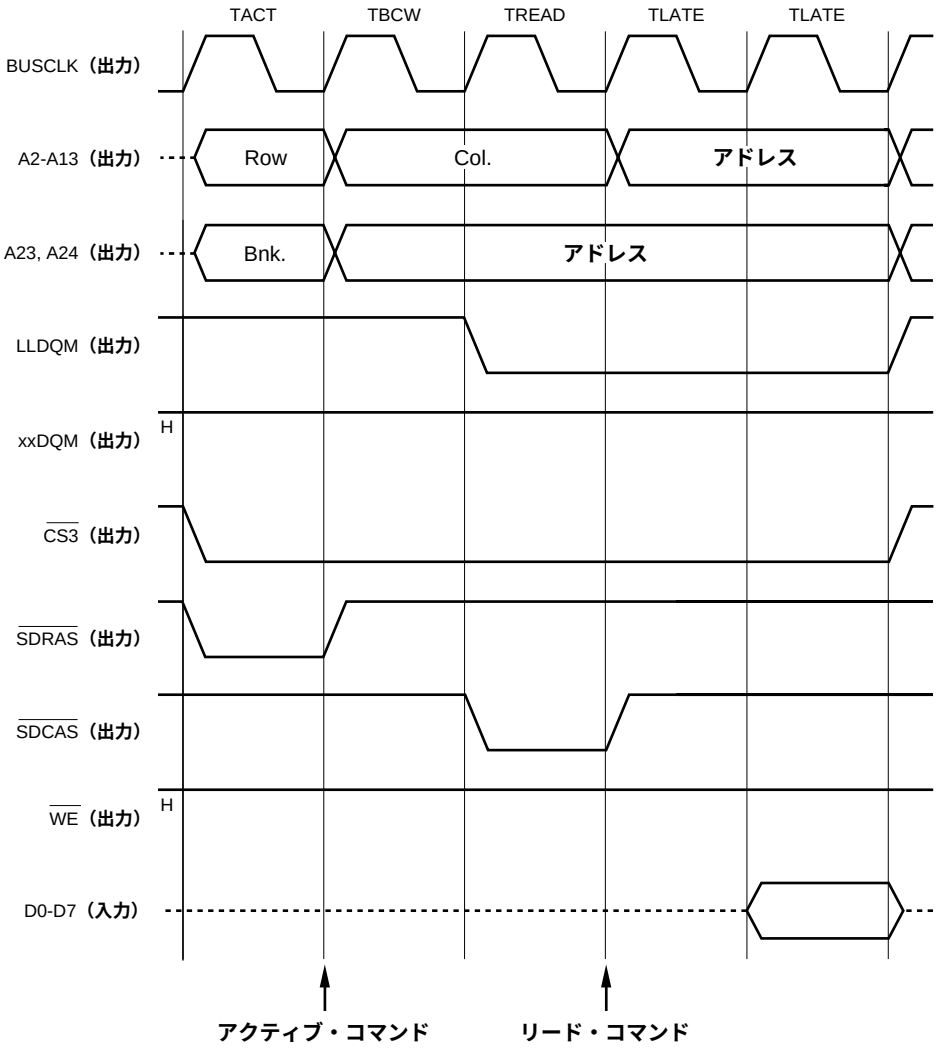


備考1.　**破線はハイ・インピーダンスを示します。**

2.　Bnk.：**バンク・アドレス**
　　Col.：**カラム・アドレス**

3.　xx = UL, LU, LL

図2－61　μPD45128163-A10のリード動作（BCW = 2，レーテンシ = 2，アイドル・ステート挿入なし，オフページ・アクセス時，D0-D7に対するバイト・アクセス）

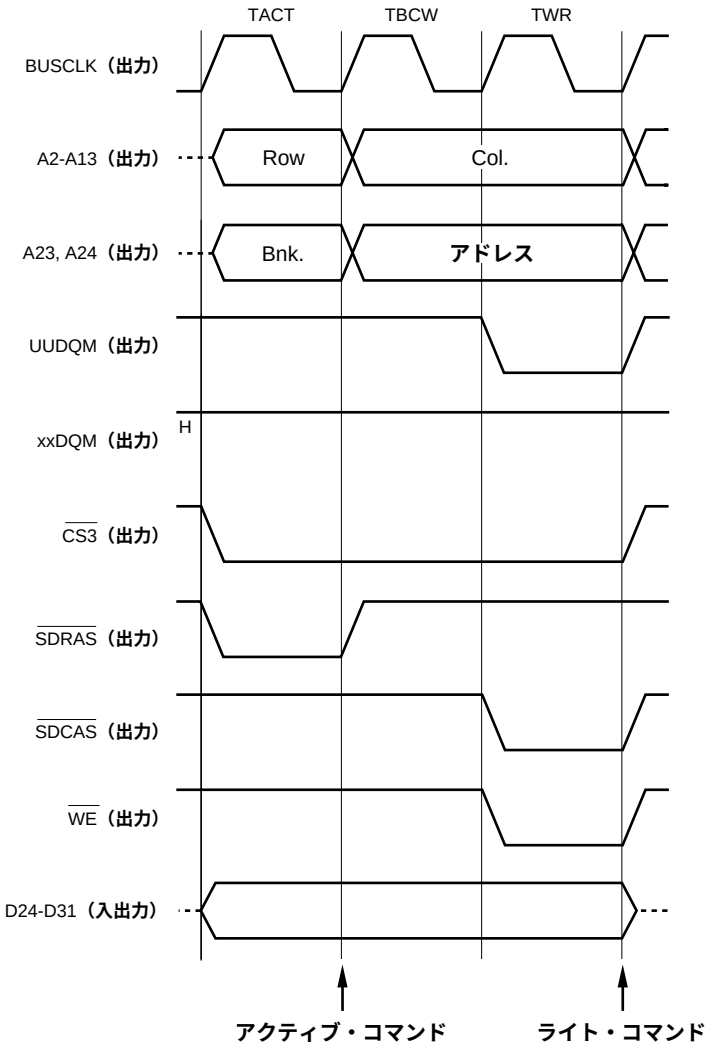


備考1. **破線はハイ・インピーダンスを示します。**

2. Bnk.：**バンク・アドレス**
   Col.：**カラム・アドレス**
   Row：**ロウ・アドレス**

3. xx = UU, UL, LU

図2－62　μPD45128163-A10のライト動作（BCW = 2，レーテンシ = 2，アイドル・ステート挿入なし，オフページ・アクセス時，D24-D31に対するバイト・アクセス）

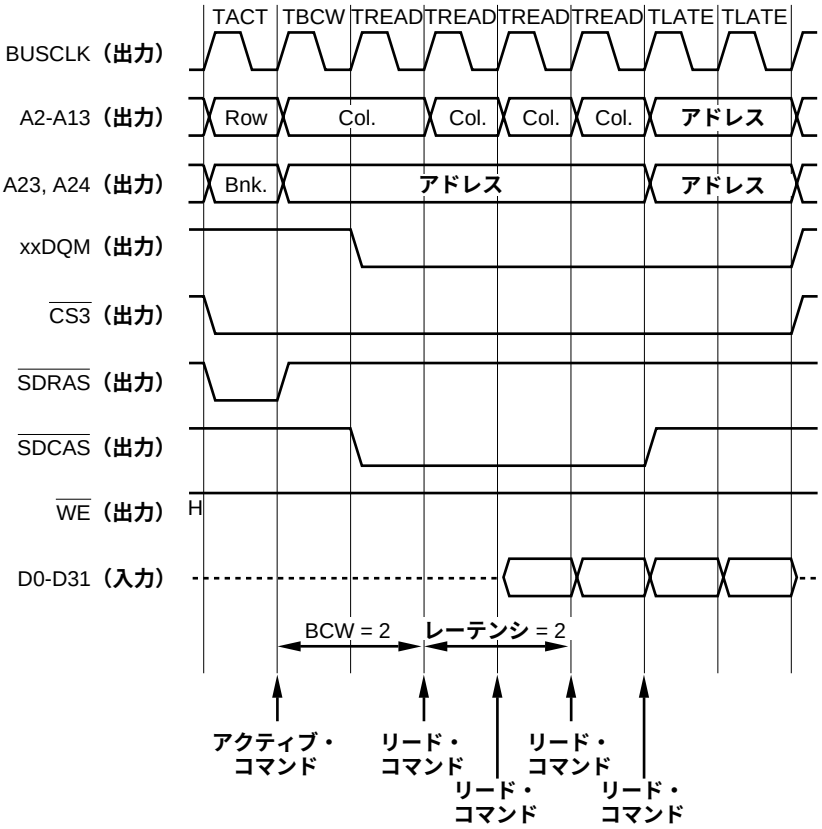


備考1. **破線はハイ・インピーダンスを示します。**

2. Bnk.：**バンク・アドレス**
   Col.：**カラム・アドレス**
   Row：**ロウ・アドレス**

3. xx = UL, LU, LL

図2－63　μPD45128163-A10のリード動作（BCW = 2，レーテンシ = 2，アイドル・ステート挿入なし，オフページ・アクセス時，命令キャッシュ・フィルによる4ワード・アクセス）



備考1.　**破線はハイ・インピーダンスを示します。**

2.　Bnk.：**バンク・アドレス**
　　Col.：**カラム・アドレス**
　　Row：**ロウ・アドレス**

3.　xx = UU, UL, LU, LL

2. 7　I/Oデバイスとの接続

8ビットI/Oデバイス（TL16C550C（UART））を接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- **接続デバイス：**TL16C550C×1つ
- **使用**$\overline{\text{CS}}$**信号：**$\overline{\text{CS7}}$

  V850E/ME2では外部メモリ空間のFE00000H-FFF7FFFH（ブロック7）に，V850E2/ME3では外部メモリ空間の1FE00000H-1FFF7FFFH（サブエリア07）に配列することとします。

  また，$\overline{\text{CS7}}$空間のバス幅は8ビットを指定します。

【接続の考え方と注意点】

- TL16C550Cの$\overline{\text{D0}}$-$\overline{\text{D7}}$, A0-A2, $\overline{\text{CS2}}$, $\overline{\text{RD1}}$, $\overline{\text{WR1}}$端子はV850E/ME2, V850E2/ME3の$\overline{\text{D0}}$-$\overline{\text{D7}}$, A0-A2, $\overline{\text{CS7}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$端子に接続します。
- TLC16550CのINTRPT端子はV850E/ME2, V850E2/ME3のポート入力（Pxx）に接続することとします。
- TL16C550Cの$\overline{\text{ADS}}$, $\overline{\text{WR2}}$, $\overline{\text{RD2}}$端子はプルダウン，$\overline{\text{CS0}}$, $\overline{\text{CS1}}$端子はプルアップします。

備考1.　Pxx = P10-P13, P20-P25, P50-P55, P65-P67, P72-P77, PAH0-PAH9, PDH0-PDH15, PCD0-PCD3, PCM0-PCM5, PCS0-PCS6, PCT0-PCT3, PCT7

2. この接続回路例では，$\overline{\text{CS7}}$空間を8ビット・バス幅としましたが，8ビット・バス幅，16ビット・バス幅，32ビット・バス幅でのV850E/ME2, V850E2/ME3のアドレスとTL16C550Cとの関係を次に示します。なお，いずれの場合もTL16C550Cへのアクセスは，バイト・アクセスで行います。

○8ビット・バス幅設定時（TL16C550CのA0-A2端子はV850E/ME2, V850E2/ME3のA0-A2端子に接続）

TL16C550Cのアクセス（A2-A0）	V850E/ME2アドレス	V850E2/ME3アドレス
受信バッファ／送信バッファ（000）	FE00000H	1FE00000H
Interrupt enable register（001）	FE00001H	1FE00001H
・ ・	・ ・	・ ・
Scratch register（111）	FE00007H	1FE00007H

V850E/ME2ではFE00008H-FFF7FFFH，V850E2/ME3では1FE00008H-1FFF7FFFHがイメージとなります。

○16ビット・バス幅設定時（TL16C550Cのアクセス（A2-A0）

TL16C550Cのアクセス（A2-A0）	V850E/ME2		V850E2/ME3	
	アドレス リトル・エンディアン設定時	アドレス ビッグ・エンディアン設定時	アドレス リトル・エンディアン設定時	アドレス ビッグ・エンディアン設定時
受信バッファ／送信バッファ（000）	FE00000H	FE00001H	1FE00000H	1FE00001H
Interrupt enable register（001）	FE00002H	FE00003H	1FE00002H	1FE00003H
・ ・	・ ・	・ ・	・ ・	・ ・
Scratch register（111）	FE0000EH	FE0000FH	1FE0000EH	1FE0000FH

V850E/ME2ではFE00010H-FFF7FFFH，V850E2/ME3では1FE00010H-1FFF7FFFHがイメージとなります。

○32ビット・バス幅設定時（TL16C550Cのアクセス（A2-A0）

TL16C550Cのアクセス（A2-A0）	V850E/ME2		V850E2/ME3	
	アドレス リトル・エンディアン設定時	アドレス ビッグ・エンディアン設定時	アドレス リトル・エンディアン設定時	アドレス ビッグ・エンディアン設定時
受信バッファ／送信バッファ（000）	FE00000H	FE00003H	1FE00000H	1FE00003H
Interrupt enable register（001）	FE00004H	FE00007H	1FE00004H	1FE00007H
・ ・	・ ・	・ ・	・ ・	・ ・
Scratch register（111）	FE0001CH	FE0001FH	1FE0001CH	1FE0001FH

V850E/ME2ではFE00020H-FFF7FFFH，V850E2/ME3では1FE00020H-1FFF7FFFHがイメージとなります。

【レジスタの設定】

- $\overline{CS7}$の使用エリアとデバイス・タイプ：

  V850E/ME2：ブロック7，V850E2/ME3：サブエリア07，SRAM／外部I/O

- ウエイト設定：3ウエイト
- アドレス・セットアップ・ウエイト：1ウエイト
- アイドル・ステート：2ステート

図2－64　チップ・エリア選択コントロール・レジスタ1（CSC1）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSC1	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS	CS
V850E/ME2 【FFFFF062H】 V850E2/ME3 【1FFFF062H】	43	42	41	40	53	52	51	50	63	62	61	60	73	72	71	70

V850E/ME2：**ブロック**7，V850E2/ME3：**サブエリア**07**アクセス時**$\overline{\text{CS7}}$**出力**

CSC1**の設定値：**xxxxxxx0xxxx0001B

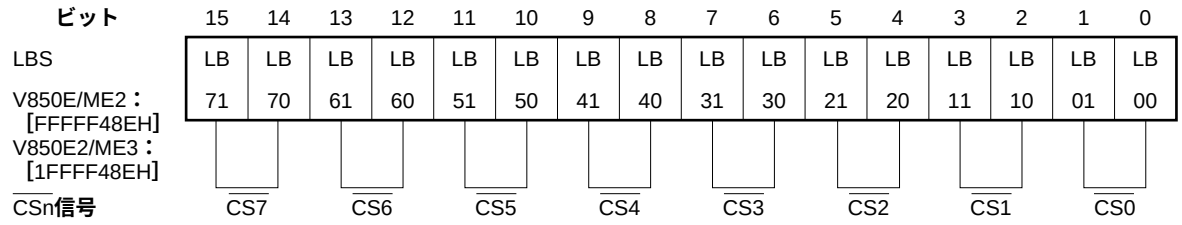
図2－65　バス・サイクル・タイプ・コンフィギュレーション・レジスタ1（BCT1）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCT1	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT	ME	0	BT	BT
V850E/ME2 【FFFFF482H】 V850E2/ME3 【1FFFF482H】	7		71	70	6		61	60	5		51	50	4		41	40
$\overline{\text{CS}}$**n信号**			$\overline{\text{CS7}}$				$\overline{\text{CS6}}$				$\overline{\text{CS5}}$				$\overline{\text{CS4}}$	

$\overline{\text{CS7}}$**空間をメモリ・コントローラ動作許可，**SRAM／**外部**I/O**に指定**

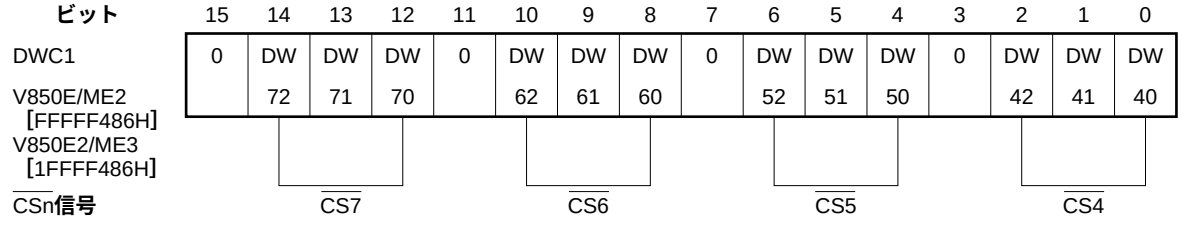
BCT1**の設定値：**1000x0xxx0xxx0xxB

図2－66　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定



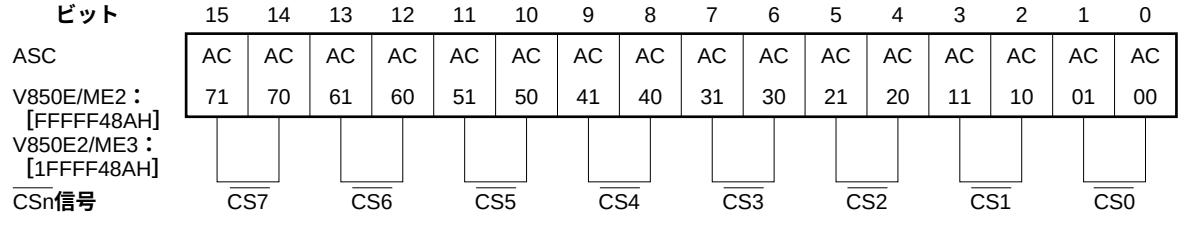
$\overline{CS7}$空間をデータ・バス幅8ビットに指定
LBSの設定値：00xxxxxxxxxxxxxxB

図2－67　データ・ウエイト・コントロール・レジスタ1（DWC1）の設定



$\overline{CS7}$空間のウエイト：3
DWC1の設定値：00110xxx0xxx0xxxB

図2－68　アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）の設定



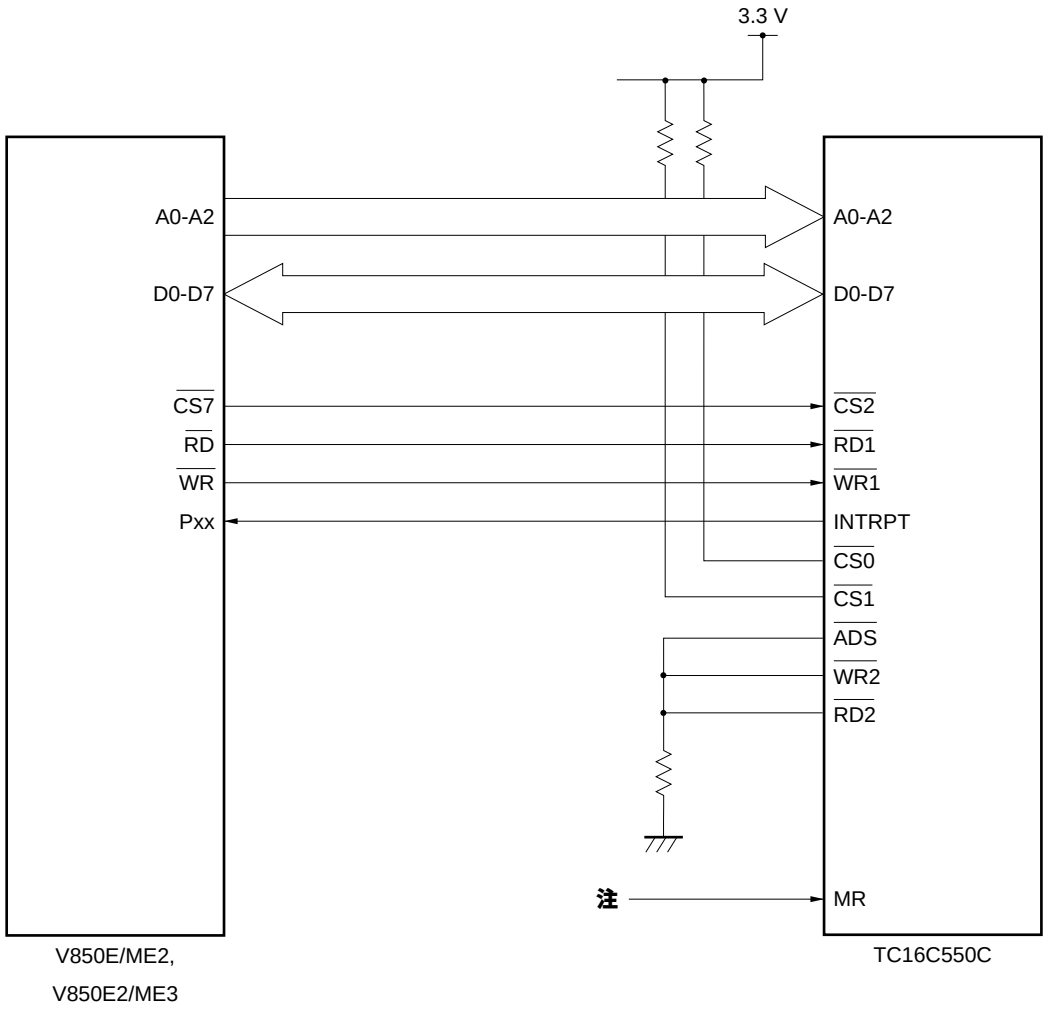
$\overline{CS7}$空間のアドレス・セットアップ・ウエイト：1
ASCの設定値：01xxxxxxxxxxxxxxB

図2－69　バス・サイクル・コントロール・レジスタ（BCC）の設定

ビット	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC	BC
V850E/ME2：［FFFFF488H］ V850E2/ME3：［1FFFF488H］	71	70	61	60	51	50	41	40	31	30	21	20	11	10	01	00
$\overline{\text{CS}}$n信号		$\overline{\text{CS7}}$		$\overline{\text{CS6}}$		$\overline{\text{CS5}}$		$\overline{\text{CS4}}$		$\overline{\text{CS3}}$		$\overline{\text{CS2}}$		$\overline{\text{CS1}}$		$\overline{\text{CS0}}$

$\overline{\text{CS7}}$**空間のアイドル・ステート：**2
BCC**の設定値：**10xxxxxxxxxxxxxxB

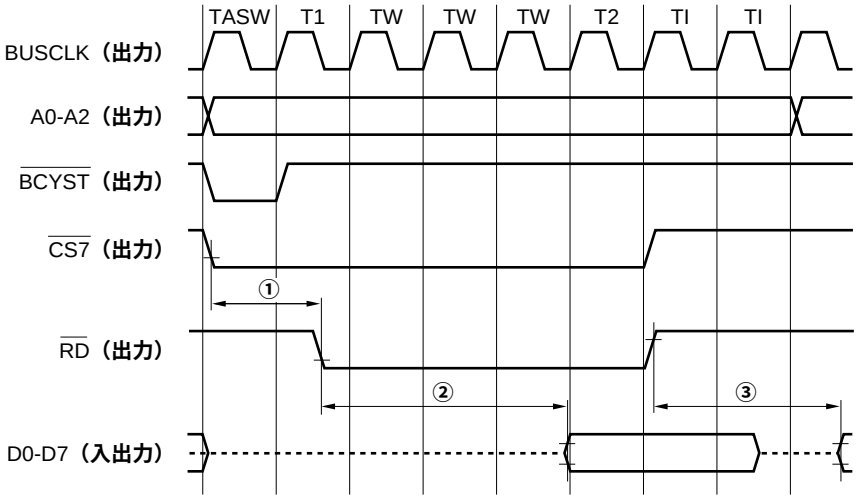
図2－70　TL16C550C接続回路例



注　TL16C550CのMR端子は，システムで作成するI/Oデバイス・リセット信号を接続します。

備考　Pxx = P10-P13, P20-P25, P50-P55, P65-P67, P72-P77, PAH0-PAH9, PDH0-PDH15,
PCD0-PCD3, PCM0-PCM5, PCS0-PCS6, PCT0-PCT3, PCT7

図2－71　TL16C550Cのリード動作



① TL16C550Cのアドレス，CS2アクティブからRD1アクティブまでの遅延時間：7 ns（MIN.）
V850E/ME2, V850E2/ME3の電気的特性より，アドレス，CS2→RD↓遅延時間の最小値

t_{DARD}（ns）＝（0.5＋w_{AS}）T－7.5
＝ 1.5×15.6－7.5；w_{AS} = 1, T = 15.6 ns
＝ 15.9 ns（＞7 ns）

② TL16C550CのRD1アクティブからのデータ出力遅延時間：45 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ入力設定時間（対RD）の最大値

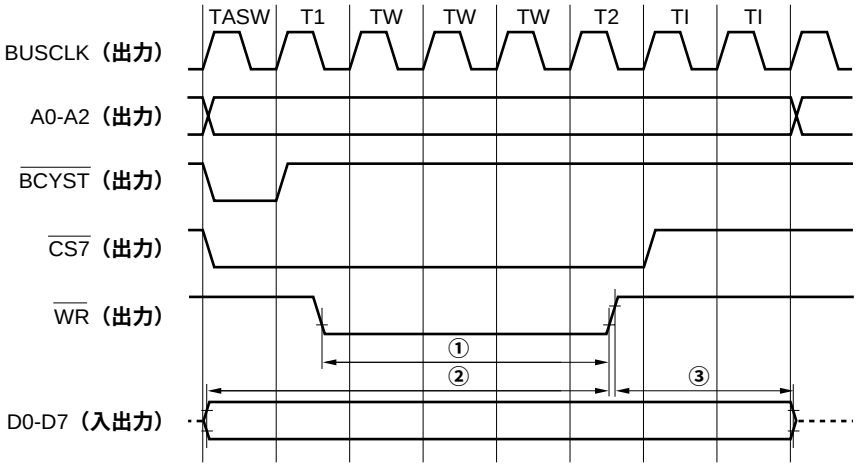
t_{SRDID}（ns）＝（1.5＋w＋w_D）T－17
＝ 4.5×15.6－17；w = 0, w_D = 3, T = 15.6 ns
＝ 53.2 ns（＞45 ns）

③ TL16C550CのRD1インアクティブからの出力フローティング遅延時間：20 ns（MAX.）
V850E/ME2, V850E2/ME3の電気的特性より，データ出力遅延時間（対RD↑）の最小値

t_{DRDOD}（ns）＝（0.5＋i）T－6
＝ 2.5×15.6－6；i = 2, T = 15.6 ns
＝ 33 ns（＞20 ns）

備考1. 破線はハイ・インピーダンスを示しています。
2. T　：t_{CYK}（BUSCLK出力周期）
w　：WAITによるウエイト数
w_D　：DWC0, DWC1レジスタによるウエイト数
w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数
i　：アイドル・ステート数

図2－72　TL16C550Cのライト動作



① TL16C550Cの$\overline{WR1}$アクティブ・パルス幅：40 ns（MIN.）

V850E/ME2, V850E2/ME3の電気的特性より，$\overline{WR}$ロウ・レベル幅の最小値

t_{WWRL}（ns）＝（1＋w＋w_D）T－5

＝ 4×15.6－5；w = 0, w_D = 3, T = 15.6 ns

＝ 57.4 ns（＞40 ns）

② TL16C550Cの$\overline{WR1}$立ち上がり時のデータ設定時間：15 ns（MIN.）

V850E/ME2, V850E2/ME3の電気的特性より，データ出力設定時間（対$\overline{WR}$↑）の最小値

t_{SODWR}（ns）＝（1.5＋w＋w_D＋w_{AS}）T－5

＝ 5.5×15.6－5；w = 0, w_D = 3, w_{AS} = 1, T = 15.6 ns

＝ 80.8 ns（＞15 ns）

③ TL16C550Cの$\overline{WR1}$立ち上がり時のデータ保持時間：5 ns（MIN.）

V850E/ME2, V850E2/ME3の電気的特性より，データ出力保持時間（対$\overline{WR}$↑）の最小値

t_{HWROD}（ns）＝（0.5＋i）T－5.5

＝ 2.5×15.6－5.5；i = 2, T = 15.6 ns

＝ 33.5 ns（＞5 ns）

備考1. 破線はハイ・インピーダンスを示しています。

2. T　：t_{CYK}（BUSCLK出力周期）

w　：$\overline{WAIT}$によるウエイト数

w_D　：DWC0, DWC1レジスタによるウエイト数

w_{AS}　：ASCレジスタによるアドレス・セットアップ・ウエイト数

i　：アイドル・ステート数

2. 8　$\overline{\text{WAIT}}$端子の制御例

2. 8. 1　デュアル・ポートRAMとの接続

デュアル・ポートRAM（IDT70V09L：128 K×8ビット）を1つ（128 Kバイトの外部メモリ空間）接続する例を示します。

【回路構成】

- BUSCLK：64 MHz
- 接続デバイス：IDT70V09L15×1つ
- 使用$\overline{\text{CS}}$信号：$\overline{\text{CS5}}$

  V850E/ME2では外部メモリ空間のFA00000H-FA1FFFFH（ブロック5）に配列することとします。

  FA20000H-FBFFFFFHはイメージとなります。

  V850E2/ME3では外部メモリ空間の1FA00000H-1FA1FFFFH（サブエリア05）に配列することとします。

  1FA20000H-1FBFFFFFHはイメージとなります。

【回路設計上の考え方と注意点】

- 8ビットSRAM接続と同様にIDT70V09L15のA0L-A16L，I/O0L-I/O7L，$\overline{\text{CE0L}}$，$\overline{\text{OEL}}$，R/$\overline{\text{WL}}$端子はV850E/ME2, V850E2/ME3のA0-A16, D0-D7, $\overline{\text{CS5}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$端子にそれぞれ接続します。
- IDT70V09L15の$\overline{\text{BUSYL}}$端子は$\overline{\text{CE0L}}$信号のアクティブからデータ確定（$\overline{\text{BUSYL}}$信号がアクティブになるか否か）まで15 ns（MAX.）であり，V850E/ME2, V850E2/ME3の電気的特性より，$\overline{\text{WAIT}}$設定時間（対$\overline{\text{CS}}$信号）は「（1＋w_{AS}）T－17」なので，DWC1レジスタの設定は2ウエイトとします。

  注意　この接続回路例では，アドレス・セットアップ・ウエイト = 2でも条件を満たしますが，その他のAC特性との関係上データ・ウエイト = 2とします。

- IDT70V09L15は，$\overline{\text{BUSYL}}$信号がアクティブになった場合，$\overline{\text{BUSYL}}$信号の立ち上がり（$\overline{\text{BUSYL}}$信号解除）からデータ確定まで15 ns（MAX.）なので，ハードウエアで1クロック遅延させた信号と$\overline{\text{BUSYL}}$信号の論理和（OR）をV850E/ME2, V850E2/ME3の$\overline{\text{WAIT}}$信号に接続します。
- IDT70V09L15のM/S端子はプルアップ，$\overline{\text{SEML}}$端子と$\overline{\text{INTL}}$端子は使用しません。

備考　M/S　：IDT70V09L15を複数接続する場合にマスタ／スレーブの設定をします。この接続回路例では，1つだけ使用するためマスタに設定します。

$\overline{\text{SEML}}$　：セマフォ機能を使用する場合のセレクト端子です。この接続回路例では，セマフォ機能を使用しないこととします。

$\overline{\text{INTL}}$　：ライト側プロセッサが特定エリアをアクセスした場合にアクティブになる割り込み要求入力端子です。この接続回路例では，使用しないこととしてオープンにします。

【レジスタの設定】

この接続回路例では，レジスタ設定値は簡略に説明します。

表2－2　IDT70V09L15との接続

レジスタ名称	設定値	機　　能
CSC1	xxxx0100xxxxx0xxB	V850E/ME2：ブロック5，V850E2/ME3：サブエリア05アクセス時$\overline{\text{CS5}}$出力
BCT1	x0xxx0xx1000x0xxB	$\overline{\text{CS5}}$：メモリ・コントローラ許可，SRAM／外部I/O
LBS	xxxx00xxxxxxxxxxB	$\overline{\text{CS5}}$：8ビット
DWC1	0xxx0xxx00100xxxB	$\overline{\text{CS5}}$：データ・ウエイト2
ASC	xxxx00xxxxxxxxxxB	$\overline{\text{CS5}}$：アドレス・セットアップ・ウエイト0
BCC	xxxx01xxxxxxxxxxB	$\overline{\text{CS5}}$：アイドル・ステート1

図2－73　IDT70V09L15接続回路例

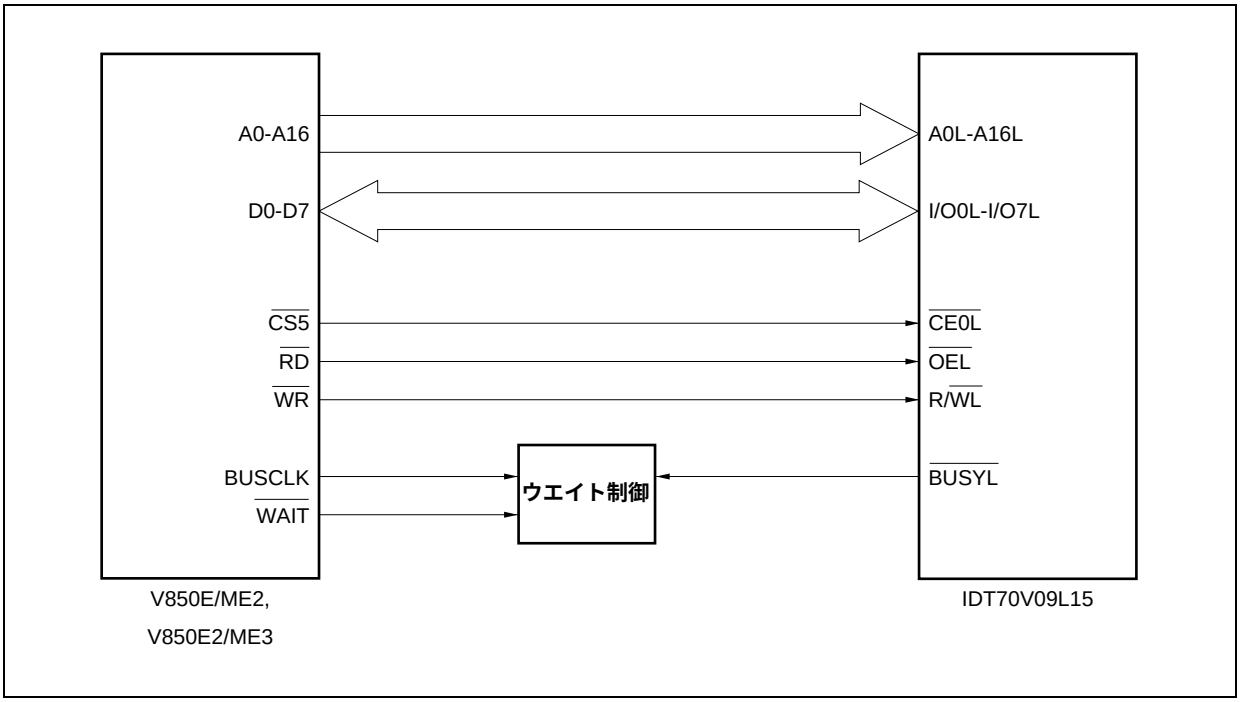


図2－74　ウエイト制御部の回路構成

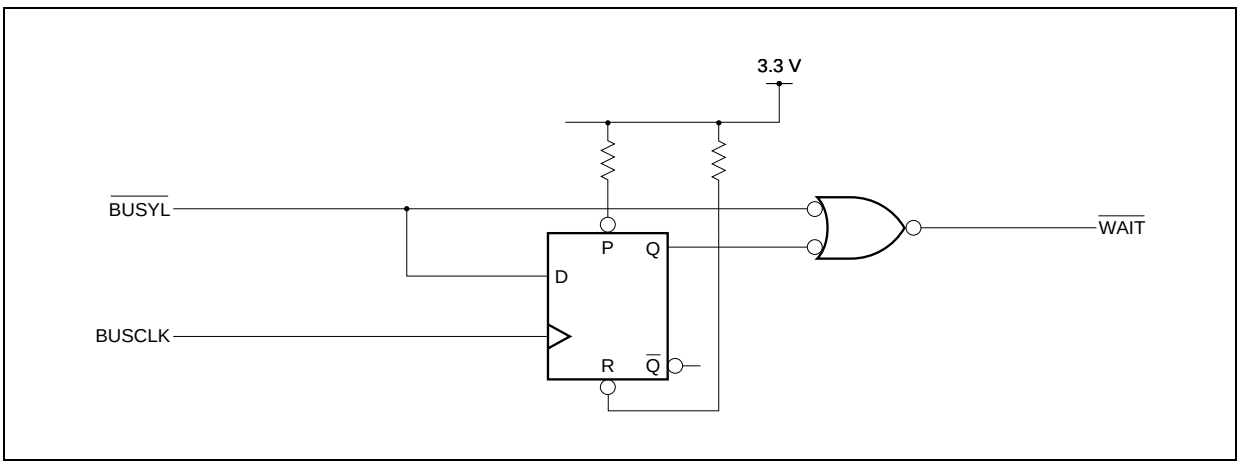
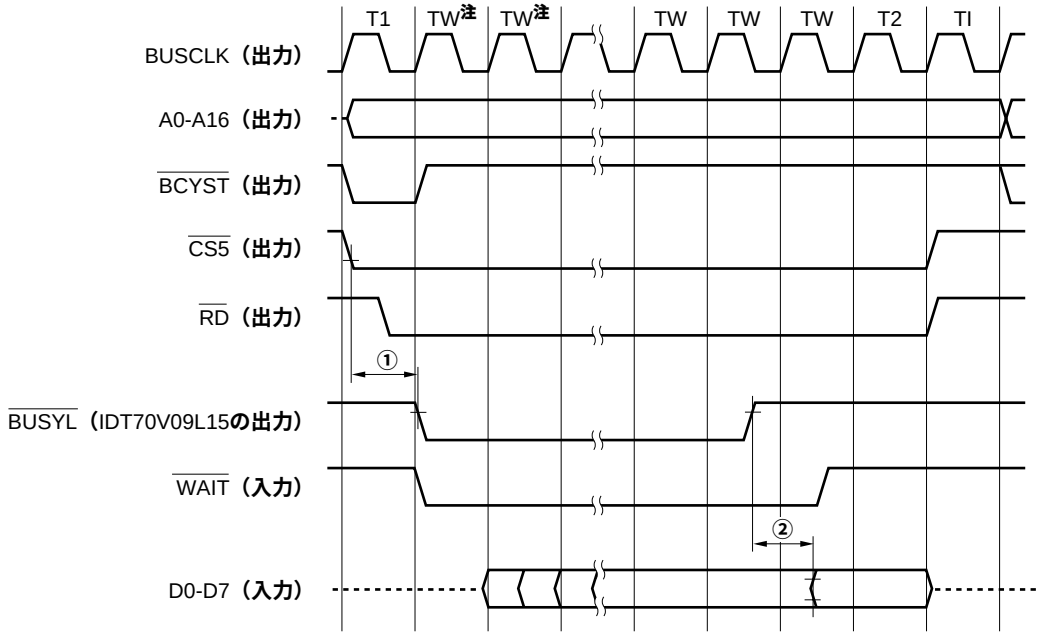


図2－75　IDT70V09L15のリード動作（$\overline{\text{BUSYL}}$信号アクティブ時）



① IDT70V09L15の$\overline{\text{CE0L}}$アクティブから$\overline{\text{BUSYL}}$確定までの時間：15 ns（MAX.）
② IDT70V09L15の$\overline{\text{BUSYL}}$解除からデータ確定までの時間：15 ns（MAX.）

注　DWC1レジスタの設定により必ず2ウエイト挿入されます。

備考　破線はハイ・インピーダンスを示します。

2. 8. 2　低速デバイスとの接続

V850E/ME2, V850E2/ME3のプログラマブル・ウエイト（アドレス・セットアップ・ウエイト，データ・ウエイト，アイドル・ステート）の設定では，アクセス・タイムが不足する低速デバイス（SRAM，外部ROM，外部I/O）を$\overline{\text{WAIT}}$端子により接続する例（ウエイト制御の回路例）を示します。

【回路構成】

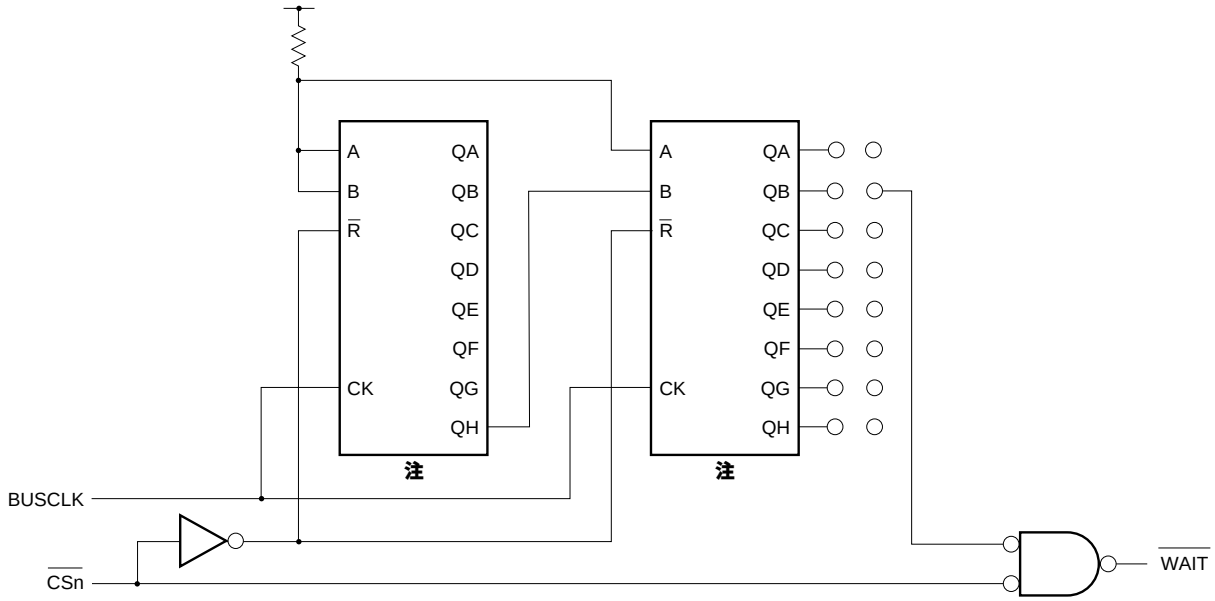
- BUSCLK：任意
- 接続デバイス：プログラマブル・ウエイトの設定でアクセス・タイムが不足する低速ROM
- 使用$\overline{\text{CS}}$信号：$\overline{\text{CSn}}$（n = 0-7）

【接続の考え方と注意点】

- 使用している$\overline{\text{CSn}}$信号がアクティブになった時点でV850E/ME2, V850E2/ME3の$\overline{\text{WAIT}}$信号をアクティブにし，カウンタなどでBUSCLKを必要数だけカウント後，$\overline{\text{WAIT}}$信号をインアクティブにします。

注意　最初の$\overline{\text{WAIT}}$信号サンプリング時にセットアップ時間が確保できないような場合は，アドレス・セットアップ・ウエイトまたはデータ・ウエイトをソフトウエアで挿入してください。

図2－76　ウエイト制御部回路構成



注　74164相当のシフト・レジスタを想定しています。

備考　n = 0-7

2. 9　5 Vデバイスとの接続

5V EPROM（M27C800（512 K×16ビット）を接続する例を示します。この節では接続の考え方と回路例だけを示します。

備考　レジスタの設定の考え方については，2. 3　PROMとの接続と同等です。

【回路構成】

- BUSCLK：任意
- 接続デバイス：M27C800-100×1つ
- 使用$\overline{\text{CS}}$信号：$\overline{\text{CS0}}$

  V850E/ME2では外部メモリ空間の0100000H-017FFFFH（ブロック0）に配列することとします。

  0180000H-01FFFFFHはイメージとなります。

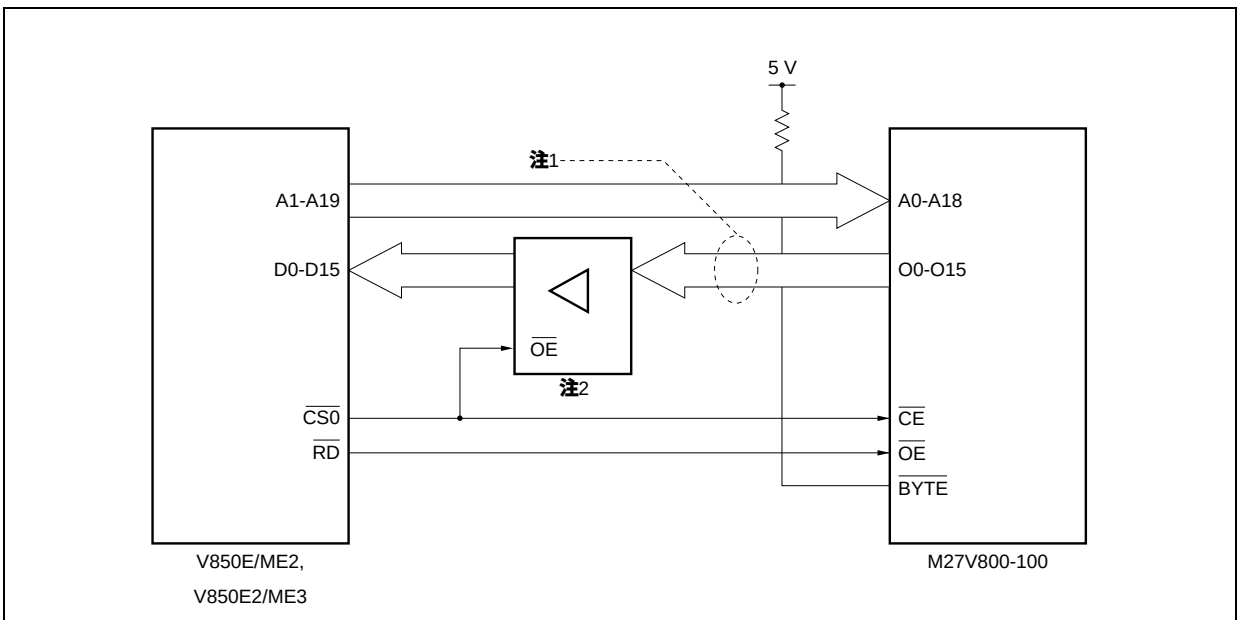
  V850E2/ME3では外部メモリ空間の0000000H-007FFFFH（サブエリア00）に配列することとします。

  0080000H-00FFFFFHはイメージとなります。

【接続の考え方と注意点】

- M27C800-100のO0-O15端子はV850E/ME2, V850E2/ME3の電気的特性より，ハイ・レベル入力電圧（V_{IH}）の最大値「EV_{DD}＋0.3」を守るためにレベル変換デバイスを経由してV850E/ME2, V850E2/ME3のD0-D15端子に接続します。この接続回路例ではレベル変換デバイスとして74LCX16244を使用しています。
- M27C800-100のA0-A18, $\overline{\text{CE}}$, $\overline{\text{OE}}$, $\overline{\text{BYTE}}$端子の接続については，2. 3　PROMとの接続と同等です。

図2－77　M27C800-100接続回路例



注1.　入力のハイ・インピーダンス状態を防ぐ目的のために5 Vでプルアップします。

2.　74LCX16244相当の回路を想定しています。ウエイト時間の算出時は付加回路による遅延時間を考慮してください。

2. 10　DMAフライバイ転送時の接続

SRAMと外部I/O間のDMAフライバイ転送を行う場合の接続回路例を示します。なお，外部I/Oはゲートアレイなどで構成するものとします。

【回路構成】

- DMAフライバイ転送時のメモリ：SRAM
- DMAフライバイ転送時のI/Oデバイス：$\overline{\text{CS7}}$空間に接続された外部I/O
- DMAチャネル番号：チャネル0
- DMAの起動要因：外部トリガ（$\overline{\text{DMARQ0}}$）

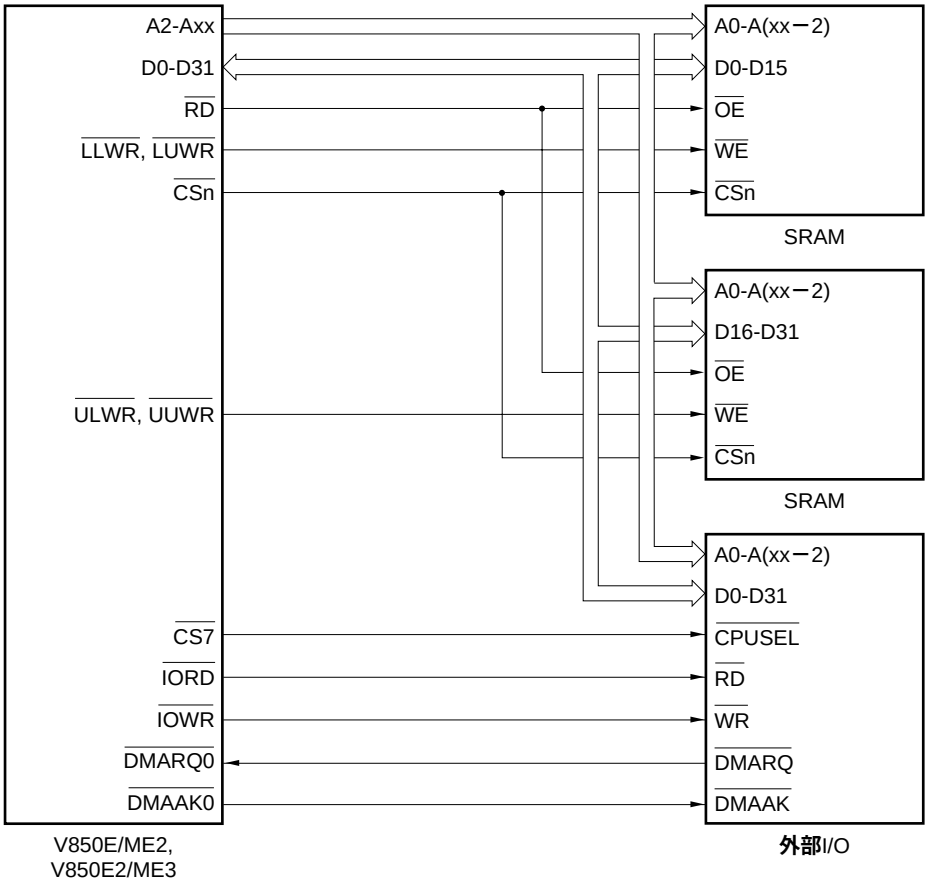
【接続の考え方と注意点】

- DMAフライバイ転送は1回のサイクルで外部メモリ-外部I/O間のデータ転送を行います。V850E/ME2, V850E2/ME3が出力するアドレスは外部メモリのアドレスです。外部I/O側は$\overline{\text{DMAAK0}}$信号で選択されるように構成します。
- 外部I/O内部の各種制御部は，V850E/ME2, V850E2/ME3のCPUが$\overline{\text{CS7}}$信号をアクティブにして，$\overline{\text{IORD}}$信号または$\overline{\text{IOWR}}$信号でアクセスできるように構成します。$\overline{\text{IORD}}$, $\overline{\text{IOWR}}$端子は$\overline{\text{CS2}}$, $\overline{\text{CS5}}$端子と兼用しているため，$\overline{\text{CS2}}$空間と$\overline{\text{CS5}}$空間にはメモリの配列はできなくなります。また，BCPレジスタは08Hを設定します。

  備考　CPUによる制御が必要ない外部I/O（DMAフライバイ転送以外のアクセスを行わない外部I/O）の場合BCPレジスタを00Hで使用することも可能です。

- DMAフライバイ転送時のV850E/ME2, V850E2/ME3の動作は外部メモリ側に設定されたサイクルを発行します。ただし，DWC0, DWC1, PRCレジスタで設定したウエイトは無効となり，DMAフライバイ転送ウエイト・コントロール・レジスタ（FWC）で設定したウエイト数が有効になります。
- この接続回路例ではDMAの起動要因を外部トリガ（$\overline{\text{DMARQ0}}$）としていますが，ソフトウエア・トリガや内蔵周辺I/Oからの割り込みによるトリガを使用する場合，$\overline{\text{DMARQ0}}$信号を使用しない回路構成が可能です。

図2－78　DMAフライバイ転送時の接続回路例



注意　DMA**フライバイ転送の転送方向は次のようになります。**

DMAAK**信号と**RD**信号がアクティブの場合：外部**I/O→SRAM

DMAAK**信号と**WR**信号がアクティブの場合：**SRAM**→外部**I/O

2. 11　システム構成例とCSCnレジスタの設定

2. 11. 1　V850E/ME2の場合

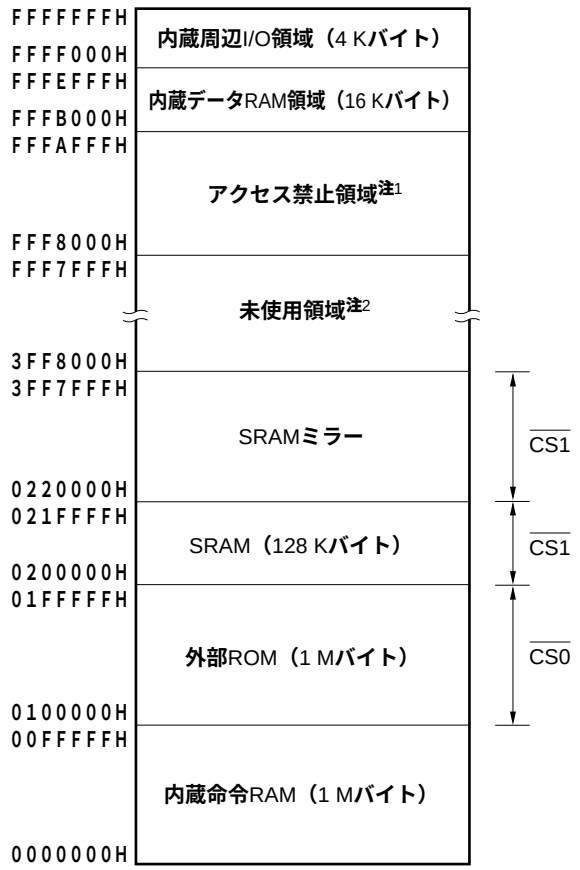
外部メモリ空間に接続するメモリの構成例とCSC0, CSC1レジスタの設定を示します。

V850E/ME2の最初のメモリ・アクセスは0100000Hに対する命令フェッチ・サイクルです。そのため，$\overline{CS0}$空間には外部ROMを配列する必要があります。

（1）外部ROM（1 Mバイト），SRAM（128 Kバイト）を接続した場合の例

- $\overline{CS0}$空間：外部ROM（1 Mバイト）
- $\overline{CS1}$空間：SRAM（128 Kバイト）
- CSC0レジスタの設定値：0001H

図2－79　外部ROM（1 Mバイト），SRAM（128 Kバイト）を接続した場合のメモリ・マップ例



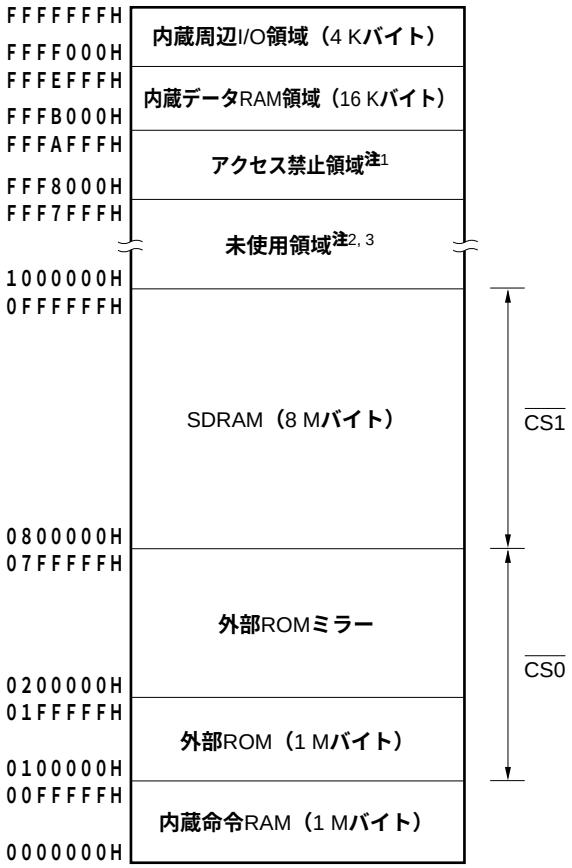
注1.　アクセス禁止領域にアクセスした場合の動作は保証しません。

2.　3FF8000H-3FFFFFFH番地には，FFF8000H-FFFFFFFH番地のイメージが見えます。

(2) 外部ROM（1 Mバイト），SDRAM（8 Mバイト）を接続した場合の例

- $\overline{\text{CS0}}$空間：外部ROM（1 Mバイト）
- $\overline{\text{CS1}}$空間：SDRAM（8 Mバイト）
- CSC0レジスタの設定値：000FH

図2－80　外部ROM（1 Mバイト），SDRAM（8 Mバイト）を接続した場合のメモリ・マップ例



注1. **アクセス禁止領域にアクセスした場合の動作は保証しません。**

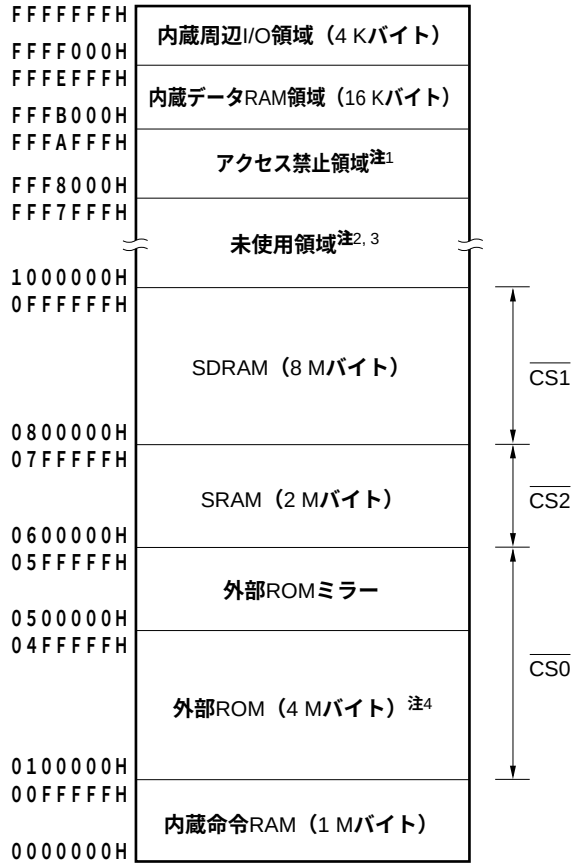
2. 3FF8000H-3FFFFFFFH**番地には，**FFF8000H-FFFFFFFH**番地のイメージが見えます。**

3. 1000000H-3FF7FFFH**番地には，**SDRAM（$\overline{\text{CS1}}$）**空間のイメージが見えます。**

(3) 外部ROM（4 Mバイト），SRAM（2 Mバイト），SDRAM（8 Mバイト）を接続した場合の例

- $\overline{\text{CS0}}$空間：外部ROM（4 Mバイト）
- $\overline{\text{CS2}}$空間：SRAM（2 Mバイト）
- $\overline{\text{CS1}}$空間：SDRAM（8 Mバイト）
- CSC0レジスタの設定値：0807H

図2－81　外部ROM（4 Mバイト），SRAM（2 Mバイト），SDRAM（8 Mバイト）を接続した場合のメモリ・マップ例



注1. アクセス禁止領域にアクセスした場合の動作は保証しません。

2. 3FF8000H-3FFFFFFH番地には，FFF8000H-FFFFFFFH番地のイメージが見えます。

3. 1000000H-3FF7FFFH番地には，SDRAM（$\overline{\text{CS1}}$）空間のイメージが見えます。

4. 32 Mビットの外部ROMで構成した場合の外部ROMのオフセット・アドレスとV850E/ME2のアドレスとの関係を次に示します。

V850E/ME2のアドレス	外部ROMのアドレス
0100000H-01FFFFFH	100000H-1FFFFFH
0200000H-03FFFFFH	200000H-3FFFFFH
0400000H-04FFFFFH	000000H-0FFFFFH
0500000H-05FFFFFH	100000H-1FFFFFH

(4) 外部ROM（4 Mバイト），SDRAM（8 Mバイト），データ空間用SRAM（2 Mバイト），データ空間用SDRAM（128 Mバイト），外部I/O（256 Kバイト）を接続した場合の例

注意　この接続例では，外部I/OをDMAフライバイ転送の対象にすることと仮定し，$\overline{CS2}$, $\overline{CS5}$を使用しないこととします。

(a) プログラム配列可能空間

- $\overline{CS0}$空間：外部ROM（4 Mバイト）
- $\overline{CS1}$空間：SDRAM（8 Mバイト）

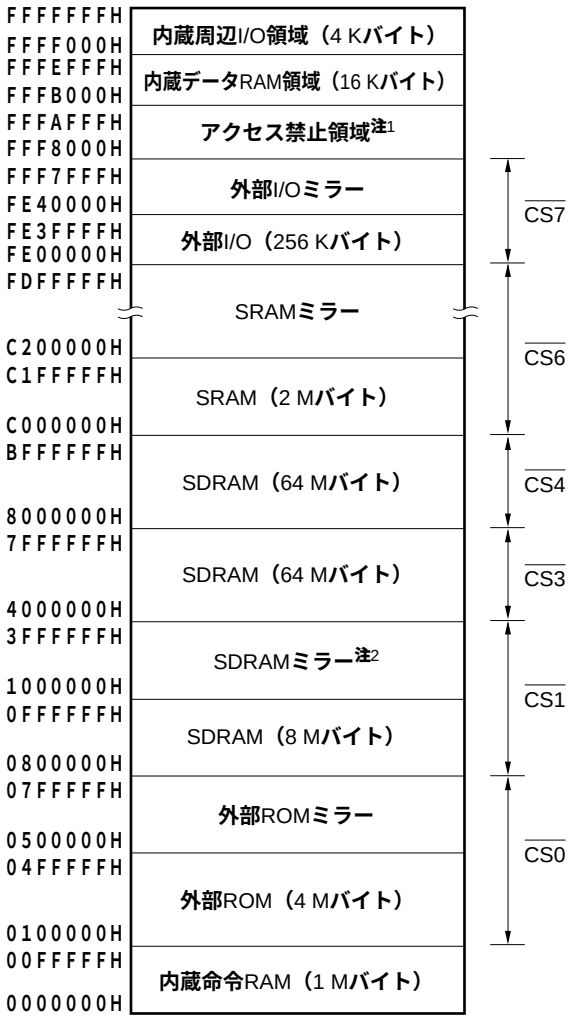
(b) データ空間

- $\overline{CS3}$空間：SDRAM（64 Mバイト）
- $\overline{CS4}$空間：SDRAM（64 Mバイト）
- $\overline{CS6}$空間：SRAM（2 Mバイト）
- $\overline{CS7}$空間：外部I/O（256 Kバイト）

(c) CSC0, CSC1**レジスタの設定**

- CSC0**レジスタの設定値：**000FH
- CSC1**レジスタの設定値：**0001H

図2－82　外部ROM（4 Mバイト），SDRAM（8 Mバイト），データ空間用SRAM（2 Mバイト），データ空間用SDRAM（128 Mバイト），外部I/O（256 Kバイト）を接続した場合のメモリ・マップ例



注1.　**アクセス禁止領域にアクセスした場合の動作は保証しません。**

2.　3FF8000H-3FFFFFFH**番地には，**FFF8000H-FFFFFFFH**番地のイメージが見えます。**

2. 11. 2　V850E2/ME3の場合

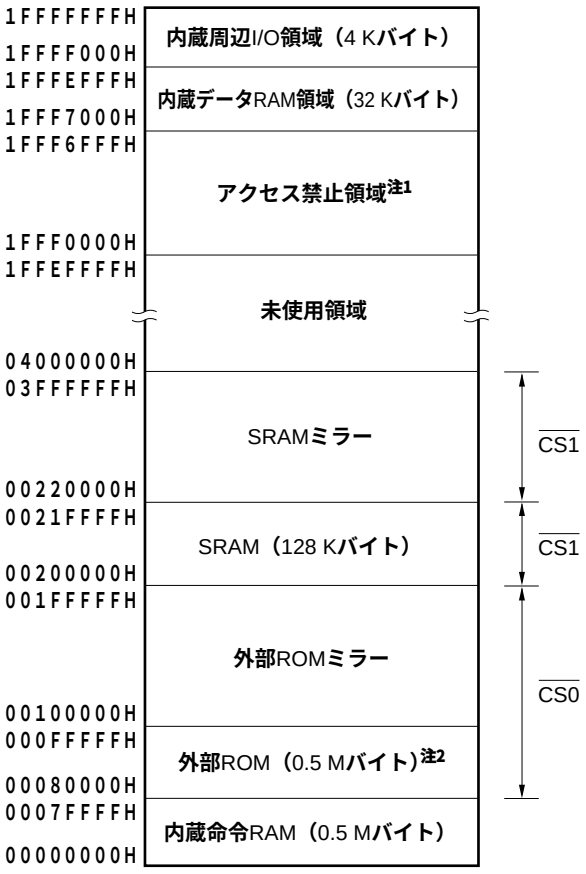
外部メモリ空間に接続するメモリの構成例とCSC0, CSC1レジスタの設定を示します。

V850E2/ME3の最初のメモリ・アクセスは0000000Hに対する命令フェッチ・サイクルです。そのため，$\overline{CS0}$空間には外部ROMを配列する必要があります。

（1）外部ROM（1 Mバイト），SRAM（128 Kバイト）を接続した場合の例

- $\overline{CS0}$空間：外部ROM（1 Mバイト）
- $\overline{CS1}$空間：SRAM（128 Kバイト）
- CSC0レジスタの設定値：0011H

図2－83　外部ROM（1 Mバイト），SRAM（128 Kバイト）を接続した場合のメモリ・マップ例



注1.　アクセス禁止領域にアクセスした場合の動作は保証しません。

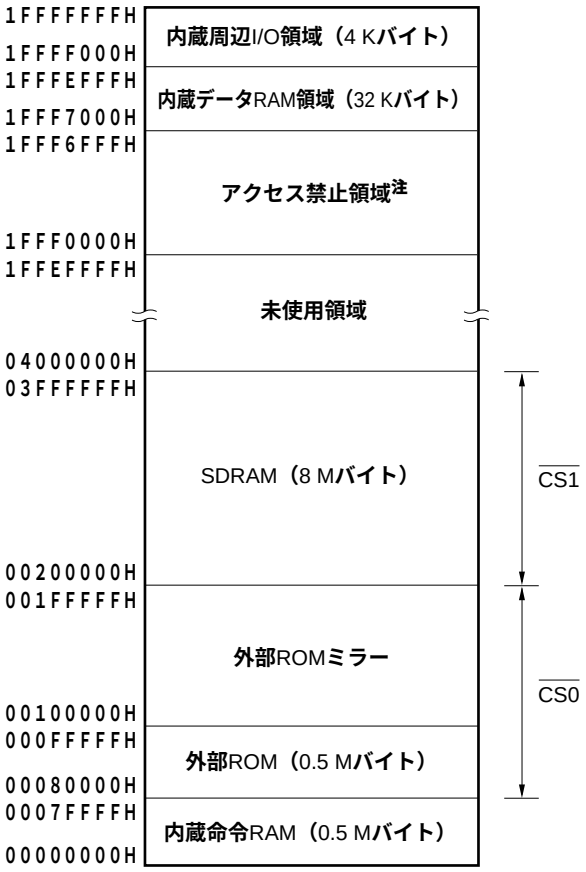
2.　32 Mビットの外部ROMで構成した場合の外部ROMのオフセット・アドレスとV850E2/ME3のアドレスとの関係を次に示します。

V850E2/ME3のアドレス	外部ROMのアドレス
00080000H-000FFFFFH	080000H-0FFFFFH
00100000H-001FFFFFH	000000H-0FFFFFH

(2) 外部ROM（1 Mバイト），SDRAM（8 Mバイト）を接続した場合の例

- $\overline{\text{CS0}}$空間：外部ROM（1 Mバイト）
- $\overline{\text{CS1}}$空間：SDRAM（8 Mバイト）
- CSC0レジスタの設定値：0011H

図2－84　外部ROM（1 Mバイト），SDRAM（8 Mバイト）を接続した場合のメモリ・マップ例

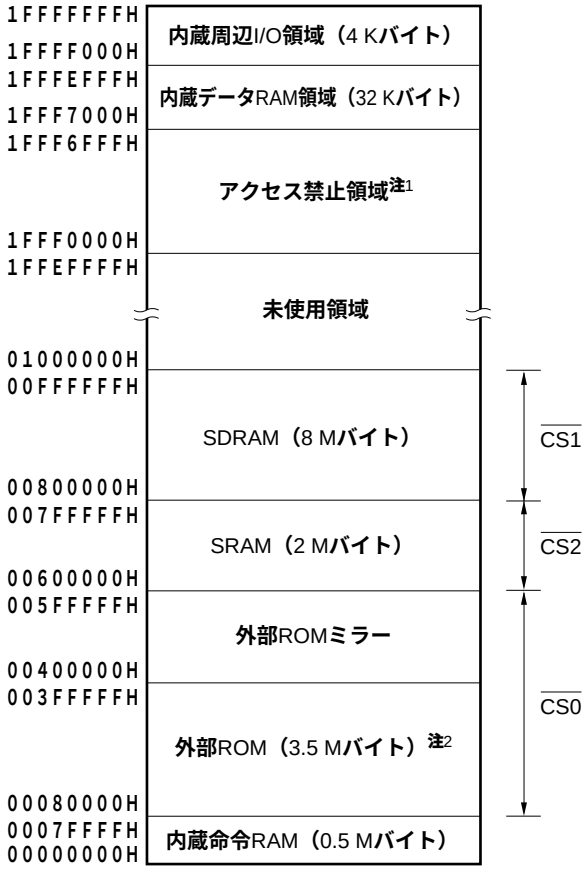


注　アクセス禁止領域にアクセスした場合の動作は保証しません。

(3) 外部ROM（4 Mバイト），SRAM（2 Mバイト），SDRAM（8 Mバイト）を接続した場合の例

- $\overline{\text{CS0}}$空間：外部ROM（4 Mバイト）
- $\overline{\text{CS2}}$空間：SRAM（2 Mバイト）
- $\overline{\text{CS1}}$空間：SDRAM（8 Mバイト）
- CSC0レジスタの設定値：0817H

図2－85　外部ROM（4 Mバイト），SRAM（2 Mバイト），SDRAM（8 Mバイト）を接続した場合のメモリ・マップ例



注1. アクセス禁止領域にアクセスした場合の動作は保証しません。

2. 32 Mビットの外部ROMで構成した場合の外部ROMのオフセット・アドレスとV850E2/ME3のアドレスとの関係を次に示します。

V850E2/ME3のアドレス	外部ROMのアドレス
00080000H-003FFFFFH	080000H-3FFFFFH
00400000H-005FFFFFH	000000H-1FFFFFH

(4) 外部ROM（4 Mバイト），SDRAM（8 Mバイト），データ空間用SRAM（2 Mバイト），データ空間用SDRAM（128 Mバイト），外部I/O（256 Kバイト）を接続した場合の例

注意1.　この接続例では，外部I/OをDMAフライバイ転送の対象にすることと仮定し，$\overline{CS2}$, $\overline{CS5}$を使用しないこととします。

2. V850E2/ME3では，すべての$\overline{CS}$空間をプログラマブル空間として使用できます。

(a) プログラム配列可能空間

- $\overline{CS0}$空間：外部ROM（4 Mバイト）
- $\overline{CS1}$空間：SDRAM（8 Mバイト）

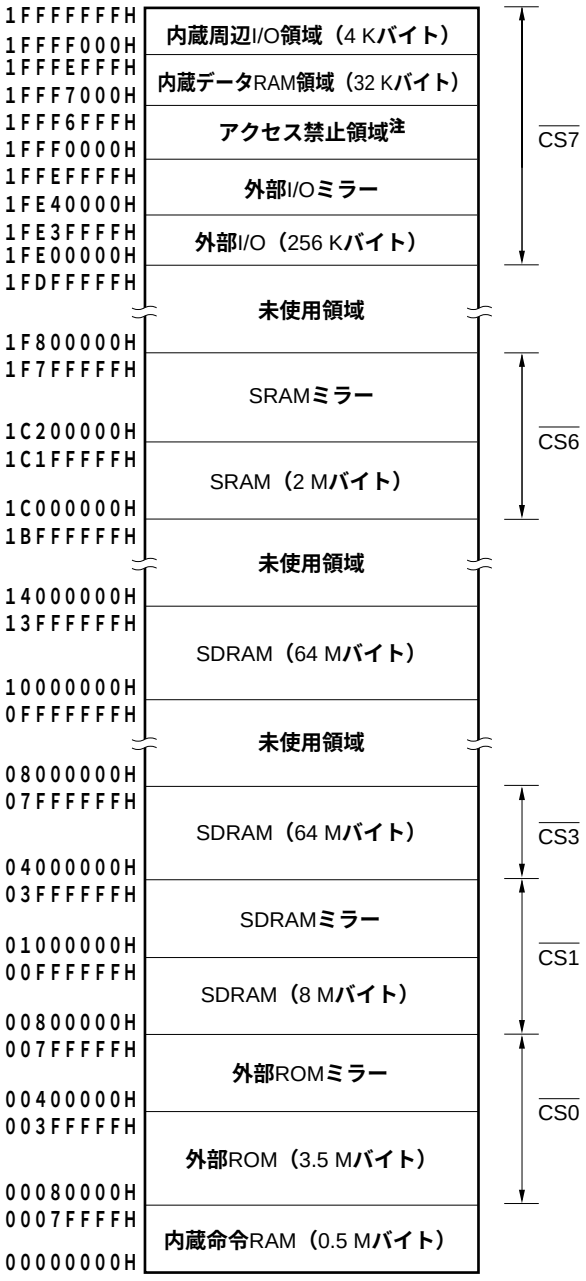
(b) データ空間

- $\overline{CS3}$空間：SDRAM（64 Mバイト）
- $\overline{CS4}$空間：SDRAM（64 Mバイト）
- $\overline{CS6}$空間：SRAM（2 Mバイト）
- $\overline{CS7}$空間：外部I/O（256 Kバイト）

(c) CSC0, CSC1レジスタの設定

- CSC0レジスタの設定値：201FH
- CSC1レジスタの設定値：8011H

図2－86　外部ROM（4 Mバイト），SDRAM（8 Mバイト），データ空間用SRAM（2 Mバイト），データ空間用SDRAM（128 Mバイト），外部I/O（256 Kバイト）を接続した場合のメモリ・マップ例



注　アクセス禁止領域にアクセスした場合の動作は保証しません。

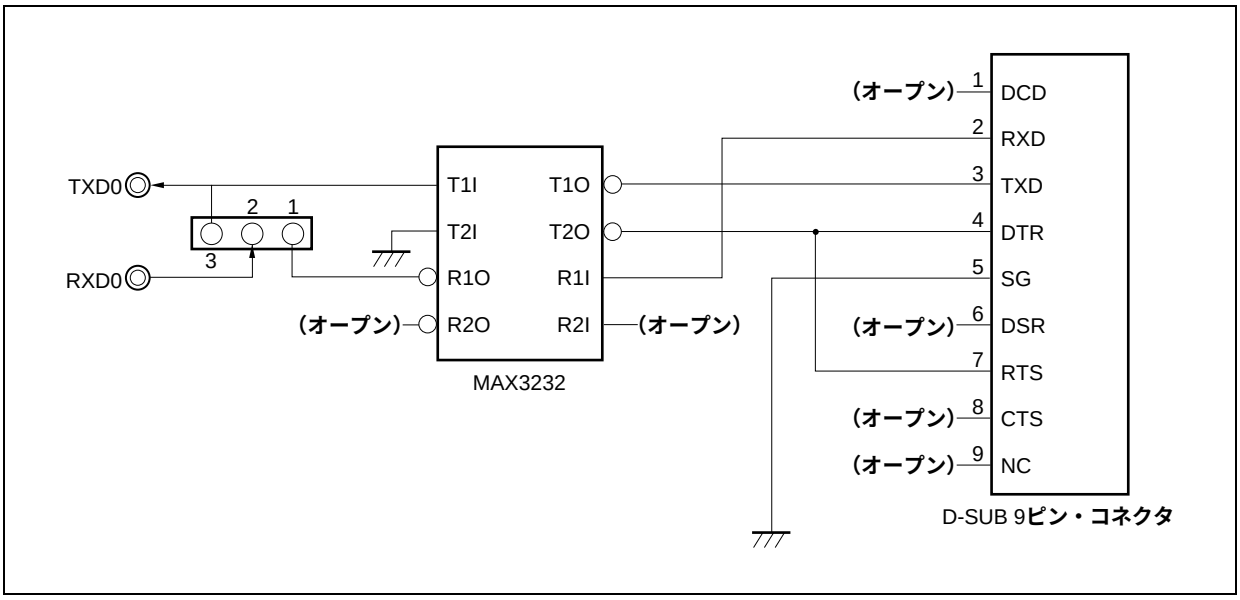
第3章　内蔵周辺機能の接続回路例

★ V850E/ME2，V850E2/ME3の内蔵周辺機能の接続回路例を説明します。接続回路例は第4章で使用しているTB-V850E/ME2の回路（V850E/ME2用）を抜粋したものです。なお，V850E2/ME3を使用した場合も基本的には同じ回路構成となります。

3. 1　シリアル・インタフェース（UARTB）の接続

UARTB0を調歩同期式のRS-232-Cインタフェースとして使用する例を示します。なお，この接続回路例では，ディバグ用としてTXD0信号とRXD0信号をジャンパ・スイッチで折り返すことができるように構成しています。

図3ー1　UARTBの接続回路例

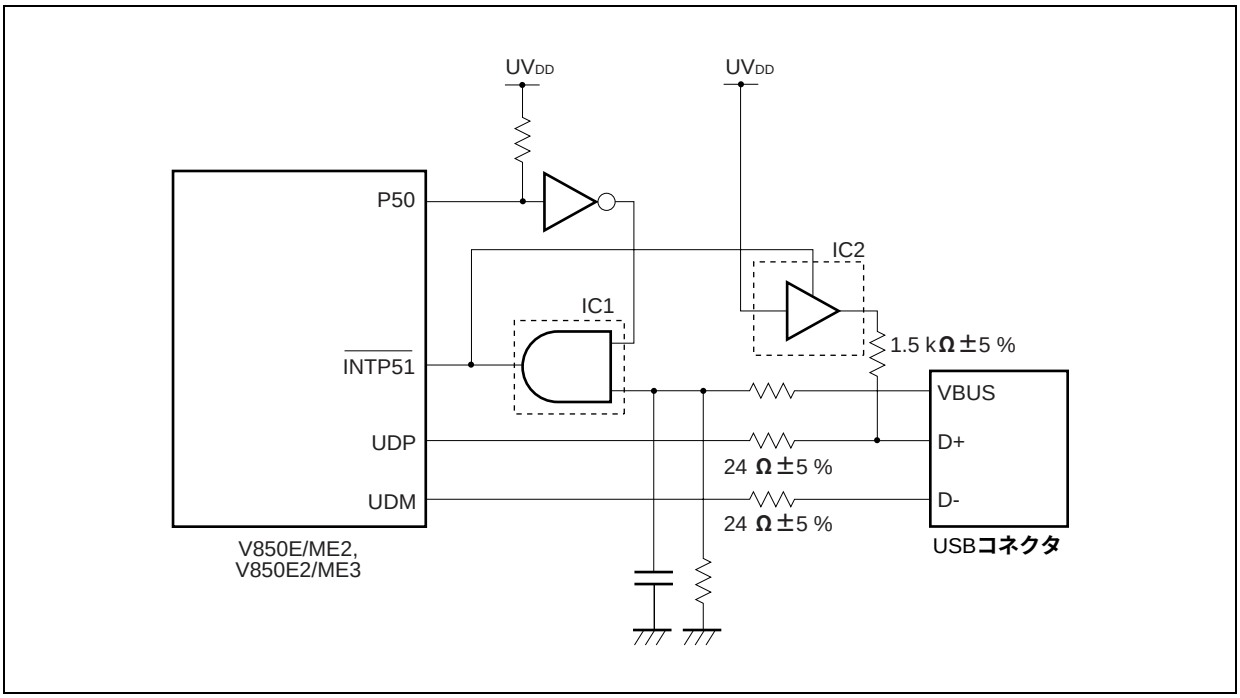


3. 2　USBファンクション・コントローラ（USBF）の接続

USBFにUSBインタフェースを接続する例を示します。

図3ー2　USBFの接続回路例

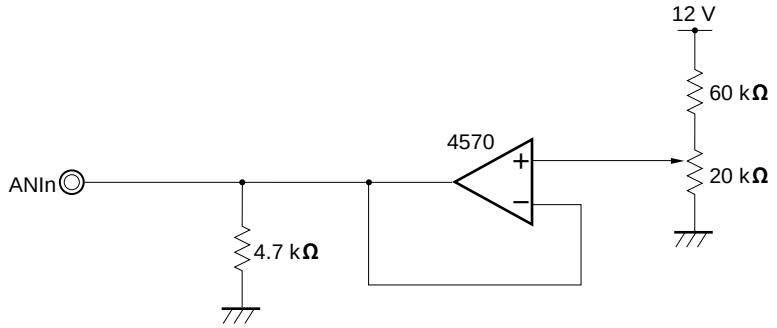
★



3. 3　A/Dコンバータの接続

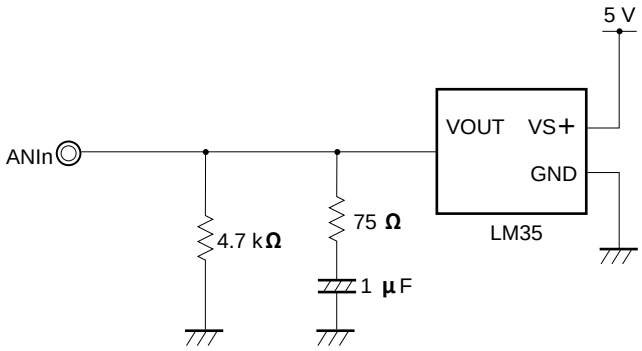
可変抵抗，温度センサを接続する例を示します。

図3－3　可変抵抗による0～3 Vの接続回路例



備考　n = 0-7

図3－4　温度センサの接続回路例

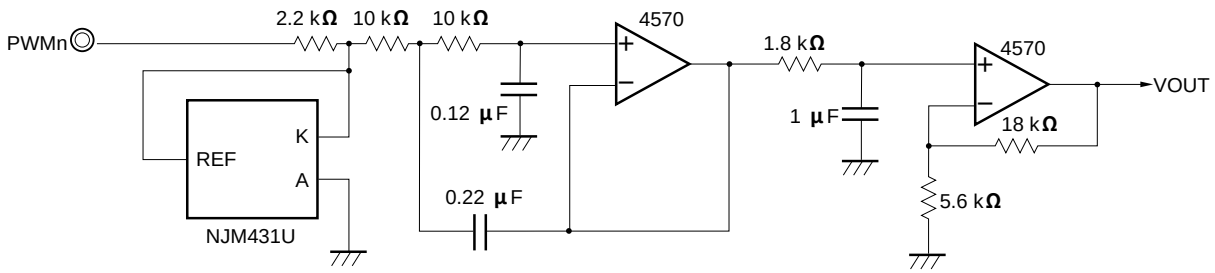


備考　n = 0-7

3. 4　PWMユニットの接続

PWM出力に外部回路を接続し，0～5 Vアナログ出力を構成した例を示します。

図3－5　PWMユニットの接続回路例



注意1.　PWM使用時には，32ビット・モードのデータ・バスは使用できません。

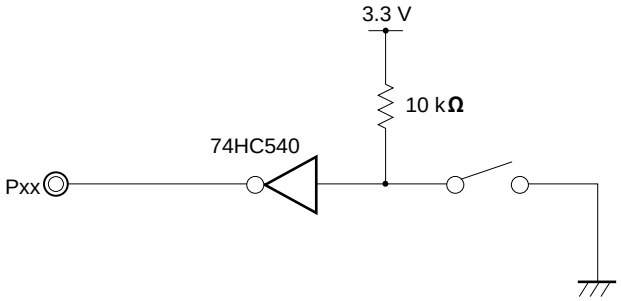
2.　**この接続回路例では，**PWM**の周期を**10 ms**以下で使用しています。**

備考　n = 0, 1

3. 5　ポート機能の接続

ポート機能を接続する例を示します。

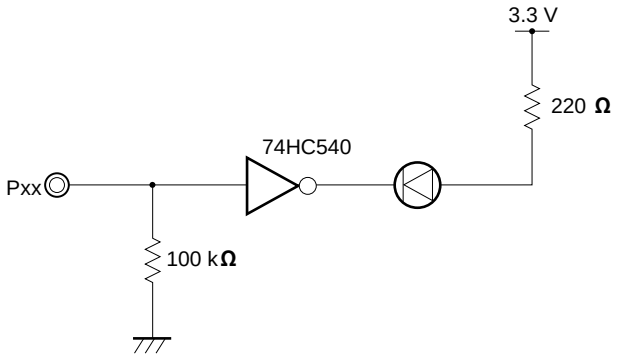
図3－6　入力ポートの接続回路例



注意　この接続回路例では，スイッチのチャタリング除去はしていません。そのため，必要に応じてソフトウエアで除去してください。

備考　Pxx = P10-P13, P20-P25, P50-P55, P65-P67, P72-P77, PAH0-PAH9, PAL0, PAL1, PDH0-PDH15, PCD0-PCD3, PCM0-PCM5, PCS0-PCS7, PCT0-PCT5, PCT7

図3－7　出力ポートの接続回路例



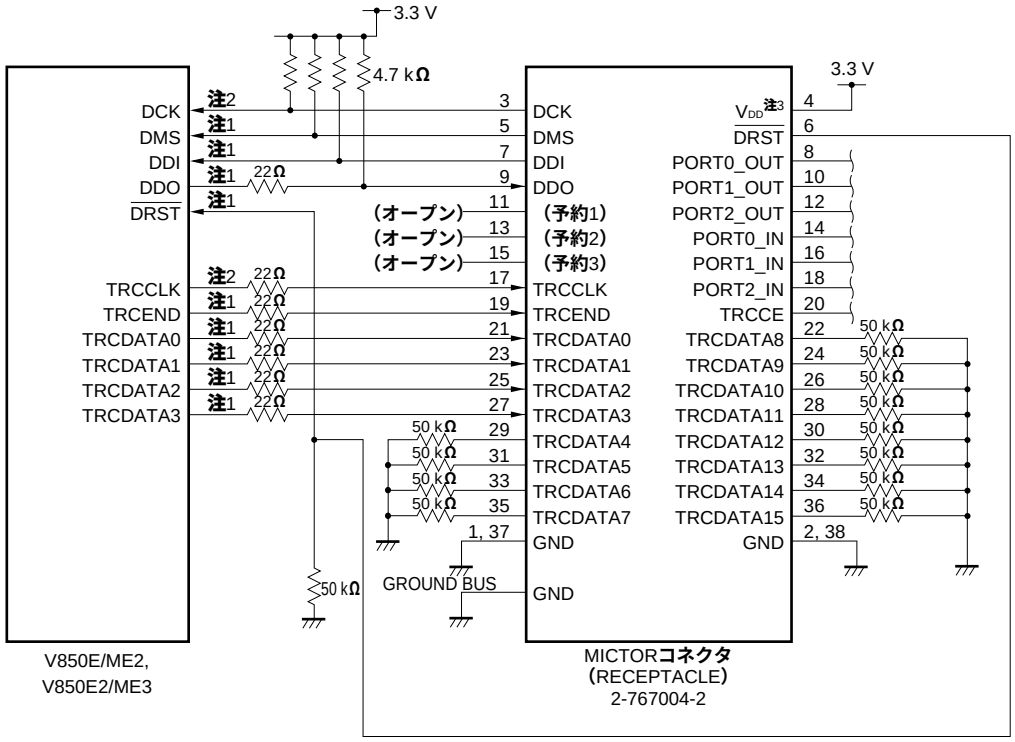
備考　Pxx = P10-P13, P21-P25, P50-P55, P65-P67, P72-P77, PAH0-PAH9, PAL0, PAL1, PDH0-PDH15, PCD0-PCD3, PCM0-PCM5, PCS0-PCS7, PCT0-PCT5, PCT7

3. 6　オンチップ・ディバグ・エミュレータとの接続

開発時に使用するオンチップ・ディバグ・エミュレータを接続する例を示します。

図3－8　オンチップ・ディバグ・エミュレータとの接続回路例

★



注1. **パターン長は**1.97**インチ**（50 mm）**以下にしてください。**

2. DCK, TRCCLK**信号は**GND**で囲んでシールドしてください。**

3. **ターゲット・ボードの電源検出用です。**

注意　この接続回路例では3.3 V**のインタフェースを前提としています。**

第4章　アプリケーション例

V850E/ME2を搭載したCPUボード（TB-V850E/ME2）の機能と回路およびプログラム例を紹介します。

★　この章では，V850E/ME2を使用した場合で記述していますが，V850E2/ME3を使用した場合も基本的には同じです。

4. 1　TB-V850E/ME2の機能

4. 1. 1　概　　要

TB-V850E/ME2の特徴は次のとおりです。

（1）V850E/ME2を搭載

初期設定端子（MODE0, SSEL0, SSEL1, JIT0, JIT1, PLLSEL）の設定はボード上のディップ・スイッチで設定します。なお，MODE1はロウ・レベルに固定です。

（2）メモリ・インタフェースに外部ROM, SRAM, SDRAMを実装

- $\overline{\text{CS0}}$空間：外部ROM（128 Kバイト）を16ビット幅で接続
- $\overline{\text{CS1}}$空間：SDRAM（32 Mバイト）を32ビット幅で接続
- $\overline{\text{CS2}}$空間：SRAM（512 Kバイト）を32ビット幅で接続
- $\overline{\text{CS3}}$空間：SRAM（256 Kバイト）を16ビット幅で接続
- $\overline{\text{CS4}}$空間：SDRAM（16 Mバイト）を16ビット幅で接続
- $\overline{\text{CS5}}$空間：SRAM（128 Kバイト）を8ビット幅で接続
- $\overline{\text{CS6}}$空間：SDRAM（16 Mバイト）を8ビット幅で接続
- $\overline{\text{CS7}}$空間：未使用

注意　$\overline{\text{CS0}}$空間の外部ROMは5 V製品を使用しているため，データ・バスは74LCX16244を経由して接続しています。

（3）RS-232-C（2チャネル）

V850E/ME2のUARTB0とUARTB1に接続

（4）USB

V850E/ME2のUSBファンクション・コントローラ（USBF）に接続

（5）汎用スイッチ入力×8，汎用LED出力×8

V850E/ME2のポート機能に接続

（6）アナログ入力

ANI0：可変抵抗を接続

ANI1：温度センサを接続

（7）1～5 Vアナログ出力

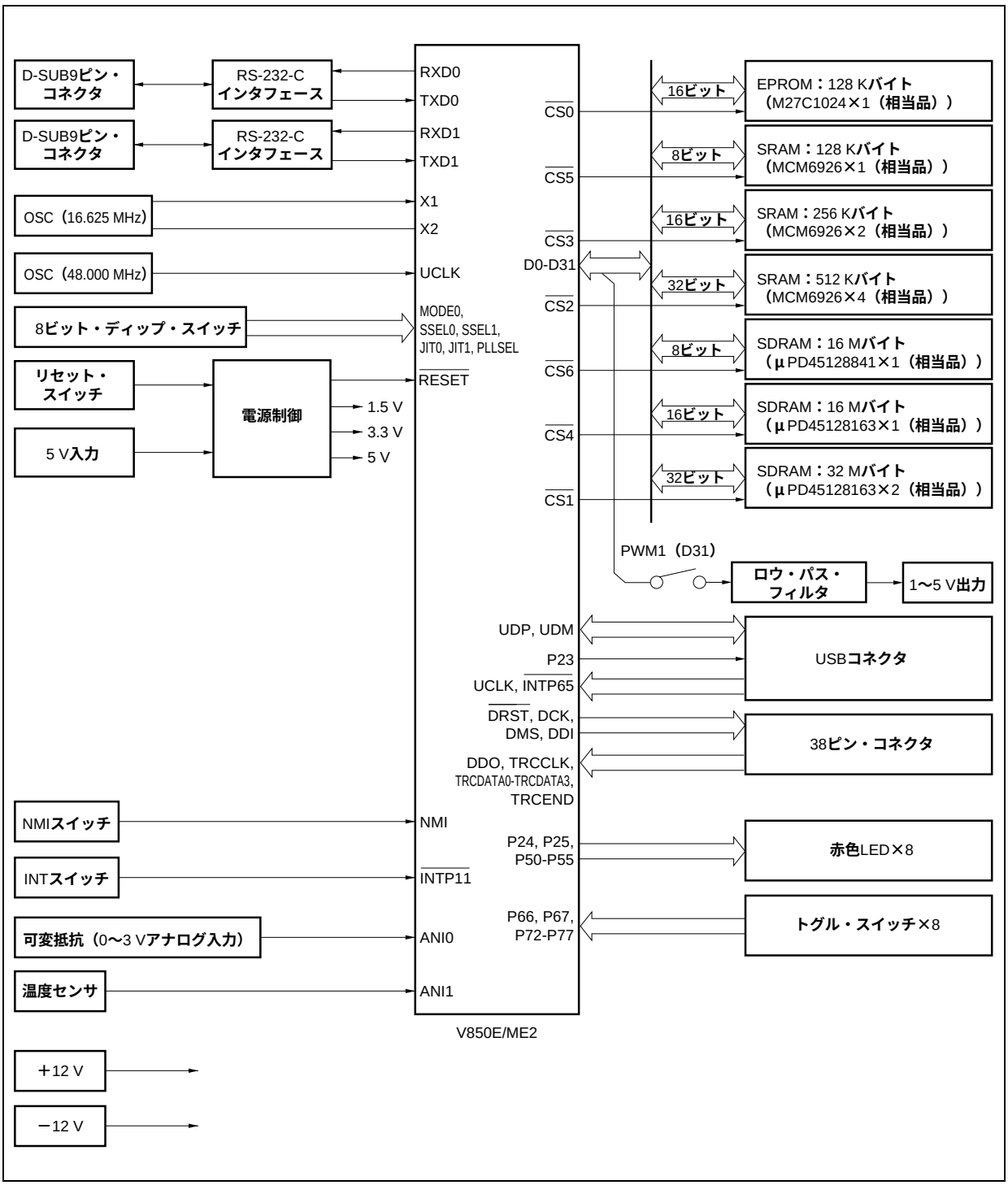
V850E/ME2のPWM1に外部ロウ・パス・フィルタを経由して接続

注意　PWM1はD31と兼用しています。そのため，外部データ・バス幅を32ビットで使用できません。必ず外部データ・バス幅を16ビットで使用してください。

（8）ディバグ・インタフェース

オンチップ・ディバグ・エミュレータ接続用コネクタを実装

図4－1　ボード構成図



4. 1. 2　メモリ・マップ

CSC0レジスタを0E01H，CSC1レジスタを0F00Hに指定した場合のメモリ・マップを示します。

図4－2　メモリ・マップ

XFFF FFFFH XFFF 8000H	内蔵周辺I/O＆内蔵データRAM領域
XFFF 7FFFH XF82 0000H	（$\overline{\text{CS5}}$空間）：この空間はSRAM空間のミラーとなります
XF81 FFFFH XF80 0000H	$\overline{\text{CS5}}$空間：SRAM（128 Kバイト／8ビット幅）
XF7F FFFFH XD00 0000H	（$\overline{\text{CS6}}$空間）：この空間はSDRAM空間のミラーとなります
XCFF FFFFH XC00 0000H	$\overline{\text{CS6}}$空間：SDRAM（16 Mバイト／8ビット幅）
XBFF FFFFH X900 0000H	（$\overline{\text{CS4}}$空間）：この空間はSDRAM空間のミラーとなります
X8FF FFFFH X800 0000H	$\overline{\text{CS4}}$空間：SDRAM（16 Mバイト／16ビット幅）
X7FF FFFFH X404 0000H	（$\overline{\text{CS3}}$空間）：この空間はSRAM空間のミラーとなります
X403 FFFFH X400 0000H	$\overline{\text{CS3}}$空間：SRAM（256 Kバイト／16ビット幅）
X3FF FFFFH X3FF 8000H	この空間は内蔵周辺I/O＆内蔵データRAM空間のミラーとなります
X3FF 7FFFH X280 0000H	（$\overline{\text{CS1}}$空間）：この空間はSDRAM空間のミラーとなります
X27F FFFFH X080 0000H	$\overline{\text{CS1}}$空間：SDRAM（32 Mバイト／32ビット幅）
X07F FFFFH X028 0000H	（$\overline{\text{CS2}}$空間）：この空間はSRAM空間のミラーとなります
X027 FFFFH X020 0000H	$\overline{\text{CS2}}$空間：SRAM（512 Kバイト／32ビット幅）
X01F FFFFH X012 0000H	（$\overline{\text{CS0}}$空間）：この空間はEPROM空間のミラーとなります
X011 FFFFH X010 0000H	$\overline{\text{CS0}}$空間：EPROM（128 Kバイト／16ビット幅）
X00F FFFFH X000 0000H	内蔵命令RAM領域

4. 1. 3　外部バス・インタフェースの接続

外部バス・インタフェースには，外部ROM, SRAM, SDRAMを接続しています。PWM1の機能を使用するときは外部データ・バス幅を16ビットで使用します。

$\overline{\text{CSn}}$信号	接続メモリ	概　　要
$\overline{\text{CS0}}$	EPROM（M27C1024-55×1）	1 MビットEPROM（55 ns）
$\overline{\text{CS1}}$	SDRAM（μPD45128163-A75A×2）	128 MビットSDRAM（133 MHz/16ビット）
$\overline{\text{CS2}}$	SRAM（MCM6926AWJ8×4）	1 Mビット高速SRAM（8 ns）
$\overline{\text{CS3}}$	SRAM（MCM6926AWJ8×2）	1 Mビット高速SRAM（8 ns）
$\overline{\text{CS4}}$	SDRAM（μPD45128163-A75A×1）	128 MビットSDRAM（133 MHz/16ビット）
$\overline{\text{CS5}}$	SRAM（MCM6926AWJ8×1）	1 Mビット高速SRAM（8 ns）
$\overline{\text{CS6}}$	SDRAM（μPD45128841-A75A×1）	128 MビットSDRAM（133 MHz/8ビット）

4.1.4　周辺機能の接続

（1）　ポート機能の接続

8本の出力ポートと8本の入力ポートを使用しています。

端子名称	入出力	機　　能
P23	出力	D+端子のプルアップ制御
P24	出力	汎用LED1（点灯 = 1，消灯 = 0）
P25	出力	汎用LED2（点灯 = 1，消灯 = 0）
P50	出力	汎用LED3（点灯 = 1，消灯 = 0）
P51	出力	汎用LED4（点灯 = 1，消灯 = 0）
P52	出力	汎用LED5（点灯 = 1，消灯 = 0）
P53	出力	汎用LED6（点灯 = 1，消灯 = 0）
P54	出力	汎用LED7（点灯 = 1，消灯 = 0）
P55	出力	汎用LED8（点灯 = 1，消灯 = 0）
P66	入力	汎用トグル・スイッチ1（オン = 0，オフ = 1）
P67	入力	汎用トグル・スイッチ2（オン = 0，オフ = 1）
P72	入力	汎用トグル・スイッチ3（オン = 0，オフ = 1）
P73	入力	汎用トグル・スイッチ4（オン = 0，オフ = 1）
P74	入力	汎用トグル・スイッチ5（オン = 0，オフ = 1）
P75	入力	汎用トグル・スイッチ6（オン = 0，オフ = 1）
P76	入力	汎用トグル・スイッチ7（オン = 0，オフ = 1）
P77	入力	汎用トグル・スイッチ8（オン = 0，オフ = 1）

（2）UARTの接続

UARTBnを調歩同期式RS-232-Cインタフェースとして使用します（n = 0, 1）。信号はTXDn, RXDnの2線式です。RS-232-Cコネクタ（CN2, CN3）上のRTSx, DTRx信号はアクティブ・レベルに固定され，CTSx, DCDx, DSRx信号は接続されません。また，UARTBnはジャンパ・スイッチの設定でTXDn信号をRXDn信号に折り返すことができます。

UART名称	インタフェース	備　　考
UARTB0	RS-232-C	D-SUB9ピン・コネクタ（CN2）
UARTB1	RS-232-C	D-SUB9ピン・コネクタ（CN3）

（3）割り込み信号の接続

NMIスイッチ，INTスイッチがそれぞれNMI, $\overline{\text{INTP11}}$端子に接続されます。

割り込み名称	有効エッジの指定	接続スイッチ
NMI	立ち下がり	NMIスイッチ
$\overline{\text{INTP11}}$	立ち下がり	INTスイッチ

(4) A/Dコンバータの接続

ANI0に0～3 Vの可変抵抗を接続，ANI1に温度センサを接続します。

端子名称	機　　能
ANI0	可変抵抗による0～3 V入力
ANI1	温度センサによる0～3 V入力（低温：0 V，高温：3 V）

(5) PWMの接続

PWM1を外部ロウ・パス・フィルタを経由して1～5 Vアナログ出力に接続します。外部ロウ・パス･フィルタは周期20 ms以下に適合します。1～5 Vアナログ出力機能は外部データ・バス幅を32ビットで使用できません。必ず外部データ・バス幅を16ビットで使用してください。

(6) USBファンクション・コントローラ（USBF）の接続

USBファンクション用コネクタを接続します。なお，P23, $\overline{\text{INTP65}}$端子は次の制御を行います。

端子名称	機　　能
P23	D+端子のプルアップ制御
$\overline{\text{INTP65}}$	USBケーブルの接続／切断の検出

(7) ディバグ・インタフェースの接続

ディバグ用の38ピン・ヘッダ・コネクタをオンチップ・ディバグ・エミュレータに接続します。

4.1.5　設定スイッチ

(1) リセット・スイッチ（RESET**）**

V850E/ME2**の**$\overline{\text{RESET}}$**端子を強制的にアクティブ・レベルにする押しボタン・スイッチです。**

(2) NMI**スイッチ（**NMI**）**

V850E/ME2**にノンマスカブル割り込み（**NMI**）を発生させる押しボタン・スイッチです。押し下げ時に** NMI**端子がロウ・レベルになります。**

(3) INT**スイッチ（**INT**）**

V850E/ME2**にマスカブル割り込みを発生させる押しボタン・スイッチです。押し下げ時に**$\overline{\text{INTP11}}$**端子がロウ・レベルになります。**

(4) 動作モード設定スイッチ（DSW1**）**

V850E/ME2**の動作モードとシステム・クロックの初期値を設定します。**V850E/ME2**の各端子はオンで「**0**」，オフで「**1**」に設定されます。**

スイッチ名称	意　味
DSW1-1	MODE0**端子の設定**
DSW1-2	**未使用**
DSW1-3	JIT0**端子の設定**
DSW1-4	JIT1**端子の設定**
DSW1-5	PLLSEL**端子の設定**
DSW1-6	SSEL0**端子の設定**
DSW1-7	SSEL1**端子の設定**
DSW1-8	**未使用**

(5) RS-232-C**インタフェース折り返しスイッチ（**JP1, JP2**）**

UARTBn**に接続される**RS-232-C**の**TXDn**信号を**RXDn**信号に折り返すジャンパ・スイッチです（**n = 0, 1**）。短絡すると**TXDn**信号が**RXDn**信号に折り返されます。**

(6) PWM**選択スイッチ（**JP3**）**

PWM**機能を有効にするジャンパ・スイッチです。**PWM1**端子を**1～5 V**アナログ出力として使用するとき短絡します。開放時には，アナログ出力は**＋1 V**に固定されます。**

4. 1. 6　周辺I/Oレジスタの設定

TB-V850E/ME2のハードウエア構成よる周辺I/Oレジスタの設定値を示します。

注意　次に示す周辺I/Oレジスタの設定値は，BUSCLKを64 MHz（f_{CLK}/2）**で使用し，メモリ・マップは**4. 1. 2　**メモリ・マップを使用した場合です（f_{CLK}：内部システム・クロック）。また，未使用端子は出力ポートを指定しています。**

（1/2）

レジスタ名称	略号	設定値
システム・ウエイト・コントロール・レジスタ	VSWC	33H
チップ・エリア選択コントロール・レジスタ0	CSC0	0E01H
チップ・エリア選択コントロール・レジスタ1	CSC1	0F00H
バス・サイクル・タイプ・コンフィギュレーション・レジスタ0	BCT0	88B8H
バス・サイクル・タイプ・コンフィギュレーション・レジスタ1	BCT1	0B8BH
ローカル・バス・サイジング・コントロール・レジスタ	LBS	0169H
エンディアン・コンフィギュレーション・レジスタ	BEC	0000H
ライン・バッファ・コントロール・レジスタ0	LBC0	0000H
ライン・バッファ・コントロール・レジスタ1	LBC1	0000H
バス・モード・コントロール・レジスタ	BMC	01H
データ・ウエイト・コントロール・レジスタ0	DWC0	0003H
データ・ウエイト・コントロール・レジスタ1	DWC1	0000H
アドレス・セットアップ・ウエイト・コントロール・レジスタ	ASC	0000H
バス・サイクル・ピリオド・コントロール・レジスタ	BCP	00H
バス・サイクル・コントロール・レジスタ	BCC	0452H
SDRAMコンフィギュレーション・レジスタ1	SCR1	20A5H
SDRAMコンフィギュレーション・レジスタ3	SCR3	0000H
SDRAMコンフィギュレーション・レジスタ4	SCR4	2095H
SDRAMコンフィギュレーション・レジスタ6	SCR6	2086H
SDRAMリフレッシュ・コントロール・レジスタ1	RFS1	801EH
SDRAMリフレッシュ・コントロール・レジスタ3	RFS3	0000H
SDRAMリフレッシュ・コントロール・レジスタ4	RFS4	801EH
SDRAMリフレッシュ・コントロール・レジスタ6	RFS6	801EH
外部割り込み立ち上がりエッジ指定レジスタ1	INTR1	00H
外部割り込み立ち下がりエッジ指定レジスタ1	INTF1	00H
外部割り込み立ち上がりエッジ指定レジスタ2	INTR2	00H
外部割り込み立ち下がりエッジ指定レジスタ2	INTF2	00H
外部割り込み立ち上がりエッジ指定レジスタ6	INTR6	20H
外部割り込み立ち下がりエッジ指定レジスタ6	INTF6	20H
クロック・コントロール・レジスタ	CKC	03H
クロック・ソース選択レジスタ	CKS	01H
SSCGコントロール・レジスタ	SSCGC	00H
ロック・レジスタ	LOCKR	0xH
ポート1モード・コントロール・レジスタ	PMC1	0FH
ポート1ファンクション・コントロール・レジスタ	PFC1	0DH

(2/2)

レジスタ名称	略号	設定値
ポート2モード・レジスタ	PM2	C7H
ポート2モード・コントロール・レジスタ	PMC2	07H
ポート2ファンクション・コントロール・レジスタ	PFC2	06H
ポート5モード・レジスタ	PM5	C0H
ポート5モード・コントロール・レジスタ	PMC5	00H
ポート6モード・レジスタ	PM6	FFH
ポート6モード・コントロール・レジスタ	PMC6	20H
ポート7モード・レジスタ	PM7	FFH
ポート7モード・コントロール・レジスタ	PMC7	00H
ポートALモード・コントロール・レジスタ	PMCAL	0003H
ポートALファンクション・コントロール・レジスタL	PFCALL	03H
ポートAHモード・コントロール・レジスタ	PMCAH	03FFH
ポートDHモード・レジスタ	PMDH[注]	0000H
ポートDHモード・コントロール・レジスタ	PMCDH[注]	8000H
ポートDHファンクション・コントロール・レジスタ	PFCDH[注]	8000H
ポートCSレジスタ	PCS	FFH
ポートCSモード・コントロール・レジスタ	PMCCS	FFH
ポートCSファンクション・コントロール・レジスタ	PFCCS	00H
ポートCTレジスタ	PCT	BFH
ポートCTモード・コントロール・レジスタ	PMCCT	BFH
ポートCTファンクション・コントロール・レジスタ	PFCCT	0FH
ポートCMモード・レジスタ	PMCM	C0H
ポートCMモード・コントロール・レジスタ	PMCCM	00H
ポートCMファンクション・コントロール・レジスタ	PFCCM	00H
ポートCDレジスタ	PCD	0CH
ポートCDモード・コントロール・レジスタ	PMCCD	0FH

注　PMDH, PMCDH, PFCDHレジスタの設定は16ビット・モード（MODE1，MODE0 = 01）時のものです。なお，16ビット・モードは，TB-V850E/ME2でPWM1機能をテストするときに使用するモードです。
ただし，アプリケーション・プログラム例は，16ビット・モードで起動し，32ビット・データ・バスに拡張して使用しているため，PMDH, PMCDH, PFCDHレジスタの設定は無効となり，PWM機能は使用できません。

4.1.7　接続コネクタ

(1) 電源コネクタ (J3)

電源コネクタは，TB-V850E/ME2**に＋5 V電源，±12 V電源を供給します。**

○使用コネクタ

- **基板側　　：**B6P-VH**（日本圧着端子製造株式会社製）**
- **ケーブル側　：ハウジング**　VHR-6N**（日本圧着端子製造株式会社製）**
  ターミナル　BVH-21T-P1.1**（日本圧着端子製造株式会社製）**

ピン番号	機能
1	＋5 V
2	＋5 V
3	GND
4	GND
5	＋12 V
6	－12 V

(2) RS-232-C**のコネクタ (**CN2, CN3**)**

○使用コネクタ

- **基板側　　：**DELC-J9PAF-20L6**（日本航空電子工業株式会社製）**
- **ケーブル側　：**DEM-9S**（日本航空電子工業株式会社製）**

ピン番号	信号名	意　味
1	DCDx	**データ・キャリア検出**
2	RXDx	**受信データ**
3	TXDx	**送信データ**
4	DTRx	**ターミナル・レディ**
5	SGx	**信号グランド**
6	DSRx	**データ・セット・レディ**
7	RTSx	**送信要求**
8	CTSx	**送信可**
9	NCx	**未接続**

(3) USBのコネクタ（J2）

○使用コネクタ

・基板側　　：XM7B-0442（オムロン株式会社製）

・ケーブル側：XM7Z-200AB-FC2（オムロン株式会社製）

Bタイプ側を接続。コネクタ付きケーブルはUSB1.1規格品を使用。

ピン番号	信号名	意　味
1	GND	グランド
2	D+	データ入出力信号の正側
3	D-	データ入出力信号の負側
4	VBUS	給電の監視

(4) 電圧出力（PWM）用のコネクタ（J1）

○使用コネクタ

・基板側　　：B2P-VH（日本圧着端子製造株式会社製）

・ケーブル側：ハウジング　VHR-2N（日本圧着端子製造株式会社製）

ターミナル　BVH-21T-P1.1（日本圧着端子製造株式会社製）

ピン番号	信号名	意　味
1	PWM	PWMによる電圧出力
2	GND	グランド

(5) オンチップ・ディバグ・エミュレータのコネクタ（CN1）

○使用コネクタ

・基板側：2-767004-2（MICTOR社製，販売元：タイコ　エレクトロニクス　アンプ株式会社）

ピン番号	接続信号名	入出力	ピン番号	接続信号名	入出力
1	GND	−	2	GND	−
3	DCK	入力	4	V_{DD}	−
5	DMS	入力	6	$\overline{\text{DRST}}$	入力
7	DDI	入力	8	PORT0_OUT	オープンにしてください
9	DDO	出力	10	PORT1_OUT	オープンにしてください
11	(予約1)	オープンにしてください	12	PORT2_OUT	オープンにしてください
13	(予約2)	オープンにしてください	14	PORT0_IN	オープンにしてください
15	(予約3)	オープンにしてください	16	PORT1_IN	オープンにしてください
17	TRCCLK	出力	18	PORT2_IN	オープンにしてください
19	TRCEND	出力	20	TRCCE	オープンにしてください
21	TRCDATA0	出力	22	TRCDATA8	出力
23	TRCDATA1	出力	24	TRCDATA9	出力
25	TRCDATA2	出力	26	TRCDATA10	出力
27	TRCDATA3	出力	28	TRCDATA11	出力
29	TRCDATA4	出力	30	TRCDATA12	出力
31	TRCDATA5	出力	32	TRCDATA13	出力
33	TRCDATA6	出力	34	TRCDATA14	出力
35	TRCDATA7	出力	36	TRCDATA15	出力
37	GND	−	38	GND	−

4. 1. 8　TB-V850E/ME2の仕様

(1) 仕様一覧

次にTB-V850E/ME2の仕様一覧を示します。

項　　目		仕様概要
V850E/ME2		内部システム・クロック = 128 MHz（16 MHz×8） BUSCLK = 64 MHz（128 MHz/2） 動作モードはスイッチ切り替え
EPROM		容量　：128 Kバイト 使用ROM　：M27C1024×1（相当品）
SRAM	8ビット・バス	容量　：128 Kバイト 使用SRAM　：MCM6926×1（相当品）
	16ビット・バス	容量　：256 Kバイト 使用SRAM　：MCM6926×2（相当品）
	32ビット・バス	容量　：512 Kバイト 使用SRAM　：MCM6926×4（相当品）
SDRAM	8ビット・バス	容量　：16 Mバイト 使用SDRAM　：μPD45128841×1（相当品）
	16ビット・バス	容量　：16 Mバイト 使用SDRAM　：μPD45128163×1（相当品）
	32ビット・バス	容量　：32 Mバイト 使用SDRAM　：μPD45128163×2（相当品）
スイッチ		トグル・スイッチ　：8個 押ボタン・スイッチ　：3個 リセット・スイッチ NMIスイッチ INTスイッチ ディップ・スイッチ　：1個（V850E/ME2のモード設定）
LED		発光ダイオード　：13個 汎用（赤）×8 電源用（緑）×5
アナログ入力		ボリュームによる電圧レベルをANI0に入力 電圧レベル　：0～3 V
温度センサ		温度センサによる電圧レベルをANI1に入力 温度範囲　：－55～＋150 ℃
RS-232-Cインタフェース		V850E/ME2内蔵UARTB0, UARTB1を使用（2チャネル）
PWM		PWM出力を電圧（リニア）に変換 電圧レベル　：1～5 V
USB		V850E/ME2内蔵USBFを使用（コネクタはBタイプ）
オンチップ・ディバグ・エミュレータ		V850E/ME2内蔵DCUを使用（コネクタは38ピン）
電源		・入力　：±12 V ＋5 V ・出力　：＋3.3 V（LT1764で＋5 Vより生成） ＋1.5 V（MIC5209で＋3.3 Vより生成）

(2) 回路図

TB-V850E/ME2の回路図を次に示します。

① CPU（1/2）とリセット回路（図4－3参照）

主にV850E/ME2，リセット回路で構成されています。

② CPU（2/2）と周辺回路（図4－4参照）

主にV850E/ME2，発振器，割り込み（NMI, INT）で構成されています。

③ メモリ周辺回路（図4－5，図4－6参照）

主にEPROM，ROM用バッファ，SRAMで構成されています。

④ メモリ周辺回路（図4－7，図4－8参照）

主にSDRAMで構成されています。

⑤ アナログ，シリアル・インタフェース回路（図4－9参照）

主に4570, LM35, MAX3232で構成されています。

⑥ PWM，入出力回路（図4－10参照）

主にNJM431, 4570, 74HC540, LED, トグル・スイッチで構成されています。

⑦ USB周辺回路（図4－11参照）

主にUSB（Bタイプ・コネクタ）で構成されています。

⑧ オンチップ・ディバグ・エミュレータ（図4－12参照）

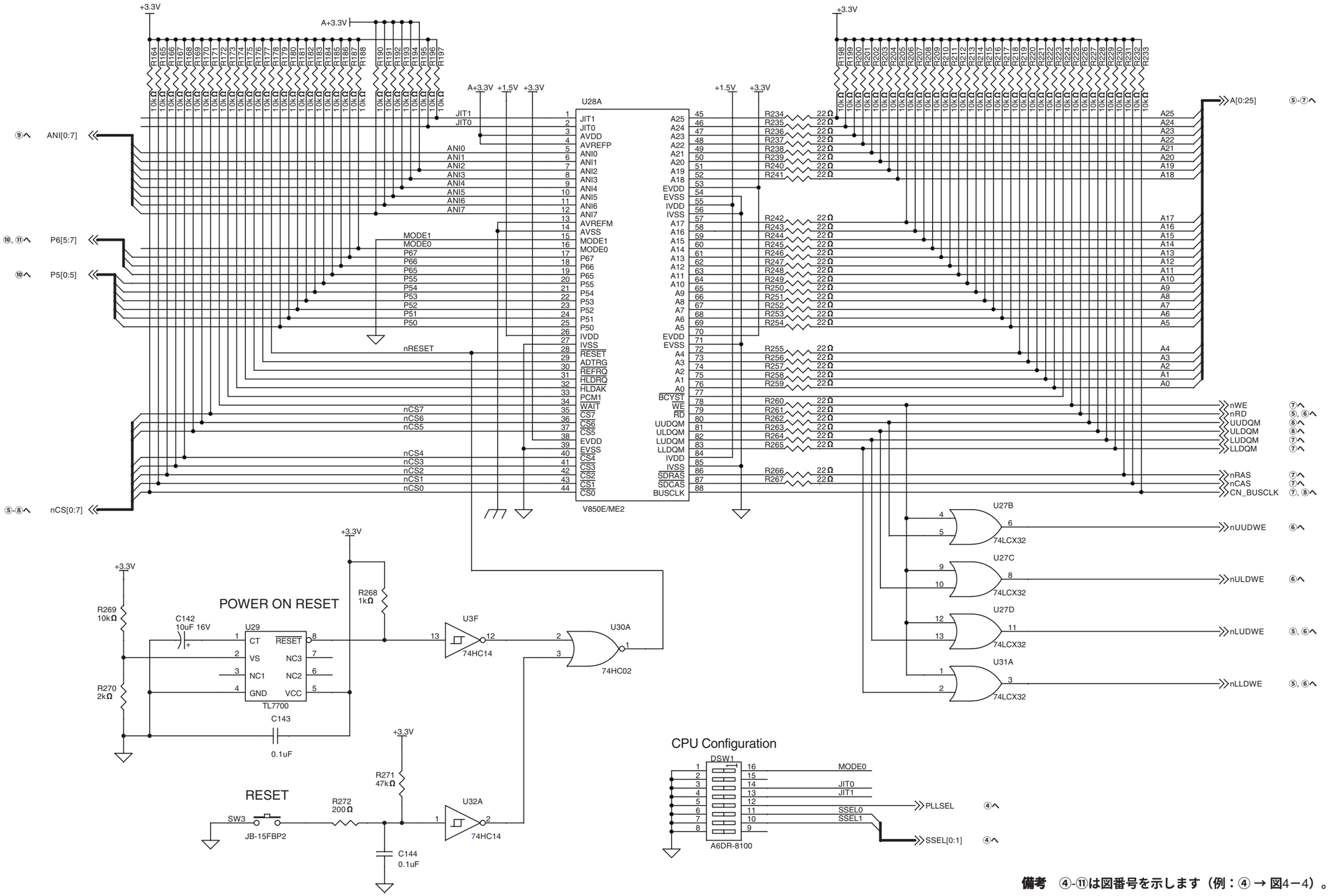
主にオンチップ・ディバグ・エミュレータ接続用コネクタで構成されています。

⑨ 電源周辺回路（図4－13参照）

主に給電コネクタ，LT1764, MIC5209, LEDで構成されています。

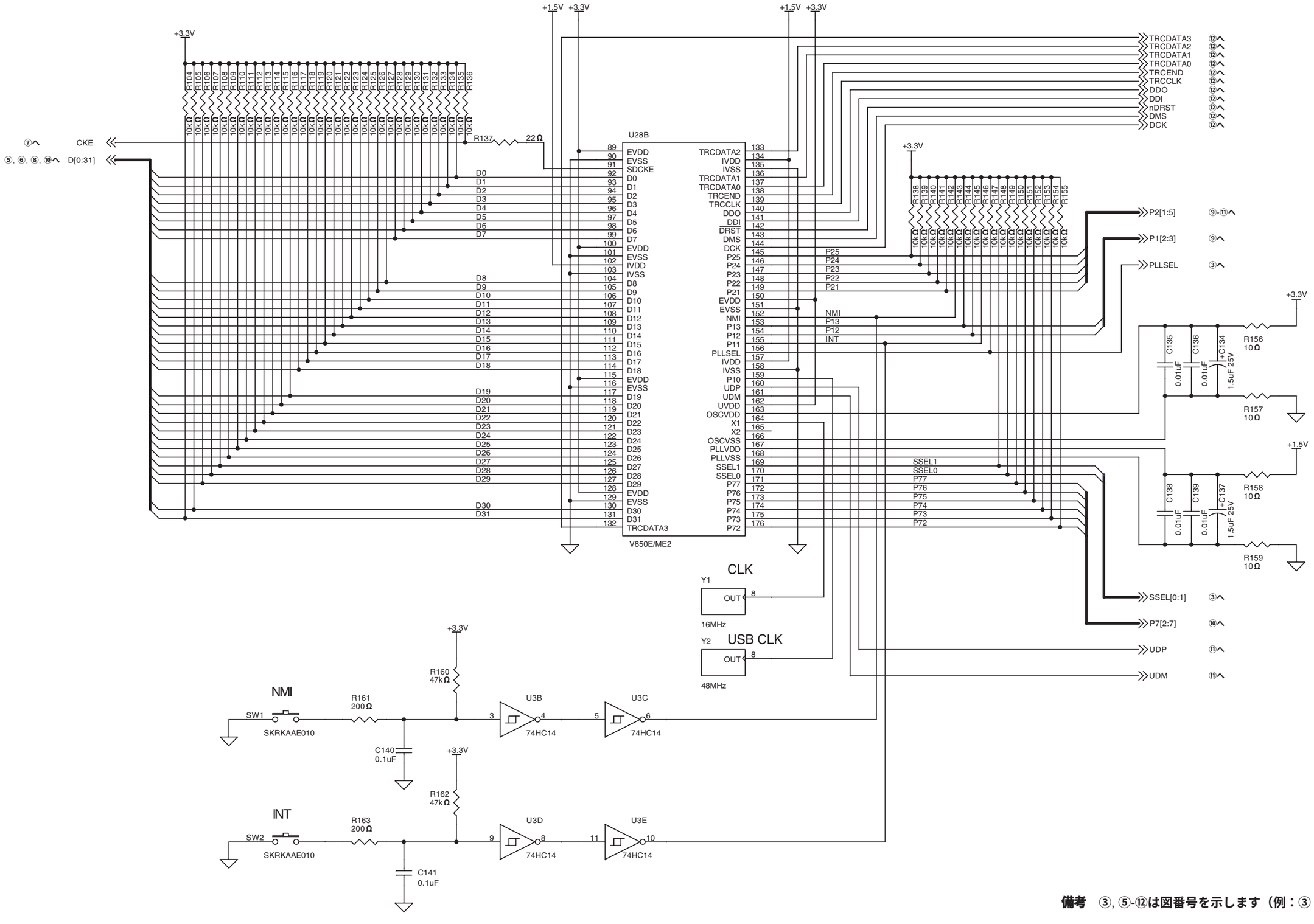
[メ　　モ]

図4－3　CPU（1/2）とリセット回路



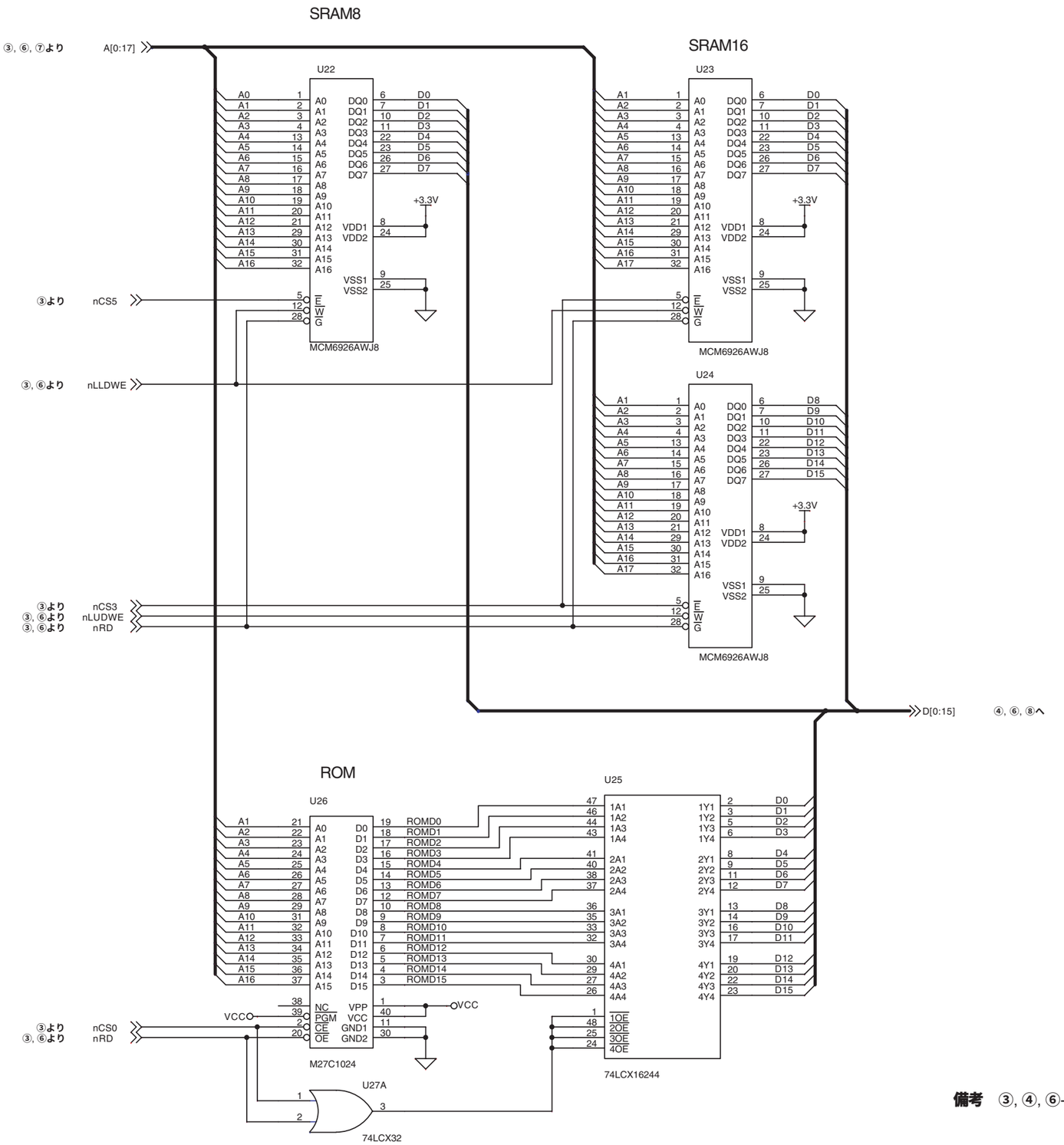
備考　④-⑪は図番号を示します（例：④ → 図4－4）。

図4－4　CPU（2/2）と周辺回路



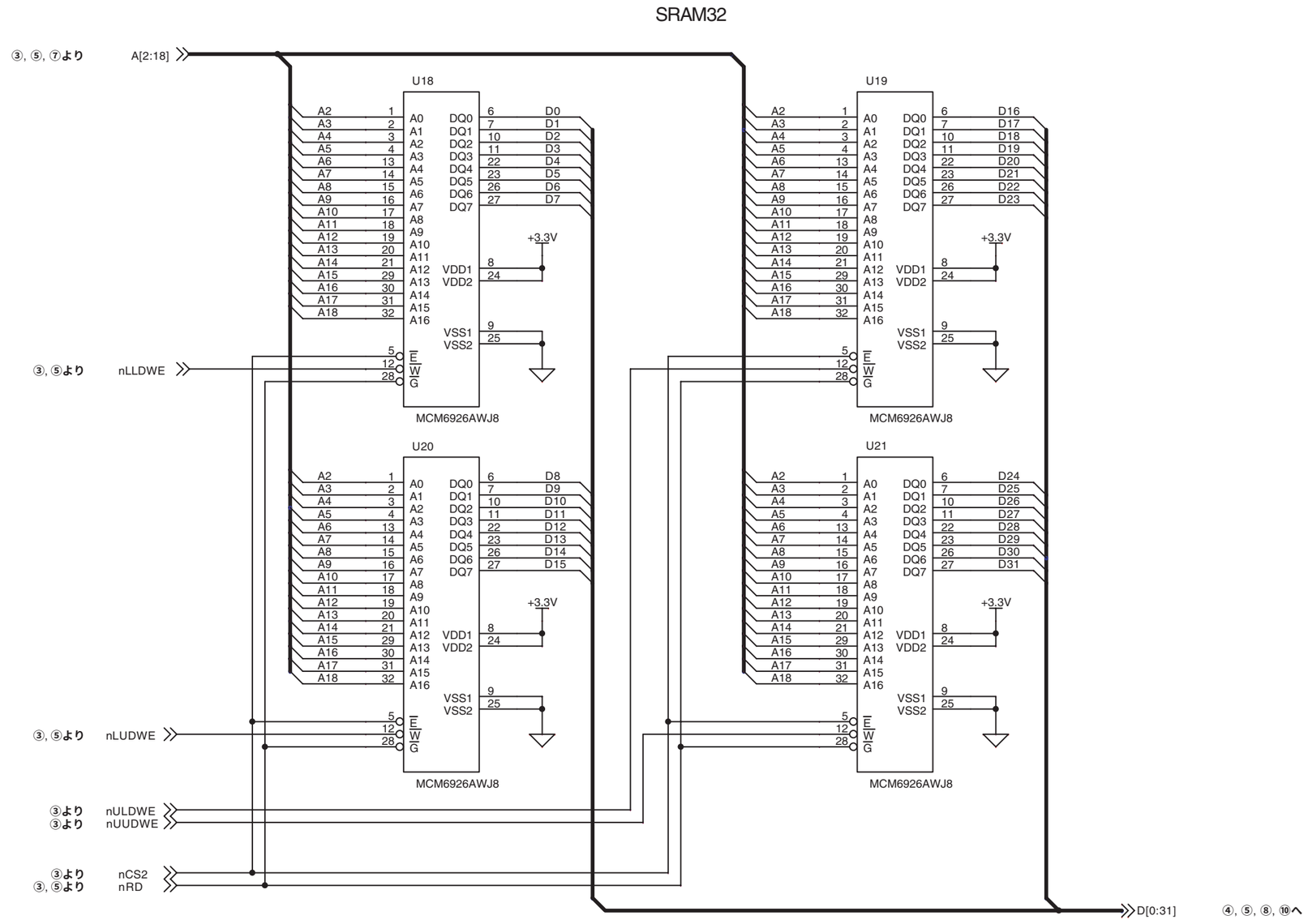
備考　③, ⑤-⑫は図番号を示します（例：③ → 図4－3）。

図4－5　メモリ周辺回路（1）



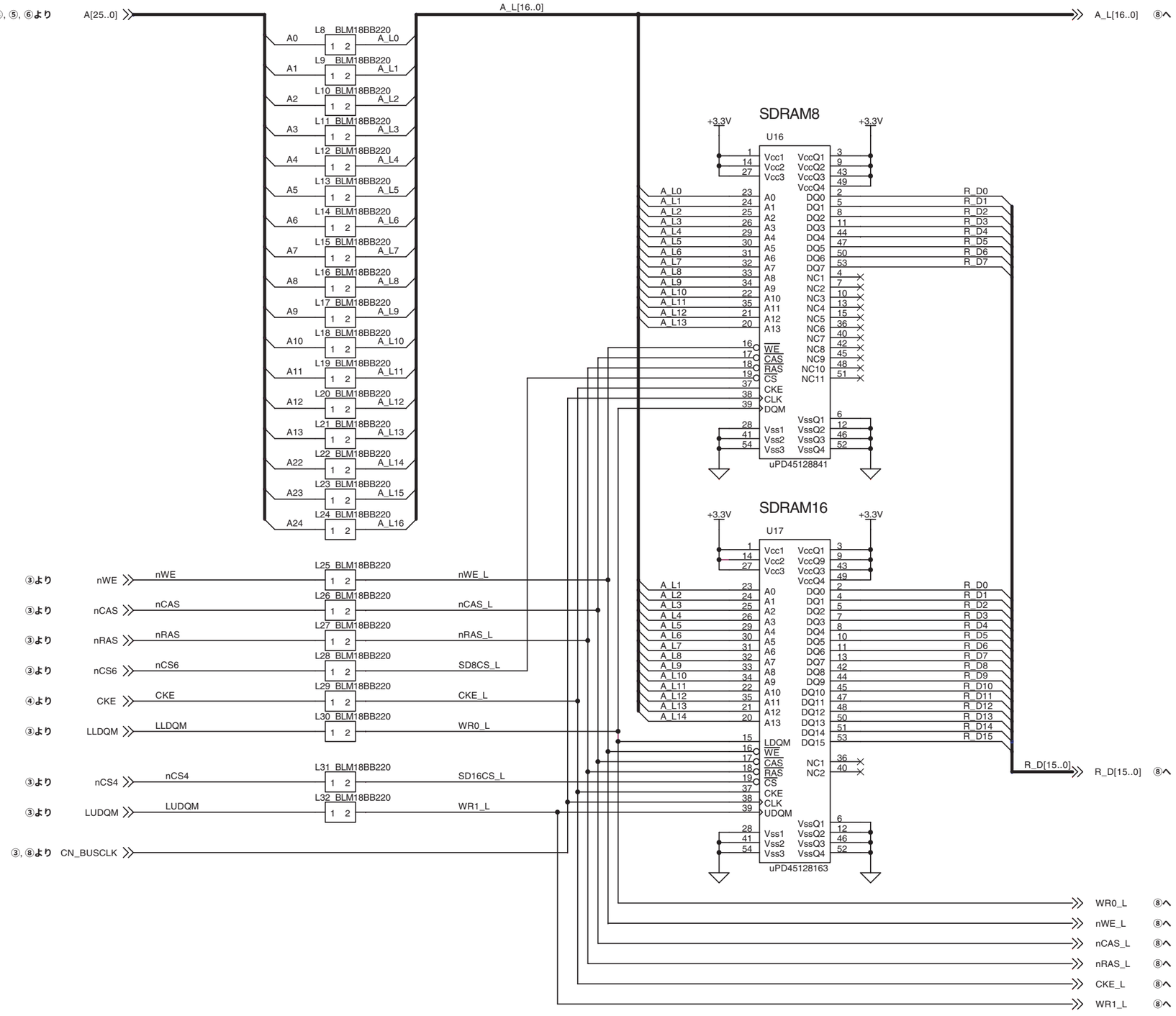
備考　③, ④, ⑥-⑧は図番号を示します（例：③ → 図4－3）。

図4－6　メモリ周辺回路（2）



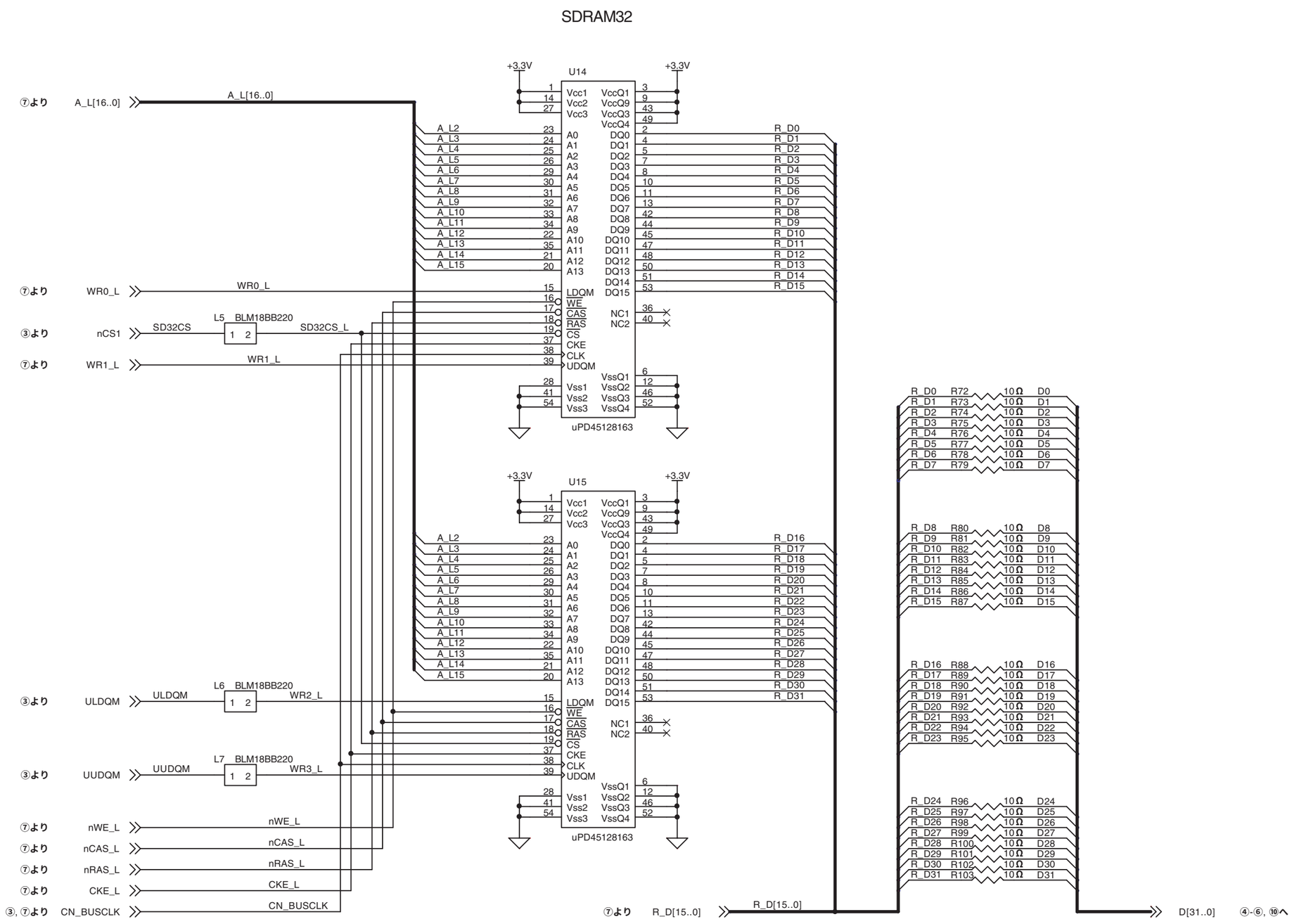
備考　③-⑤，⑦，⑧，⑩は図番号を示します（例：③ → 図4－3）。

図4－7　メモリ周辺回路（3）



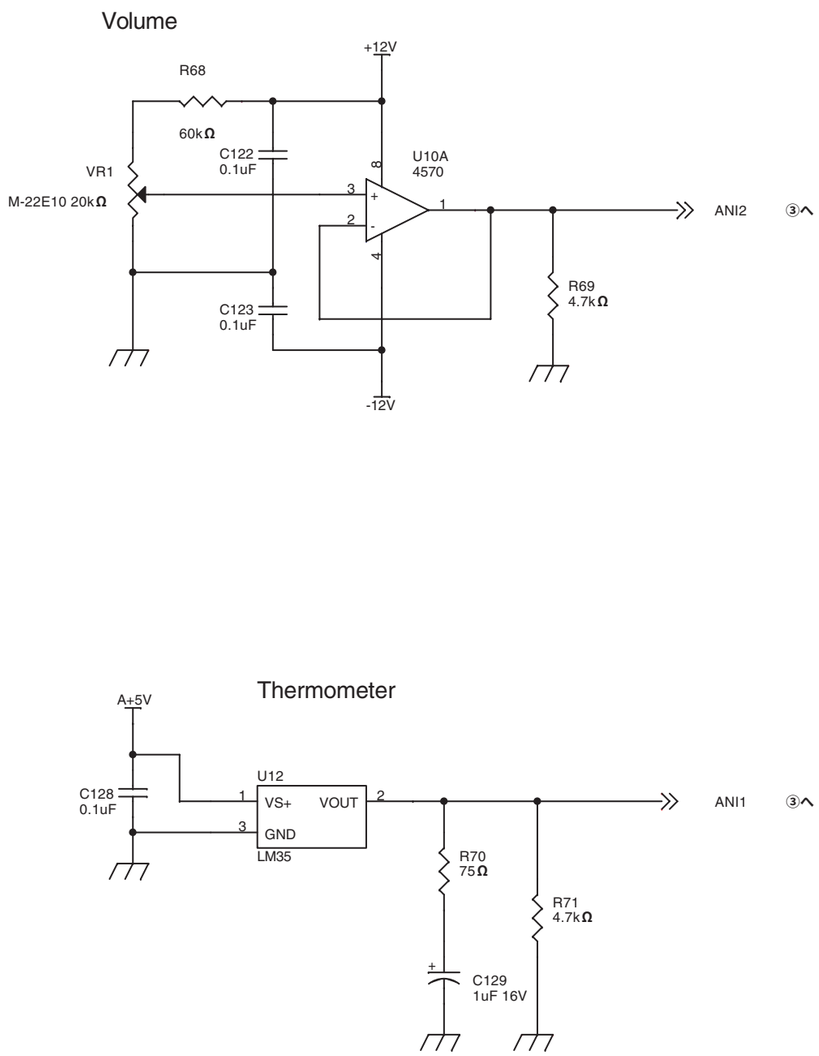
備考　③-⑥，⑧は図番号を示します（例：③ → 図4－3）。

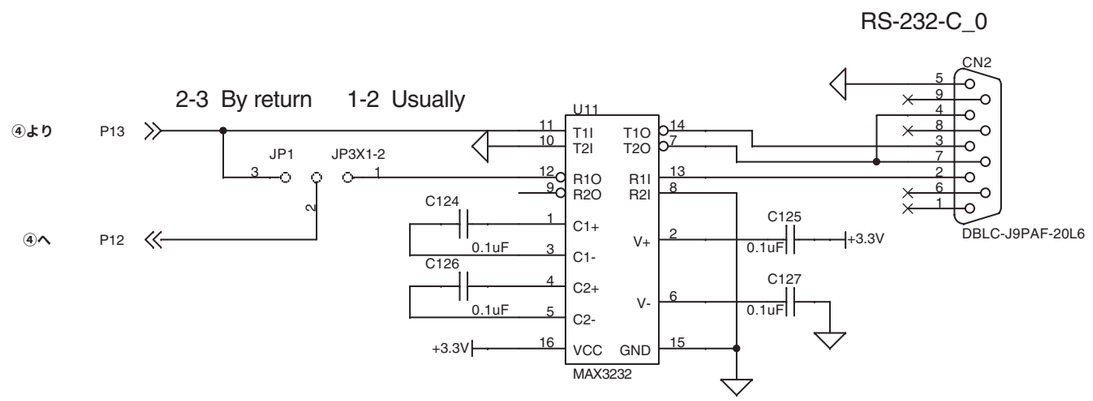
図4－8　メモリ周辺回路（4）

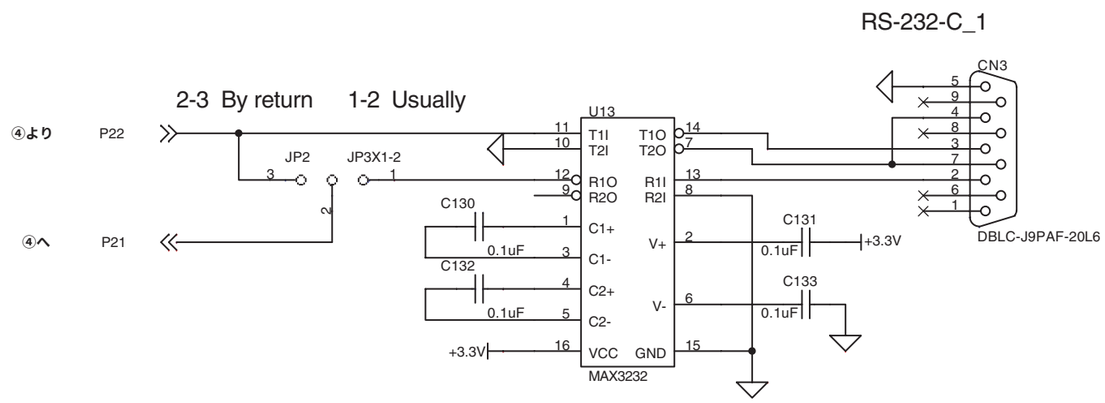


備考　③-⑦，⑩は図番号を示します（例：③ → 図4－3）。

図4－9　アナログ，シリアル・インタフェース回路

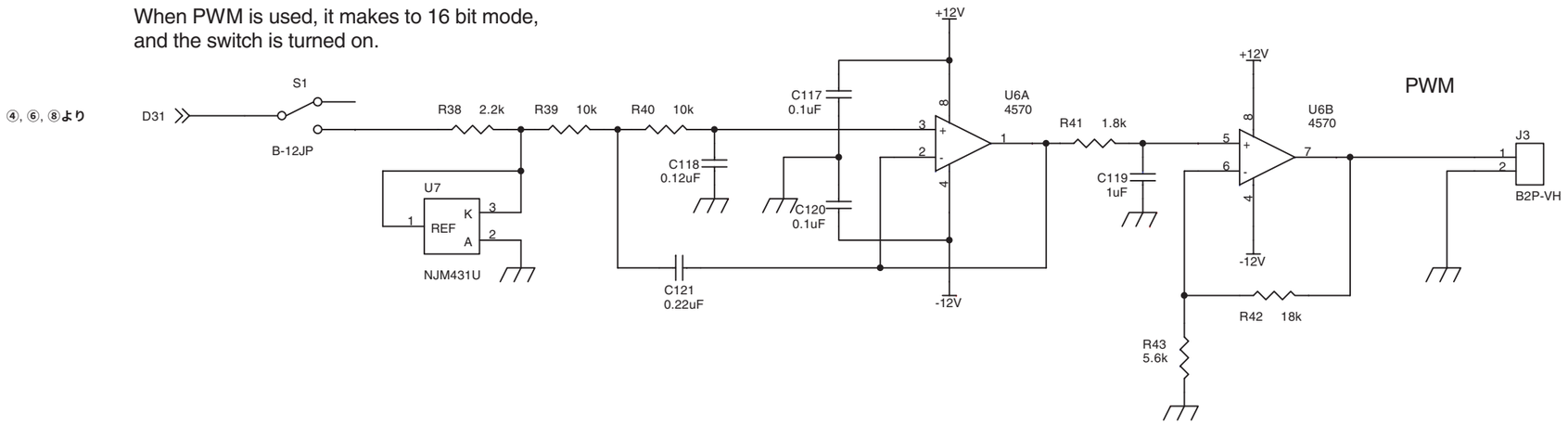


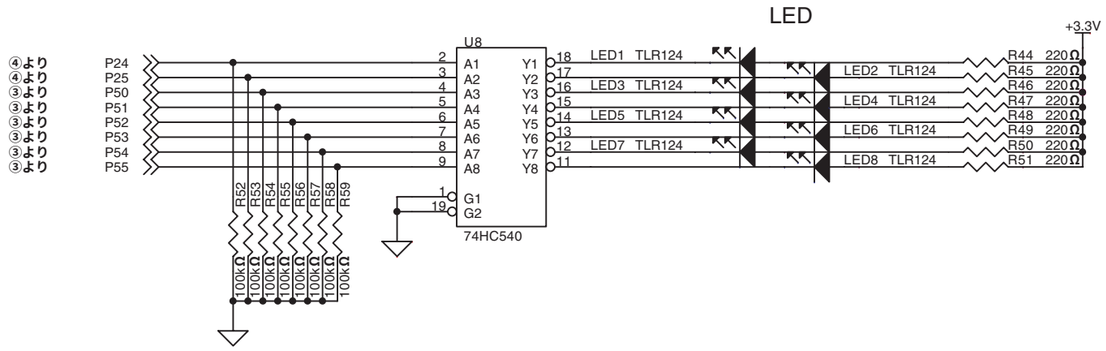


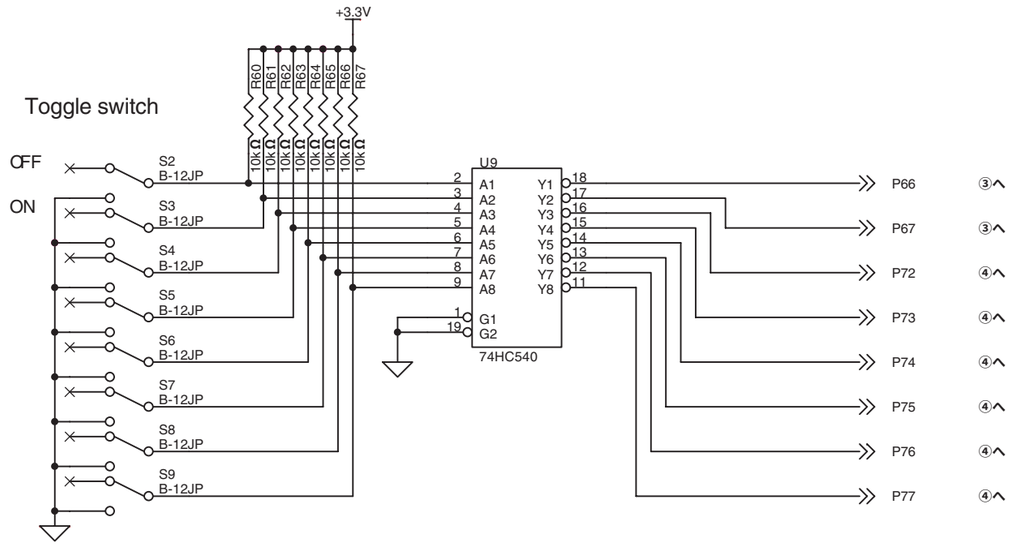


備考　③，④は図番号を示します（例：③ → 図4－3）。

図4－10　PWM，入出力回路

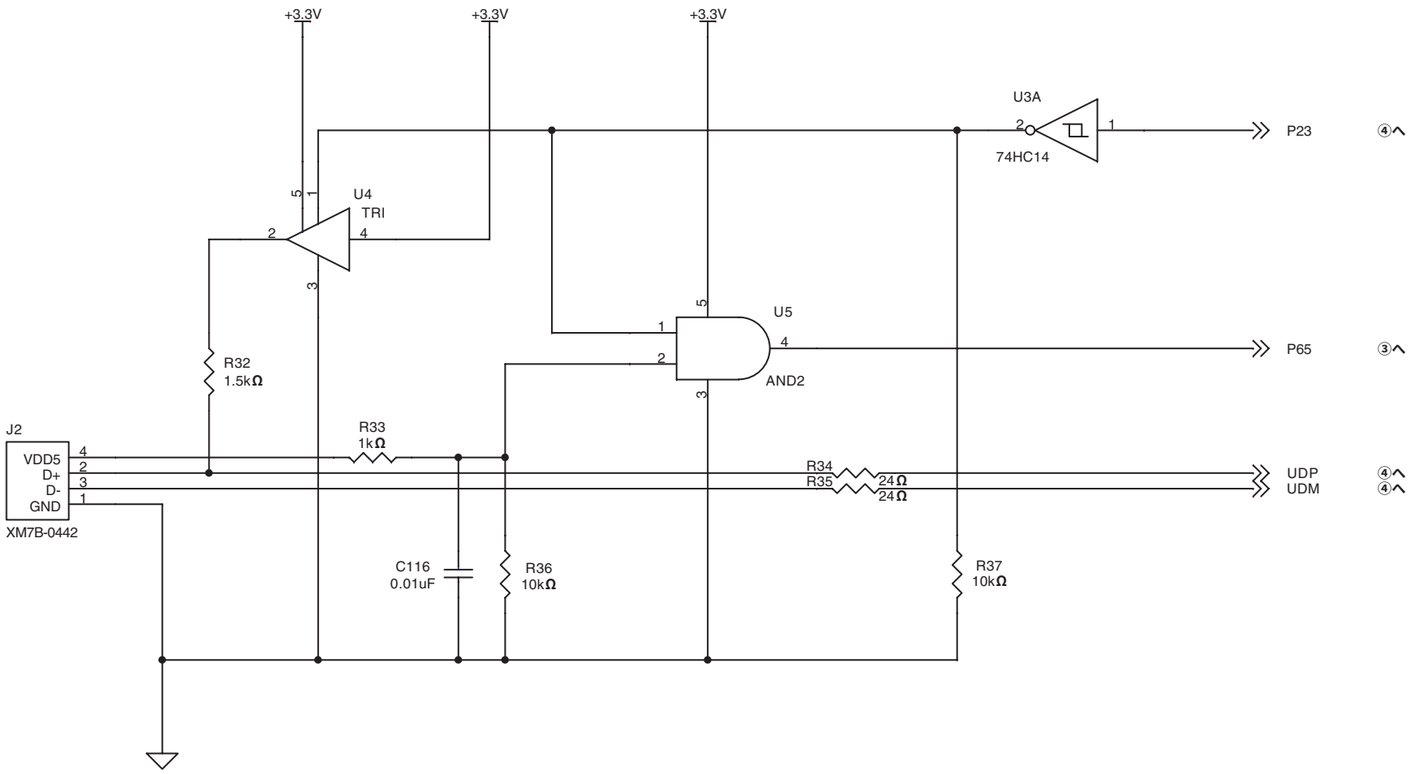






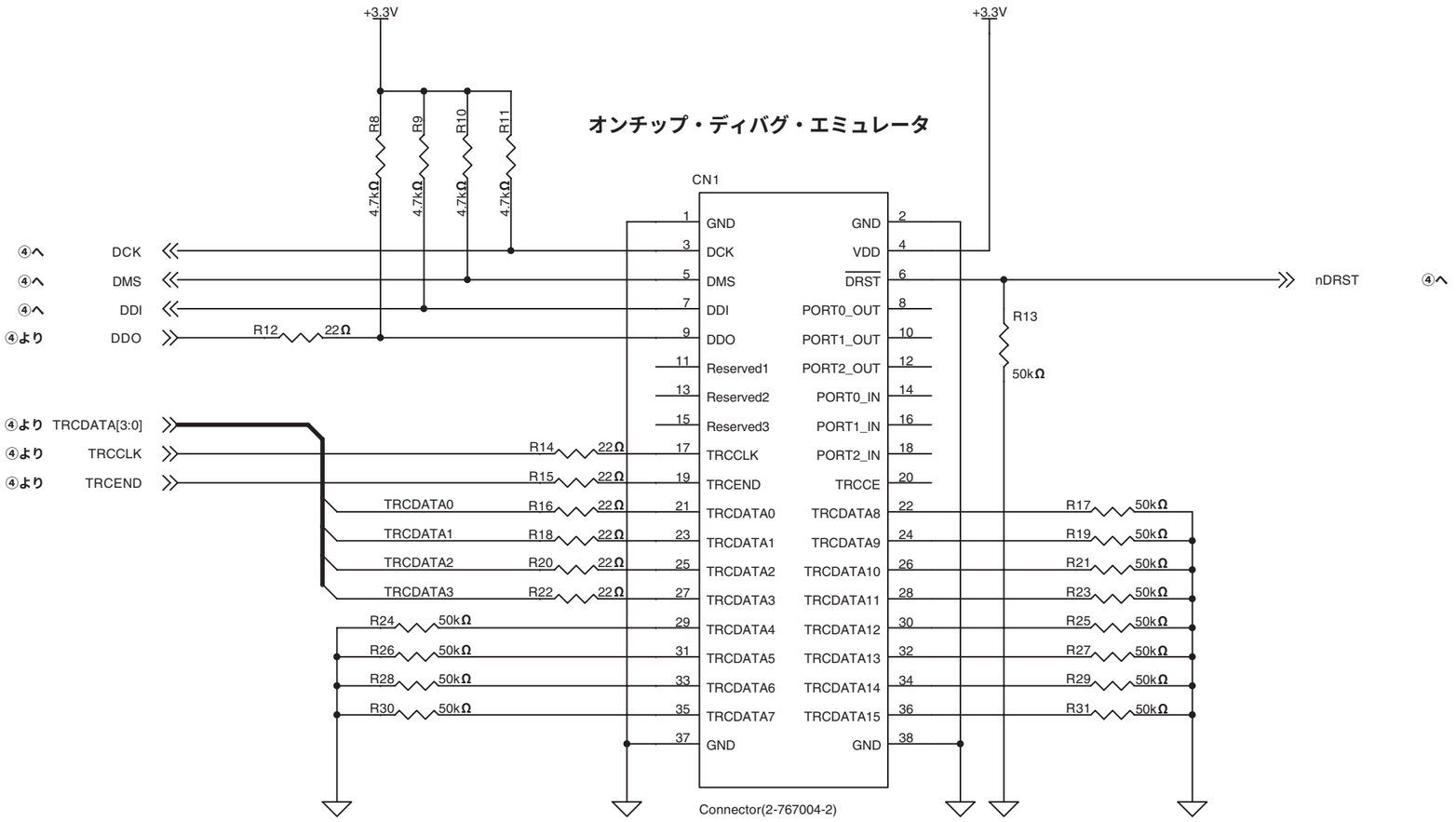
備考　③, ④, ⑥, ⑧は図番号を示します（例：③ → 図4－3）。

図4－11　USB周辺回路



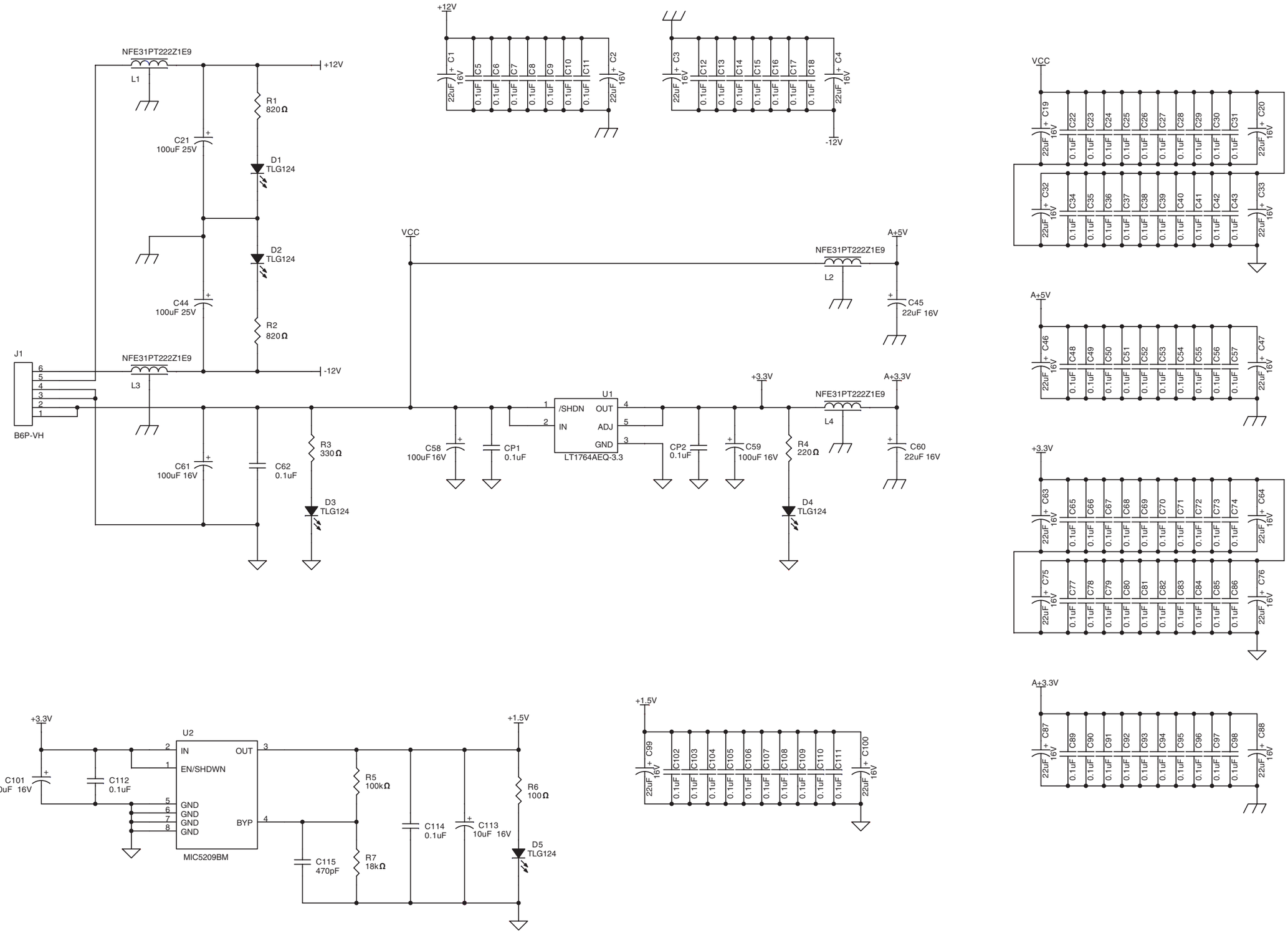
備考　③, ④は図番号を示します（例：③ → 図4－3）。

図4－12　オンチップ・ディバグ・エミュレータ



備考　④は図4－4を示します。

図4－13　電源周辺回路


+12V
C1 22uF 16V
C5 0.1uF
C6 0.1uF
C7 0.1uF
C8 0.1uF
C9 0.1uF
C10 0.1uF
C11 0.1uF
C2 22uF 16V
C3 22uF 16V
C12 0.1uF
C13 0.1uF
C14 0.1uF
C15 0.1uF
C16 0.1uF
C17 0.1uF
C18 0.1uF
C4 22uF 16V
-12V
NFE31PT222Z1E9
L1
+12V
R1
820Ω
C21
100uF 25V
D1
TLG124
D2
TLG124
C44
100uF 25V
R2
820Ω
J1
6
5
4
3
2
1
B6P-VH
NFE31PT222Z1E9
L3
-12V
C61
100uF 16V
C62
0.1uF
R3
330Ω
D3
TLG124
VCC
NFE31PT222Z1E9
L2
A+5V
C45
22uF 16V
U1
/SHDN
IN
OUT
ADJ
GND
LT1764AEQ-3.3
C58
100uF 16V
CP1
0.1uF
CP2
0.1uF
+3.3V
C59
100uF 16V
R4
220Ω
D4
TLG124
NFE31PT222Z1E9
L4
A+3.3V
C60
22uF 16V
VCC
C19 22uF 16V
C22 0.1uF
C23 0.1uF
C24 0.1uF
C25 0.1uF
C26 0.1uF
C27 0.1uF
C28 0.1uF
C29 0.1uF
C30 0.1uF
C31 0.1uF
C20 22uF 16V
C32 22uF 16V
C34 0.1uF
C35 0.1uF
C36 0.1uF
C37 0.1uF
C38 0.1uF
C39 0.1uF
C40 0.1uF
C41 0.1uF
C42 0.1uF
C43 0.1uF
C33 22uF 16V
A+5V
C46 22uF 16V
C48 0.1uF
C49 0.1uF
C50 0.1uF
C51 0.1uF
C52 0.1uF
C53 0.1uF
C54 0.1uF
C55 0.1uF
C56 0.1uF
C57 0.1uF
C47 22uF 16V
+3.3V
C63 22uF 16V
C65 0.1uF
C66 0.1uF
C67 0.1uF
C68 0.1uF
C69 0.1uF
C70 0.1uF
C71 0.1uF
C72 0.1uF
C73 0.1uF
C74 0.1uF
C64 22uF 16V
C75 22uF 16V
C77 0.1uF
C78 0.1uF
C79 0.1uF
C80 0.1uF
C81 0.1uF
C82 0.1uF
C83 0.1uF
C84 0.1uF
C85 0.1uF
C86 0.1uF
C76 22uF 16V
+3.3V
C101
10uF 16V
C112
0.1uF
U2
IN
EN/SHDWN
GND
GND
GND
GND
OUT
BYP
MIC5209BM
R5
100kΩ
C115
470pF
R7
18kΩ
C114
0.1uF
C113
10uF 16V
+1.5V
R6
100Ω
D5
TLG124
+1.5V
C99 22uF 16V
C102 0.1uF
C103 0.1uF
C104 0.1uF
C105 0.1uF
C106 0.1uF
C107 0.1uF
C108 0.1uF
C109 0.1uF
C110 0.1uF
C111 0.1uF
C100 22uF 16V
A+3.3V
C87 22uF 16V
C89 0.1uF
C90 0.1uF
C91 0.1uF
C92 0.1uF
C93 0.1uF
C94 0.1uF
C95 0.1uF
C96 0.1uF
C97 0.1uF
C98 0.1uF
C88 22uF 16V

4.2　サンプル・プログラム例

4.2.1　開発ツール

掲載してあるプログラム例は，すべてNECエレクトロニクス製の開発環境を使用して作成しております。

（1）Cコンパイラ・パッケージ（CA850）

Cコンパイラ・パッケージに関するユーザーズ・マニュアルは次の4つです。

- CA850　Cコンパイラ・パッケージ　操作編（U17293J）
- CA850　Cコンパイラ・パッケージ　C言語編（U17291J）
- CA850　Cコンパイラ・パッケージ　アセンブリー言語編（U17292J）
- CA850　Cコンパイラ・パッケージ　リンク・ディレクティブ編（U17294J）
- PM+（U17178J）

備考　C言語中の#pragma拡張記述やアセンブリ言語中の周辺内蔵I/Oレジスタ名略号記述，擬似命令記述などはNECエレクトロニクス製Cコンパイラ・パッケージ（CA850）固有のものです。サードパーティ製開発ツールを使用する場合は，それらの記述がそのまま使用できるとは限りませんので，十分に注意してください。

（2）ID850NW　V850シリーズ用Cソース・ディバッガ

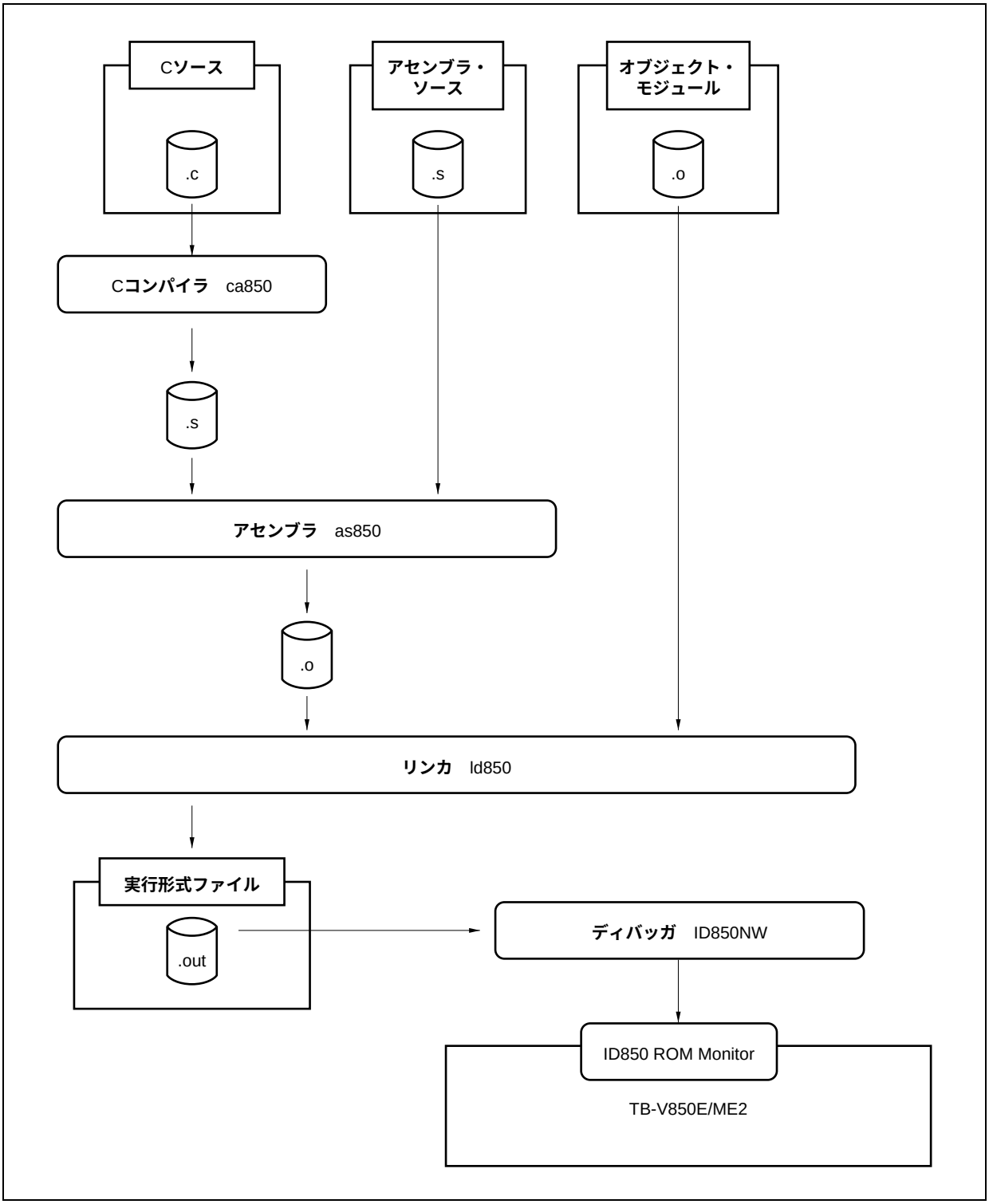
V850シリーズ用Cソース・ディバッガに関するユーザーズ・マニュアルです。

- ID850NW　統合デバッガ　操作編（U17369J）

（3）TB-V850E/ME2

V850E/ME2評価用ボードに接続するドータ・ボードです。TB-V850E/ME2の詳細については，4．1 TB-V850E/ME2の機能を参照してください。

図4－14　開発ツールの構成



はじめてNECエレクトロニクス製開発ツールを使う方は，導入として次のマニュアルが便利です。

・V800シリーズ開発ツール（32ビット対応）　アプリケーション・ノート　チュートリアル・ガイド（Windows®ベース）（U15196J）

4.2.2　プログラムの構成

サンプル・プログラムは次に示す構成になっています。

（1）リセット後のCPU初期化時の内蔵命令RAMでの簡単なプログラムの動作例

V850E/ME2のRESETハンドラ・アドレスは100000H番地です。この番地は外部メモリ空間のため，CPU本来の高速性を生かすために，CPUの初期化を行ったあと，CPUの内蔵命令RAMへプログラムを展開し実行します。

通常，V850E/ME2リセット後の初期化シーケンスは次に示す3段階で行われます。

① パワーオンまたはリセット解除後，自動的に100000H番地に分岐後，次の処理を実行

- CPU基本機能の初期化（4.2.3　CPU基本機能の初期化例参照）
- バス制御機能兼用端子の初期化（4.2.4　バス制御機能兼用端子の初期化例参照）
- 上記以外の兼用端子初期化（4.2.5　汎用入出力ポート兼用端子の初期化例参照）

② ロック・レジスタ（LOCKR）のLOCKビットの確認後，次の処理を実行

- 外部バスの周波数分周値設定（4.2.6　クロックの初期化例参照）
- 内部システム・クロック周波数分周値設定（4.2.6　クロックの初期化例参照）
- メイン・クロック供給制御を設定（4.2.6　クロックの初期化例参照）
- SDRAM関連の初期化（4.2.7　SDRAMの初期化例参照）

③ 内蔵命令RAMにプログラム・コードを転送後，次の処理を実行

- プログラムの先頭へジャンプ（4.2.8　内蔵命令RAMの転送例，4.2.9　内蔵命令RAMに転送するプログラム例参照）

（2）内蔵命令RAMで動的実行するプログラム例

外部EPROM空間に存在する複数のプログラムを同じ番地の内蔵命令RAMへ展開し，実行します。内蔵命令RAM空間内にすべてのプログラムが入りきらない大規模なアプリケーションの場合，外部メモリ空間から必要に応じて実行したいプログラムを内蔵命令RAM空間へダウンロードするイメージの動作となります。この動作は，4.2.10　内蔵命令RAMで動的実行するプログラム例に相当します。

（3）周辺機能を使った動作例

内蔵周辺I/Oや外部デバイスなどの動作を確認します。この動作は，4.2.11　内蔵UARTBnのFIFOプログラム例に相当します。

4. 2. 3　CPU基本機能の初期化例

V850E/ME2の初期化シーケンスでは，必ず最初に次の設定を行ってください。

- システム・ウエイト・コントロール・レジスタ（VSWC）
- データ・ウエイト・コントロール・レジスタ0, 1（DWC0, DWC1）
- アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）
- バス・サイクル・コントロール・レジスタ（BCC）

その後必要に応じて，チップ・エリア選択コントロール・レジスタ0, 1（CSC0, CSC1），バス・サイクル・タイプ・コンフィギュレーション・レジスタ0, 1（BCT0, BCT1），ローカル・バス・サイジング・コントロール・レジスタ（LBS），エンディアン・コンフィギュレーション・レジスタ（BEC），ライン・バッファ・コントロール・レジスタ0, 1（LBC0, LBC1），ページROMコンフィギュレーション・レジスタ（PRC），ポートDHファンクション・コントロール・レジスタ（PFCDH）などの設定を行ってください。

（1）システム・ウエイト機能の初期化

システム・ウエイト・コントロール・レジスタ（VSWC）の設定は次のとおりです。

図4－15　システム・ウエイト・コントロール・レジスタ（VSWC）の設定

アドレス：FFFFF06EH

	7	6	5	4	3	2	1	0
初期値	0	1	1	1	0	1	1	1
ビット名	－	－	－	－	－	－	－	－
設定値	0	0	1	1	0	0	1	1

動作周波数（fx）	VSWCの設定値
125.00 MHz＜fx≦150.00 MHz	33H

【プログラム例】

```
 [file:initial.s]
#########################################################
# Ｉ外部バス・アクセス性能に影響するレジスタの設定
#########################################################
#   内蔵周辺I/Oアクセスのウエイト設定 ####
     mov    0x33,  r6          -- 125.00MHz＜fx≦150.00MHz時のvswc設定値
     st.b   r6,    VSWC[r0]    -- システム・ウエイト・コントロール・レジスタ
```

(2) プログラマブル・ウエイト（ウエイト機能）初期化のポイント

- 内蔵命令RAM（リード・モード時），内蔵データRAM領域は，プログラマブル・ウエイトの対象外で，常にノー・ウエイトでアクセスを行います。内蔵周辺I/O領域は，プログラマブル・ウエイトの対象外で，VSWCレジスタの設定によりウエイト制御されます。
- ページROMオンページ・アクセス，SDRAMアクセスのウエイト制御は各メモリ・コントローラごとに行います。DWC0, DWC1の設定は無効です。
- DWC0, DWC1レジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。

図4－16　データ・ウエイト・コントロール・レジスタ0, 1（DWC0, DWC1）の設定

DWC0　　アドレス：FFFFF484H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
ビット名	0	DW32	DW31	DW30	0	DW22	DW21	DW20	0	DW12	DW11	DW10	0	DW02	DW01	DW00
設定値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
		$\overline{CS3}$				$\overline{CS2}$				$\overline{CS1}$				$\overline{CS0}$		

DWC1　　アドレス：FFFFF486H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
ビット名	0	DW72	DW71	DW70	0	DW62	DW61	DW60	0	DW52	DW51	DW50	0	DW42	DW41	DW40
設定値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		$\overline{CS7}$				$\overline{CS6}$				$\overline{CS5}$				$\overline{CS4}$		

ビット位置	ビット名	意　味
14-12, 10-8, 6-4, 2-0	DWn2-DWn0	$\overline{CSn}$空間に挿入するウエイトのステート数を指定 ・$\overline{CS0}$（EPROM空間（16ビット幅））：3ウエイト挿入 ・$\overline{CS1}$（SDRAM空間（32ビット幅））：ウエイトなし ・$\overline{CS2}$（SRAM空間（32ビット幅））：ウエイトなし ・$\overline{CS3}$（SRAM空間（16ビット幅））：ウエイトなし ・$\overline{CS4}$（SDRAM空間（16ビット幅））：ウエイトなし ・$\overline{CS5}$（SRAM空間（8ビット幅））：ウエイトなし ・$\overline{CS6}$（SDRAM空間（8ビット幅））：ウエイトなし ・$\overline{CS7}$（未使用領域）

備考　n = 0-7

【プログラム例】

```
 [file:initial.s]
#########################################################
#   各CS空間ごとのデータ・ウエイトを指定
#########################################################
     mov    0x0003,   r6        -- CS0:EPROM    空間 3 wait(TW)挿入
                                -- CS1:SDRAM    空間 0 wait(TW)
                                -- CS2:SRAM     空間 0 wait(TW)
                                -- CS3:SRAM     空間 0 wait(TW)
     mov    0x0000,   r7        -- CS4:SDRAM    空間 0 wait(TW)
                                -- CS5:SRAM     空間 0 wait(TW)
                                -- CS6:SDRAM    空間 0 wait(TW)
                                -- CS7:なし
     st.h   r6,       DWC0[r0]  -- データ・ウエイト・コントロール・レジスタ0
     st.h   r7,       DWC1[r0]  -- データ・ウエイト・コントロール・レジスタ1
```

(3) アドレス・セットアップ・ウエイト（ウエイト機能）の初期化のポイント

- 内蔵命令RAM（リード・モード時），内蔵データRAM領域，内蔵周辺I/O領域は，アドレス・セットアップ・ウエイト挿入の対象外です。
- 内蔵命令RAM（ライト・モード時）にアクセスしたときは，$\overline{CS0}$空間に対するアドレス・セットアップ・ウエイト設定値が有効になります。
- $\overline{WAIT}$端子による外部ウエイト機能でアドレス・セットアップ・ウエイトを挿入することはできません。
- ASCレジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。

図4－17　アドレス・セットアップ・ウエイト・コントロール・レジスタ（ASC）の設定

アドレス：FFFFF48AH

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ビット名	AC71	AC70	AC61	AC60	AC51	AC50	AC41	AC40	AC31	AC30	AC21	AC20	AC11	AC10	AC01	AC00
設定値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

ビット位置	ビット名	意　味
15-0	ACn1, ACn0	SRAM／ページROMサイクルの前に挿入するアドレス・セットアップ・ウエイト・ステート数を$\overline{CSn}$空間ごとに指定 ・$\overline{CS0}$-$\overline{CS6}$：挿入しない ・$\overline{CS7}$（未使用領域）

備考　n = 0-7

【プログラム例】

```
 [file:initial.s]
#########################################################
#    各CS空間ごとのアドレス・セットアップ・ウエイトを指定
#########################################################
     mov    0x0000,   r6        -- CS0:EPROM    空間 0 wait(TASW)
                                -- CS1:SDRAM     空間 0 wait(TASW)
                                -- CS2:SRAM      空間 0 wait(TASW)
                                -- CS3:SRAM      空間 0 wait(TASW)
                                -- CS4:SDRAM     空間 0 wait(TASW)
                                -- CS5:SRAM      空間 0 wait(TASW)
                                -- CS6:SDRAM     空間 0 wait(TASW)
                                -- CS7:なし
     st.h   r6,       ASC[r0]   -- アドレス・セットアップ・ウエイト・コントロール・レジスタ
```

(4) アイドル・ステート挿入機能の初期化のポイント

- 内蔵命令RAM（リード・モード時），内蔵データRAM領域，内蔵周辺I/O領域は，アイドル・ステート挿入の対象外です。
- 内蔵命令RAM（ライト・モード時）にアクセスしたときは，$\overline{CS0}$空間に対するアイドル・ステート設定値が有効になります。
- BCCレジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。
- アイドル・ステート時に$\overline{CSn}$信号はアクティブになりません（n = 0-7）。

図4－18　バス・サイクル・コントロール・レジスタ（BCC）の設定

アドレス：FFFFF488H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ビット名	BC71	BC70	BC61	BC60	BC51	BC50	BC41	BC40	BC31	BC30	BC21	BC20	BC11	BC10	BC01	BC00
設定値	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0
	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

ビット位置	ビット名	意　味
15-0	BCn1, BCn0	$\overline{CSn}$空間に対するアイドル・ステートの挿入を指定 ・$\overline{CS0}$（EPROM空間）：2アイドル挿入 ・$\overline{CS1}$, $\overline{CS4}$, $\overline{CS6}$：挿入しない ・$\overline{CS2}$, $\overline{CS3}$, $\overline{CS5}$：1アイドル挿入 ・CS7（未使用領域）

備考　n = 0-7

【プログラム例】

```
[file:initial.s]
#########################################################
#    各CS空間ごとのアイドル・ステートを指定
#########################################################
     mov    0x0452,   r6        -- CS0:EPROM    空間 2 idle(TI)挿入
                                -- CS1:SDRAM    空間 0 idle(TI)
                                -- CS2:SRAM     空間 1 idle(TI)挿入
                                -- CS3:SRAM     空間 1 idle(TI)挿入
                                -- CS4:SDRAM    空間 0 idle(TI)
                                -- CS5:SRAM     空間 1 idle(TI)挿入
                                -- CS6:SDRAM    空間 0 idle(TI)
                                -- CS7:なし
     st.h   r6,       BCC[r0]   -- バス・サイクル・コントロール・レジスタ
```

(5) メモリ・ブロック機能の初期化のポイント

256 Mバイトのメモリ空間が64 Mバイトずつ４つの領域（エリア0-エリア3）に分割されています。

エリア0：下位8 Mバイト（0000000H-07FFFFFH）$\overline{CS0}$, $\overline{CS1}$, $\overline{CS2}$信号選択可能

エリア0：残りの56 Mバイト（0800000H-3FFFFFFH）$\overline{CS1}$信号

エリア1：64 Mバイト（4000000H-7FFFFFFH）$\overline{CS3}$信号固定

エリア2：64 Mバイト（8000000H-BFFFFFFH）$\overline{CS4}$信号固定

エリア3：初めの56 Mバイト（C000000H-F7FFFFFH）$\overline{CS6}$信号

エリア3：上位8 Mバイト（F800000H-FFFFFFFH）$\overline{CS7}$, $\overline{CS6}$, $\overline{CS5}$信号選択可能

図4－19　チップ・エリア選択コントロール・レジスタ0（CSC0）の設定

アドレス：FFFFF060H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1
ビット名	CS33	CS32	CS31	CS30	CS23	CS22	CS21	CS20	CS13	CS12	CS11	CS10	CS03	CS02	CS01	CS00
設定値	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1

設定に意味を持ちません。　　設定に意味を持ちません。

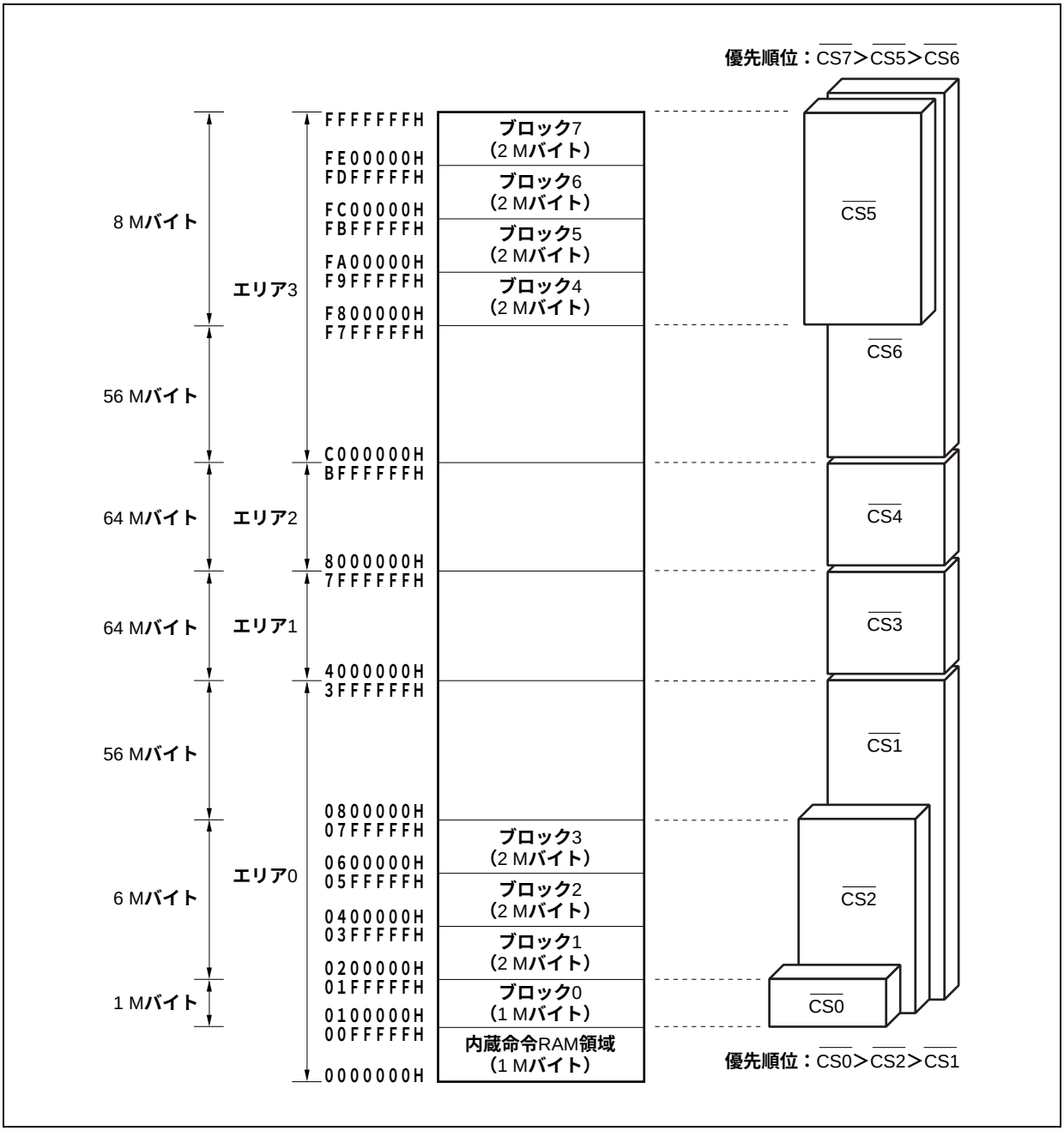
図4－20　チップ・エリア選択コントロール・レジスタ1（CSC1）の設定

アドレス：FFFFF062H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1
ビット名	CS43	CS42	CS41	CS40	CS53	CS52	CS51	CS50	CS63	CS62	CS61	CS60	CS73	CS72	CS71	CS70
設定値	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

設定に意味を持ちません。　　設定に意味を持ちません。

図4－21　プログラム例の$\overline{CSn}$信号設定（CSC0レジスタ = 0E01H，CSC1レジスタ = 0F00H）



【プログラム例】

```
[file:initial.s]
#########################################################
#   256MBのメモリ空間のうち，
#   下位8MB(00000000H-07FFFFFFH)
#   上位8MB(0F800000H-0FFFFFFFH)
#   を2MB単位でのメモリ・ブロックに分割し
#   チップ・セレクト信号を指定
#########################################################
     mov    0x0e01,   r6        -- エリア0(64MB)
                                -- ブロック0  x0100000H-x01fffffH:( 1MB)CS0
                                -- ブロック1  x0200000H-x03fffffH:( 2MB)CS2
                                -- ブロック2  x0400000H-x05fffffH:( 2MB)CS2
                                -- ブロック3  x0600000H-x07fffffH:( 2MB)CS2
                                --          x0800000H-x3fffffffH:(56MB)CS1固定

                                -- エリア1(64MB)
                                --          x4000000H-x7fffffffH:(64MB)CS3固定
                                -- エリア2(64MB)
                                --          x8000000H-xbfffffffH:(64MB)CS4固定

     mov    0x0f00,   r7        -- エリア3(64MB)
                                --          xc000000H-xf7fffffH:(56MB)CS6固定
                                -- ブロック4  xf800000H-xf9fffffH:( 2MB)CS5
                                -- ブロック5  xfa00000H-xfbfffffH:( 2MB)CS5
                                -- ブロック6  xfc00000H-xfdfffffH:( 2MB)CS5
                                -- ブロック7  xfe00000H-xfeffffffH:( 2MB)CS5
                                -- CS7 未　　使　　用
     st.h   r6,       CSC0[r0]  -- チップ・エリア選択コントロール・レジスタ0
     st.h   r7,       CSC1[r0]  -- チップ・エリア選択コントロール・レジスタ1
```

(6) バス・サイクル・タイプ制御機能の初期化のポイント

BCT0, BCT1レジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。

図4－22　バス・サイクル・タイプ・コンフィギュレーション・レジスタ0, 1（BCT0, BCT1）の設定

BCT0　　アドレス：FFFFF480H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
ビット名	ME3	0	BT31	BT30	ME2	0	BT21	BT20	ME1	0	BT11	BT10	ME0	0	BT01	BT00
設定値	1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0
			$\overline{CS3}$				$\overline{CS2}$				$\overline{CS1}$				$\overline{CS0}$	

BCT1　　アドレス：FFFFF482H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
ビット名	ME7	0	BT71	BT70	ME6	0	BT61	BT60	ME5	0	BT51	BT50	ME4	0	BT41	BT40
設定値	0	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1
			$\overline{CS7}$				$\overline{CS6}$				$\overline{CS5}$				$\overline{CS4}$	

ビット位置	ビット名	意　味
15, 11, 7, 3	MEn	メモリ・コントローラの動作許可を$\overline{CSn}$信号ごとに指定 ・$\overline{CS7}$　：動作禁止 ・$\overline{CS0}$-$\overline{CS6}$　：動作許可
13, 12, 9, 8, 5, 4, 1, 0	BTn1, BTn0	直接接続するデバイスを$\overline{CSn}$信号ごとに指定 ・$\overline{CS0}$, $\overline{CS2}$, $\overline{CS3}$, $\overline{CS5}$　：SRAMまたは外部I/O ・$\overline{CS1}$, $\overline{CS4}$, $\overline{CS6}$　：SDRAM ・$\overline{CS7}$（未使用領域）

備考　n = 0-7

【プログラム例】

```
[file:initial.s]
#########################################################
#    各CS空間ごとの接続する外部デバイスを指定
#########################################################
     mov    0x88b8,   r6        -- CS0:EPROM x0100000H-x011ffffH(128KB)
                                -- CS1:SDRAM x1000000H-x2ffffffH( 32MB)
                                -- CS2:SRAM  x0200000H-x027ffffH(512KB)
                                -- CS3:SRAM  x4000000H-x403ffffH(256KB)
     mov    0x0b8b,   r7        -- CS4:SDRAM x8000000H-x8ffffffH( 16MB)
                                -- CS5:SRAM  xF800000H-xF81ffffH(128KB)
                                -- CS6:SDRAM xC000000H-xCffffffH( 16MB)
                                -- CS7:なし
     st.h   r6,       BCT0[r0]  -- バス・サイクル・タイプ・コンフィギュレーション・レジスタ0
     st.h   r7,       BCT1[r0]  -- バス・サイクル・タイプ・コンフィギュレーション・レジスタ1
```

(7) バス・サイジング機能の初期化のポイント

LBSレジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。

図4－23　ローカル・バス・サイジング・コントロール・レジスタ（LBS）の設定

アドレス：FFFFF48EH

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
ビット名	LB71	LB70	LB61	LB60	LB51	LB50	LB41	LB40	LB31	LB30	LB21	LB20	LB11	LB10	LB01	LB00
設定値	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1
	$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$	

ビット位置	ビット名	意　味
15-0	LBn1, LBn0	$\overline{CSn}$空間のデータ・バス幅を指定 ・$\overline{CS5}$, $\overline{CS6}$　：8ビット ・$\overline{CS0}$, $\overline{CS3}$, $\overline{CS4}$　：16ビット ・$\overline{CS1}$, $\overline{CS2}$　：32ビット ・$\overline{CS7}$（未使用領域）

注意　32ビット・モードで起動した場合の初期値は，全$\overline{CS}$空間32ビット・データ・バス幅のAAAAHです。

備考　n = 0-7

【プログラム例】

```
[file:initial.s]
#########################################################
#   CS空間ごとのデータ・バス幅を指定
#########################################################
     mov    0x0169,   r6        -- CS0:EPROM  x0100000H-x011ffffH(128KB)16bit
                                -- CS1:SDRAM  x1000000H-x2fffffH( 32MB)32bit
                                -- CS2:SRAM   x0200000H-x027ffffH(512KB)32bit
                                -- CS3:SRAM   x4000000H-x403ffffH(256KB)16bit
                                -- CS4:SDRAM  x8000000H-x8fffffH( 16MB)16bit
                                -- CS5:SRAM   xF800000H-xF81ffffH(128KB) 8bit
                                -- CS6:SDRAM  xC000000H-xCfffffH( 16MB) 8bit
                                -- CS7:なし                              8bit
     st.h   r6,       LBS[r0]   -- ローカル・バス・サイジング・コントロール・レジスタ
```

（8）エンディアン制御機能の初期化のポイント

・内蔵命令RAM，内蔵データRAM領域（および3FFB000H-3FFEFFFH番地の内蔵データRAMミラー領域），内蔵周辺I/O領域，外部メモリのプログラム領域は，リトル・エンディアン形式固定になるので，BECレジスタの設定は無効です。

・BECレジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。

・NECエレクトロニクス製純正開発ツール（ディバッガやコンパイラ）においては，ビッグ・エンディアン形式の場合，使用制限事項がありますので注意してください。

制限事項については，V850E/ME2　**ユーザーズ・マニュアル　ハードウエア編**　4. 5. 4　NEC**エレクトロニクス製開発ツールにおけるビッグ・エンディアン形式の使用制限**を参照してください。

図4－24　エンディアン・コンフィギュレーション・レジスタ（BEC）の設定

アドレス：FFFFF068H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ビット名	0	BE70	0	BE60	0	BE50	0	BE40	0	BE30	0	BE20	0	BE10	0	BE00
設定値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		$\overline{CS7}$		$\overline{CS6}$		$\overline{CS5}$		$\overline{CS4}$		$\overline{CS3}$		$\overline{CS2}$		$\overline{CS1}$		$\overline{CS0}$

ビット位置	ビット名	意　味
14, 12, 10, 8, 6, 4, 2, 0	BEn0	エンディアン形式を$\overline{CSn}$空間ごとに指定 ・$\overline{CS0}$-$\overline{CS6}$：リトル・エンディアン ・$\overline{CS7}$（未使用領域）

備考　n = 0-7

【プログラム例】

```
[file:initial.s]
#########################################################
#    各CS空間ごとのエンディアンを指定
#########################################################
     mov    0x0000,   r6         -- CS0-CS7全空間 リトル・エンディアン
     st.h   r6,       BEC[r0]    -- エンディアン・コンフィギュレーション・レジスタ
```

(9) データ・リード制御機能の初期化のポイント

- LBC0, LBC1レジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。
- $\overline{\text{CSn}}$空間ごとに先読み機能を設定する場合には，先読み機能を有効にする$\overline{\text{CSn}}$空間に対して，BCCレジスタによるアイドル・ステートの挿入をしないでください（n = 0-7）。先読み機能を有効にする$\overline{\text{CSn}}$空間に対して，アイドル・ステートの挿入が必要な場合には，全$\overline{\text{CS}}$空間に対して，先読み機能を有効にする（LBC0, LBC1レジスタをともに3333Hに設定）か，先読み機能を無効（LBC0, LBC1レジスタをともに0000Hに設定）するかしてください。

図4－25　ライン・バッファ・コントロール・レジスタ0, 1（LBC0, LBC1）の設定

LBC0　　アドレス：FFFFF490H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ビット名	0	0	RB31	RB30	0	0	RB21	RB20	0	0	RB11	RB10	0	0	RB01	RB00
設定値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			$\overline{\text{CS3}}$				$\overline{\text{CS2}}$				$\overline{\text{CS1}}$				$\overline{\text{CS0}}$	

LBC1　　アドレス：FFFFF492H

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ビット名	0	0	RB71	RB70	0	0	RB61	RB60	0	0	RB51	RB50	0	0	RB41	RB40
設定値	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			$\overline{\text{CS7}}$				$\overline{\text{CS6}}$				$\overline{\text{CS5}}$				$\overline{\text{CS4}}$	

ビット位置	ビット名	意　味
13, 12, 9, 8, 5, 4, 1, 0	RBn1, RBn0	リード・バッファの動作条件（先読みを行うタイミング）を$\overline{\text{CSn}}$空間ごとに指定 ・$\overline{\text{CS0}}$-$\overline{\text{CS6}}$：先読みなし（リード・バッファの動作禁止） ・$\overline{\text{CS7}}$（未使用領域）

備考　n = 0-7

【プログラム例】

```
[file:initial.s]
#########################################################
#   CS空間ごとのデータ・リード制御機能を指定
#########################################################
     mov    r0,       r6        -- CS0-CS3全空間 先読みなし(リード・バッファ動作の禁止)
     mov    r0,       r7        -- CS4-CS7全空間 先読みなし(リード・バッファ動作の禁止)
     st.h   r6,       LBC0[r0]  -- ライン・バッファ・コントロール・レジスタ0
     st.h   r7,       LBC1[r0]  -- ライン・バッファ・コントロール・レジスタ1
```

（10）ページROMの初期化のポイント

・PRCレジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。

・オンページ・アクセス時の場合，PRCレジスタの設定によってウエイト制御します。

・オフページ・アクセス時の場合，DWCnレジスタの設定によってウエイト制御します（n = 0,1）。

各レジスタの設定については，このプログラム例では，ページROMを使用していないため，ディフォールト設定値のままです。

【プログラム例】

```
[file:initial.s]
#########################################################
#   オンページ・アクセス時のウエイトの指定など（未実装）
#########################################################
--   mov    x7000,    r6         -- 2×32bit,4×16bit,8×8bit
                                 -- オン・ページ・アクセス時に 7 wait(TW)挿入
--   st.h   r6,       PRC[r0]    -- ページROMコンフィギュレーション・レジスタ
```

（11）外部データ・バスを16ビット・モード（MODE1, MODE0 = 01）で起動し，32ビット・データ・バスと混在使用する場合の注意

・PFCDHレジスタのBMODCNビットにより，D16-D31端子の使用／未使用を設定できます。

ビット名	意　味
BMODCN	16ビット・モード（16ビット・データ・バス）でのD16-D31端子の使用／未使用を指定 0：16ビット・モードでの起動時にD16-D31端子として使用しない（データ・バス幅：16/8ビット） 1：16ビット・モードでの起動時にD16-D31端子として使用する（データ・バス幅：32/16/8ビット）

・外部データ・バスを16ビット・モードで起動し，32ビット・データ・バスとの混在で使用する場合は，PFCDHレジスタのBMODCNビットに1を書き込んだあと，必ず読み出した値が1になっていることを確認してから，32ビット・バス幅の外部デバイスへのアクセスを行なってください。

・BMODCNビットの書き換えは，1回のみ有効です。2回以上書き換えた場合の動作は保証しません。

・16ビット・モードで起動（MODE1, MODE0 = 01）した場合，32ビット・データ・バスで使用する外部素子がない場合は，BMODCNビットの設定は必要ありません。

【プログラム例】

```
[file:initial.s]
#########################################################
#   D16-D31端子の使用有無を指定
#########################################################
-- PMDH,PMCDH,PFCDHの設定は16bitバス・モード時の設定です。
-- MODE1端子=1,MODE0端子=0設定時16bitバス・モードになります。
     mov    0x0000,   r6         -- PDH15-PDH0のインアクティブ・レベル・ロウを設定
     st.h   r6,       PDH[r0]
     mov    0x0000,   r6         -- PDH15-PDH0は出力モード選択
     st.h   r6,       PMDH[r0]
     mov    0x8000,   r6         -- PDH15は制御端子選択(INTPD15/PWM1)
     st.h   r6,       PMCDH[r0]
-- PFCDHのBMODCN=1とすると32bitデータ・バスとの混在使用が可能になります。
--   mov    0x8000,   r6         -- PDH15はPWM1端子選択
     mov    0x0001,   r6         -- 16bitモードでの起動時にPDH15-PDH0をD31-D16端子として使用
     st.h   r6,       PFCDH[r0] -- ポートDHファンクション・コントロール・レジスタを書き込み
-- PFCDHのBMODCN=1設定時PDH15-PDH0はD31-D16端子となり，PDH,PMDH,PMCDHの上記設定は無効です。

_BMODCN_NOFIX:
     ld.h   PFCDH[r0],r6         -- ポートDHファンクション・コントロール・レジスタを読み出し
     movea  0x0001,   r0, r7     -- BMODCN bit抽出
     and    r7,       r6
     cmp    r6,       r7         -- BMODCN設定未完了?
     jnz    _BMODCN_NOFIX       -- はい
-- MODE1端子=0,MODE0端子=0設定時32bitバス・モードとなり，PMDH,PMCDH,PFCDHの上記設定は無効です。
```

4.2.4　バス制御機能兼用端子の初期化例

CPU基本機能の初期化に加えて，バス制御機能兼用端子の初期化も行います。

まず，バス制御機能兼用端子の設定方法についての基本的な考え方を記述します。バス制御機能端子には，バス制御機能以外に，汎用入出力（ポート）機能，割り込み機能，その他の制御機能と兼用になっています。兼用機能の指定は，レジスタ設定により行い，設定の流れは大きく次の3パターンあります。

表4－1　ポート機能／バス制御機能／その他の制御機能／割り込み機能を持つ兼用端子の設定

レジスタ / ポート名	PMCn	PMn	PFCn	INTRn	INTFn
ポートn	制御機能	設定無効	バス制御機能 または その他の制御機能[注]	設定無効	
			割り込み機能	立ち下がりエッジ	
				立ち上がりエッジ	
				レベル検出（H/L検出）	
				立ち上がり／立ち下がり両エッジ	
	ポート機能	入力	設定無効		
		出力			

注　ポートALの制御機能：バス制御（A0, A1端子）機能のみ

ポートDHの制御機能：バス制御（D16-D31）機能とその他の制御機能

備考1.　H：ハイ・レベル入力

L：ロウ・レベル入力

2.　n = AL, DH（n = ALのとき，PFCnレジスタはPFCALLレジスタ）

表4－2　ポート機能／バス制御機能を持つ兼用端子の設定

レジスタ / ポート名	PMCn	PMn	PFCn	INTRn	INTFn
ポートn	制御機能	設定無効	なし		
	ポート機能	入力			
		出力			

備考　n = AH, CD

表4－3　ポート機能／バス制御機能／その他のバス制御機能を持つ兼用端子の設定

レジスタ / ポート名	PMCn	PMn	PFCn	INTRn	INTFn
ポートn	制御機能	設定無効	バス制御機能	なし	
			その他のバス制御機能		
	ポート機能	入力	設定無効		
		出力			

備考　n = CS, CT, CM

次に，CPU制御機能の初期化時に注意するポイントを示します。

・ポート・モードからコントロール・モードへの切り替え手順
・ビット操作命令でポート操作を行う場合の注意
・割り込みトリガ・モード設定時の注意
・ポートDH設定時の注意
・$\overline{\text{IOWR}}$, $\overline{\text{IORD}}$信号出力の動作の許可／禁止設定

(1) ポート・モードからコントロール・モードへの切り替え手順

コントロール・モード時に出力または入出力端子として動作するポートをポート・モードからコントロール・モードに切り替える場合は，必ず次に示す手順で設定してください。

① コントロール・モードで出力する信号のインアクティブ・レベルをポートnの該当するビットに設定します（n = AL, AH, DH, CS, CT, CM, CD）。
② ポートnモード・コントロール・レジスタ（PMCn）により，コントロール･モードに切り替えます。

注意　上記①を行わない場合は，ポート・モードからコントロール・モードに切り替える際にポートnの内容が一瞬出力されることがあります（n = AL, AH, DH, CS, CT, CM, CD）。

(2) ビット操作命令でポート操作を行う場合の注意

ビット操作命令（SET1, CLR1, NOT1）でポート操作を行う場合，ポートに対してバイト・データ・リードを行い，操作対象のビットのみデータの加工を行い，変換後のバイト・データをポートに書き戻します（リード・モディファイ・ライト）。たとえば，入力／出力が混在しているポートでは，操作対象ビット以外のビットにも出力ラッチの内容が上書きされるため，入力端子の出力ラッチは不定になります（ただし，入力モードの場合，出力バッファがオフしているため，端子状態は変化しません）。したがって，ポートを入力から出力に切り替える場合は，ポートnレジスタ（Pn）の該当するビットに出力期待値を設定してから，出力ポートに切り替えてください（n = AL, AH, DH, CS, CT, CM, CD）。また，コントロール・モードと出力ポートが混在する場合も同様です。

(3) 割り込みトリガ・モード設定時の注意

トリガ・モードを設定する場合は，PMCnレジスタを設定したあとで行ってください（n = AL, DH）。

INTRn, INTFnレジスタを設定したあとでPMCnレジスタの設定を行うと，PMCnレジスタの設定タイミングで不正な割り込みが発生することがあります。

(4) ポートDH設定時の注意

(a) バス制御（D16-D31）機能

① ポートDHをバス制御（D16-D31）機能へ切り替え

ポートDHのバス制御（D16-D31）機能への切り替えは，基本的にMODE1, MODE0端子の動作モード指定によって決定されます。MODE1, MODE0端子とポートDH設定レジスタとの関係を次に示します。

表4－4　MODE1, MODE0端子とポートDH設定レジスタとの関係

モード / レジスタ	16ビット・モード（MODE1, MODE0 = 01）	32ビット・モード（MODE1, MODE0 = 00）
PDH	有効（ポートまたは制御端子として機能）	無効（D16-D31端子として機能）
PMDH	有効（入力／出力の切り替え）	無効（意味を持ちません）
PMCDH	有効（入出力ポート／制御端子の切り替え）	無効（意味を持ちません）
PFCDH	有効（制御端子機能切り替え，およびD16-D31端子機能の有効／無効設定）	無効（意味を持ちません）

② 16ビット・モード起動時にポートDHをバス制御（D16-D31）機能へ切り替え

16ビット・モードであっても，ポートDHをバス制御（D16-D31）機能にし，32ビット・データ・バス幅の外部デバイスと混在させて動作させることもできます。PFCDHレジスタのBMODCNビットによって設定します。ポートDHの制御端子機能を次に示します。

表4－5　MODE1, MODE0端子とBMODCNビットによるPDH制御端子機能の設定

制御 / ポート名	MODE1	MODE0	データ・モード	BMODCN	端子機能
ポートDH	ロウ・レベル	ロウ・レベル	32ビット・モード	設定無効	D16-D31
	ロウ・レベル	ハイ・レベル	16ビット・モード	1	
				0	その他の制御機能

16ビット・モードでの起動時，PFCDHレジスタのBMODCNビットの初期値は0（D16-D31端子機能無効（データ・バス幅32ビット外部デバイス使用不可））であることに注意してください。また，16ビット・モードでの起動時，PFCDHレジスタのBMODCNビット = 1（D16-D31端子機能有効（データ・バス幅32ビット外部デバイス使用可））の場合は，その他の制御機能は使用できません。

この設定の詳細については，4. 2. 3 (11) 外部データ・バスを16ビット・モード（MODE1, MODE0 = 01）で起動し，32ビット・データ・バスと混在使用する場合の注意を参照してください。

(b) タイマENC1n制御入力と兼用の端子に関する留意点

① PDH7, PDH11端子をタイマENC1n制御入力へ切り替え

PDH7, PDH11端子は，PFCDHレジスタによって割り込み機能を選択しなかった場合，タイマENC1nの外部キャプチャ・トリガ（INTP1n1）入力／TCLR1n入力となりますが，入力モードを切り替えるレジスタはありません（n = 0, 1）。したがって，次に示すタイマENC1n設定レジスタの組み合わせで，使用する入力モードを決定します。

表4－6　タイマENC1nの外部キャプチャ・トリガ（INTP1n1）入力／TCLR1n入力の切り替え設定

PDH7, PDH11の機能	設定値				割り込み
	TUM1nレジスタ	TMC1nレジスタ	CCR1nレジスタ	CC1nIC1レジスタ	
TCLR1n入力として使用	1000xx00B	0x00xx00B	0000000xB	x0000xxxB	INTP1n1発生
				x1000xxxB	INTP1n1マスク
			0000001xB	x0000xxxB	INTCC1n1発生
				x1000xxxB	INTCC1n1マスク
		0x00xx01B	0000001xB	x0000xxxB	INTCC1n1発生
				x1000xxxB	INTCC1n1マスク
		0x00xx10B	0000000xB	x0000xxxB	INTP1n1発生
				x1000xxxB	INTP1n1マスク
			0000001xB	x0000xxxB	INTCC1n1発生
				x1000xxxB	INTCC1n1マスク
		0x00xx11B	0000001xB	x0000xxxB	INTCC1n1発生
				x1000xxxB	INTCC1n1マスク
	1000xx01B	設定無効	0000000xB	x0000xxxB	INTCC1n1発生
				x1000xxxB	INTCC1n1マスク
	上記以外は設定禁止				
外部キャプチャ・トリガ（INTP1n1）として使用	0000xx00B	設定無効		x0000xxxB	INTP1n1発生
				x1000xxxB	INTP1n1マスク
	1000xx00B	0x00xx01B	0000000xB	x0000xxxB	INTP1n1発生
				x1000xxxB	INTP1n1マスク
		0x00xx11B		x0000xxxB	INTP1n1発生
				x1000xxxB	INTP1n1マスク
	1000xx01B	設定無効		x0000xxxB	INTP1n1発生
				x1000xxxB	INTP1n1マスク
	上記以外は設定禁止				

備考　n = 0, 1

② PDH6, PDH10端子をタイマENC1n制御入力へ切り替え

PDH6,PDH10端子は，PFCDHレジスタによって割り込み機能を選択しなかった場合，タイマENC1nの外部キャプチャ・トリガ（INTP1n0）入力／TCUD1n入力となりますが，入力モードを切り替えるレジスタはありません（n = 0, 1）。したがって，次に示すタイマENC1n設定レジスタの組み合わせで，使用する入力モードを決定します。

表4－7　タイマENC1nの外部キャプチャ・トリガ（INTP1n0）入力／TCUD1n入力の切り替え設定

PDH6, PDH10の機能	設定値				割り込み
	TUM1nレジスタ	TMC1nレジスタ	CCR1nレジスタ	CC1nIC0レジスタ	
TCUD1n入力として使用	1000xx0xB	設定無効	000000x0B	x0000xxxB	INTP1n0発生
				x1000xxxB	INTP1n0マスク
			000000x1B	x0000xxxB	INTCC1n0発生
				x1000xxxB	INTCC1n0マスク
	上記以外は設定禁止				
外部キャプチャ・トリガ（INTP1n0）として使用	0000xx00B	設定無効		x0000xxxB	INTP1n0発生
				x1000xxxB	INTP1n0マスク
	上記以外は設定禁止				

備考　n = 0, 1

③ タイマENC1n制御入力のSESA1n設定とPMCDH設定の順番について

ポートDHをタイマENC1n制御入力端子として使用し，INTP100, INTP101, INTP110, INTP111, TIUD10, TIUD11, TCUD10, TCUD11, TCLR10, TCLR11端子のトリガ・モードを設定する場合はPMCDHレジスタを設定したあとで行ってください。SESA1nレジスタを設定したあとでPMCDHレジスタの設定を行うと，PMCDHレジスタの設定タイミングで不正な割り込みや誤カウント，誤クリアが発生することがあります。

(5) $\overline{\text{IOWR}}$, $\overline{\text{IORD}}$信号出力の動作の許可／禁止設定

次に$\overline{\text{IOWR}}$, $\overline{\text{IORD}}$信号出力の動作の許可／禁止設定のポイントを示します。

- SRAM，外部ROM，外部I/Oを対象としたフライバイのDMA転送時は，IOENビットの設定にかかわらず$\overline{\text{IORD}}$, $\overline{\text{IOWR}}$信号が出力されます。

  外部I/O→外部メモリへのフライバイ転送時：$\overline{\text{IORD}}$, $\overline{\text{WR}}$信号がアクティブ
  外部メモリ→外部I/Oへのフライバイ転送時：$\overline{\text{IOWR}}$, $\overline{\text{RD}}$信号がアクティブ

ページROMサイクルでは，IOENビットの設定は意味を持ちません。

- BCPレジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。
- IOENビットをセット（1）した状態で内蔵命令RAM（ライト・モード時）にアクセスした場合，$\overline{\text{IOWR}}$信号がアクティブになります。

図4－26　バス・サイクル・ピリオド・コントロール・レジスタ（BCP）の設定

アドレス：FFFFF48CH

	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	0	0
ビット名	0	0	0	0	IOEN	0	0	0
設定値	0	0	0	0	0	0	0	0

ビット位置	ビット名	意　味
3	IOEN	SRAM，外部ROM，外部I/Oサイクルにおける$\overline{\text{IORD}}$, $\overline{\text{IOWR}}$動作の許可／禁止の設定 0：禁止

(6) プログラム例

```
[file:initial.s]
#########################################################
#    バス制御端子を指定
#########################################################
     mov    0x0003,   r6         -- 制御端子選択
     st.h   r6,       PMCAL[r0]
     mov    0x03,     r6         -- A0-A1出力端子選択
     st.b   r6,       PFCALL[r0]

     mov    0x03ff,   r6         -- A16-A25出力端子選択
     st.h   r6,       PMCAH[r0]

     mov    0xff,     r6         -- CS0-CS7出力信号のインアクティブ・レベル・ハイを設定
     st.b   r6,       PCS[r0]
     mov    0xff,     r6         -- CS0-CS7端子選択
     st.b   r6,       PMCCS[r0]

     mov    0xbf,     r6         -- BCYST,WE/WR,RD,
                                 -- UUWR/UUBE/UUDQM,
                                 -- ULWR/ULBE/ULDQM,
                                 -- LUWR/LUBE/LUDQM,
                                 -- LLWR/LLBE/LLDQM出力信号の
     st.b   r6,       PCT[r0]    -- インアクティブ・レベル・ハイ設定
     mov    0xbf,     r6         -- BCYST,ASTB,WE/WR,RD,
                                 -- UUWR/UUBE/UUDQM,
                                 -- ULWR/ULBE/ULDQM,
                                 -- LUWR/LUBE/LUDQM,
                                 -- LLWR/LLBE/LLDQM出力端子選択
     st.b   r6,       PMCCT[r0]

     mov    0xC0,     r6         -- PCM5-PCM0は出力モード
     st.b   r6,       PMCM[r0]
     mov    0x00,     r6         -- PCM5-PCM0はすべて汎用ポート
     st.b   r6,       PMCCM[r0]

     mov    0x0c,   r6           -- SDRAS,SDCAS出力信号のインアクティブ・レベル・ハイ
     st.b   r6,     PCD[r0]      -- BUSCLK,SDCKE出力信号のインアクティブ・レベル・ロウ設定
     mov    0x0f,   r6           -- SDRAS,SDCAS,BUSCLK,SDCKE出力端子選択
     st.b   r6,     PMCCD[r0]

     mov    0x00,   r6           -- SRAM,外部ROM,外部I/Oサイクル時の
                                 -- IORD,IOWR動作禁止
     st.b   r6,     BCP[r0]      -- バス・サイクル・ピリオド・コントロール・レジスタ
```

4.2.5　汎用入出力ポート兼用端子の初期化例

バス制御機能兼用端子の初期化に加えて，汎用入出力ポート兼用端子の初期化を行います。

まず，汎用入出力ポート兼用端子の設定方法について基本的な考え方を記述します。汎用入出力ポート端子には，汎用入出力ポート機能以外に，割り込み機能，バス制御機能以外の制御機能と兼用になっています。兼用機能の指定は，レジスタ設定により行います。

表4－8　ポート機能／バス制御機能以外の制御機能／割り込み機能を持つ兼用端子の設定

レジスタ / ポート名	PMCn	PMn	PFCn	INTRn	INTFn
ポートn	制御機能	設定無効	バス制御機能以外の制御機能	設定無効	
			割り込み機能	立ち下がりエッジ	
				立ち上がりエッジ	
				レベル検出（H/L検出）	
				立ち上がり／立ち下がり両エッジ	
	ポート機能	入力	設定無効		
		出力			

備考1.　H：ハイ・レベル入力

L：ロウ・レベル入力

2.　n = 1, 2, 5-7（n = 7のとき，INTR7, INTF7レジスタはなし）

次に，バス制御機能以外の制御機能兼用端子の初期化時に注意するポイントを示します。

- ポート・モードからコントロール・モードへの切り替え手順
- ビット操作命令でポート操作を行う場合の注意
- 割り込みトリガ・モード設定時の注意

(1) ポート・モードからコントロール・モードへの切り替え手順

コントロール・モード時に出力または入出力端子として動作するポートをポート・モードからコントロール・モードに切り替える場合は，必ず次に示す手順で設定してください。

① コントロール・モードで出力する信号のインアクティブ・レベルをポートnの該当するビットに設定します（n = 1, 2, 5-7）。

② ポートnモード・コントロール・レジスタ（PMCn）により，コントロール・モードに切り替えます。

注意　上記①を行わない場合は，ポート・モードからコントロール・モードに切り替える際にポートnの内容が一瞬出力されることがあります（n = 1, 2, 5-7）。

(2) ビット操作命令でポート操作を行う場合の注意

ビット操作命令（SET1, CLR1, NOT1）でポート操作を行う場合，ポートに対してバイト・データ・リードを行い，操作対象のビットのみデータの加工を行い，変換後のバイト・データをポートに書き戻します（リード・モディファイ・ライト）。たとえば，入力／出力が混在しているポートでは，操作対象ビット以外のビットにも出力ラッチの内容が上書きされるため，入力端子の出力ラッチは不定になります（ただし，入力モードの場合，出力バッファがオフしているため，端子状態は変化しません）。したがって，ポートを入力から出力に切り替える場合は，ポートnレジスタ（Pn）の該当するビットに出力期待値を設定してから，出力ポートに切り替えてください（n = 1, 2, 5-7）。また，コントロール・モードと出力ポートが混在する場合も同様です。

(3) 割り込みトリガ・モード設定時の注意

トリガ・モードを設定する場合は，PMCnレジスタを設定したあとで行ってください（n = 1, 2, 5, 6）。

INTRn, INTFnレジスタを設定したあとでPMCnレジスタの設定を行うと，PMCnレジスタの設定タイミングで不正な割り込みが発生することがあります。

(4) プログラム例

```
[file:initial.s]
#########################################################
#    バス制御端子以外の機能を指定
#########################################################

# 外部割り込み立ち上がりエッジ指定レジスタ1(INTR1)は初期値(00H)
# 外部割り込み立ち下がりエッジ指定レジスタ1(INTF1)は初期値(00H)
                                --INTP11,INTP10端子は立ち下がりエッジ入力
# 外部割り込み立ち上がりエッジ指定レジスタ2(INTR2)は初期値(00H)
# 外部割り込み立ち下がりエッジ指定レジスタ2(INTF2)は初期値(00H)
                                --INTP2n(n=1-5),NMI端子は立ち下がりエッジ入力
# 外部割り込み立ち上がりエッジ指定レジスタ6(INTR6)は初期値(20H)
# 外部割り込み立ち下がりエッジ指定レジスタ6(INTF6)は初期値(20H)
                                --INTP65端子は立ち上がり／立ち下がり両エッジ入力

    st.b    r0,         P1[r0]      -- TxD0出力信号のインアクティブ・レベル・ロウを設定
    mov     0x0f,       r6          -- P13-10はすべて制御端子モード
    st.b    r6,         PMC1[r0]
    mov     0x0d,       r6          -- P13=TxD0出力,P12=RxD0入力,
                                    -- P11=INTP11入力,P10=UCLK入力
    st.b    r6,         PFC1[r0]

    mov     0x03,       r6          -- P25-P24(LED)のインアクティブ・レベル・ロウ(消灯)
    st.b    r6,         P2[r0]      -- P23(USB制御),TxD1端子にインアクティブ・レベル・ロウ
    mov     0xc7,       r6
    st.b    r6,         PM2[r0]     -- P25-P24(LED),P23(USB制御)出力モード
    mov     0x07,       r6
    st.b    r6,         PMC2[r0]    -- P25-P23汎用ポート,P22-P20は制御端子
    mov     0x06,       r6
    st.b    r6,         PFC2[r0]    -- P22=TxD1,P21=RxD1,P20=NMI

    mov     0x00,       r6          -- P55-P50(LED)のインアクティブ・レベル・ロウ(消灯)
    st.b    r6,         P5[r0]
    mov     0xc0,       r6
    st.b    r6,         PM5[r0]     -- P55-P50(LED)出力モード
    st.b    r0,         PMC5[r0]    -- P55-P50すべて汎用ポート

    mov     0xff,       r6
    st.b    r6,         PM6[r0]     -- P67-P66(TOGLE SW),P65(USB給電監視)入力モード
    mov     0x20,       r6
    st.b    r6,         INTR6[r0]
    st.b    r6,         INTF6[r0]
    st.b    r6,         PMC6[r0]    -- P67-P66汎用ポート,P65=INTP65

    mov     0xff,       r6
    st.b    r6,         P7[r0]
    st.b    r6,         PM7[r0]     -- P77-P72(TOGLE SW)入力モード
    st.b    r0,         PMC7[r0]    -- P77-P72すべて汎用ポート
```

4. 2. 6　クロックの初期化例

CPU内部，および外部動作に関わるクロックの初期化を行います。

4. 2. 3から4. 2. 5までの初期化終了後，LOCKRレジスタのLOCKビットがクリア（0）されている（PLLがロックされている）ことを確認したあと，次に示す手順で設定します。

（i）システム・ウエイト・コントロール・レジスタ（VSWC）一時設定

x7Hに設定（x：最初にVSWCで設定した値の上位4ビット）

（ii）バス・モード・コントロール・レジスタ（BMC）設定

外部バスの周波数分周値設定

（iii）システム・ウエイト・コントロール・レジスタ（VSWC）書き戻し

最初にVSWCで設定した値を書き戻し

（iv）クロック・コントロール・レジスタ（CKC）設定

内部システム・クロック周波数分周値設定

（v）クロック・ソース選択レジスタ（CKS）設定

CKSレジスタのCKSSELビットをセット（1）し，CPUへのクロック供給（メイン・クロック（f_X））をX1, X2端子への入力周波数からPLLにより8逓倍された周波数（OSC出力からSSCG出力）に切り替え

備考　CKC, CKSレジスタは，特定レジスタなので，専用の書き換えシーケンスに従って設定します。

（1）バス・モード・コントロール・レジスタ（BMC）の初期化のポイント

上記（ii）の初期化シーケンスに従って設定します。

図4－27　バス・モード・コントロール・レジスタ（BMC）の設定

アドレス：FFFFF498H

	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	0	0
ビット名	0	0	0	0	0	0	CKM1	CKM0
設定値	0	0	0	0	0	0	0	1

ビット位置	ビット名	意　味
1, 0	CKM1, CKM0	内部システム・クロック（f_{CLK}）に対するバス・クロック（BUSCLK）の分周比を指定 01：f_{CLK}/2

(2) クロック・コントロール・レジスタ（CKC）の初期化のポイント

- **特定シーケンスによってのみ書き込み可能です。**
- **内部システム・クロック（f_{CLK}）を変更した場合，バス・クロック（BUSCLK）の周波数も変更されますので注意してください。**

図4－28　クロック・コントロール・レジスタ（CKC）の設定

アドレス：FFFFF822H

	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	1	1
ビット名	0	0	0	0	0	0	CKDIV1	CKDIV0
設定値	0	0	0	0	0	0	1	1

ビット位置	ビット名	意　味
1, 0	CKDIV1, CKDIV0	PLLモード時の内部システム・クロック（f_{CLK}）を指定 11：f_X

備考　f_X：メイン・クロック

(3) クロック・ソース選択レジスタ（CKS）の初期化のポイント

- **特定シーケンスによってのみ書き込み可能です。**
- **V850E/ME2では，メイン・クロック供給を常時OSC出力（CKSSELビット = 0）の状態でCPUを動作させることは想定していません。このため，必ず初期化シーケンスにおいて，LOCKRレジスタのLOCKビット = 0であることを確認してからメイン・クロック供給をSSCG出力（CKSSELビット = 1）に切り替えてください。この制御以外で設定した場合の動作は保証しません。**

図4－29　クロック・ソース選択レジスタ（CKS）の設定

アドレス：FFFFF82CH

	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	0	0
ビット名	0	0	0	0	0	0	0	CKSSEL
設定値	0	0	0	0	0	0	0	1

ビット位置	ビット名	意　味
0	CKSSEL	メイン・クロック（f_X）供給を指定 1：SSCG出力クロック（F_X×8）

(4) SSCGコントロール・レジスタ（SSCGC）の初期化のポイント

- **特定シーケンスによってのみ書き込み可能です。**
- **SSCGCレジスタは，メイン・クロック供給がOSC出力（CKSレジスタのCKSSELビット = 0）のときのみ設定できます。また，SSCGCレジスタを変更した場合，SSCGはアンロック状態（LOCKRレジスタのLOCKビット = 1）になりますので，必ずLOCKビット = 0であることを確認してからメイン・クロック供給をSSCG出力 （CKSSELビット = 1）に切り替えてください。この制御以外で設定した場合の動作は保証しません。**
- **X1, X2端子への入力周波数（Fx）×8 = fxの値に応じて，PLLSEL, SSEL1, SSEL0端子を次のように設定してください。**

表4－9　周波数一覧

逓倍数	PLLSEL端子	SSEL1端子	SSEL0端子	入力周波数（MHz）（目標値）	メイン・クロック（fx）周波数（MHz）
8	1	0	1	設定禁止	設定禁止
		1	0	10.00～10.19	80.00～81.59
		1	1	10.20～11.99	81.60～95.99
	0	0	0	12.00～14.39	96.00～115.19
		0	1	14.40～17.39	115.20～139.19
		1	0	17.40～18.75	139.20～150.00
		1	1	設定禁止	設定禁止

注意　f_{CLK}**の**MAX.**値は**100 MHz**品の場合は**100 MHz, 133 MHz**品の場合は**133 MHz**，**150 MHz**品の場合は**150 MHz**です。**

f_{CLK}**（**MAX.**）＜**f_X**となる構成で使用した場合の動作は保証しません。**

f_X**は，各製品の動作保証周波数を越えない値にしてください。**

このプログラム例では，f_X = 133.33 MHz**なので，**PLLSEL**端子** = 0**，**SSEL1**端子** = 0**，**SSEL0**端子** = 1**に設定しています。**

図4－30　SSCGコントロール・レジスタ（SSCGC）の設定

アドレス：FFFFF836H

	7	6	5	4	3	2	1	0
初期値	0	0	0	1	注	注	注	注
ビット名	0	0	SMDL1	SMDL0	ADJON	ADJ2	ADJ1	ADJ0
設定値	0	0	0	0	0	0	0	0

ビット位置	ビット名	意　味
5, 4	SMDL1, SMDL0	SSCG出力の変調周期を指定 00：13～27 kHZ
3-0	ADJON, ADJ2-ADJ0	SSCG出力の周波数変調率を指定 0000：変調率なし（周波数固定）

注　ADJON, ADJ2-ADJ0ビットの初期値は，JIT1, JIT0端子の設定により表4－10のようになります。

ADJON, ADJ2-ADJ0ビットはJIT1, JIT0端子の設定により次のようになります。

表4－10　JIT1, JIT0端子の設定によるADJON, ADJ2-ADJ0ビットの初期値

JIT1端子	JIT0端子	初期値			
		ADJONビット	ADJ2ビット	ADJ1ビット	ADJ0ビット
0	0	0	0	0	0
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	1

このプログラム例では，SSCG出力の周波数変調率固定とする，JIT1, JIT0端子 = 00と設定しています。

(5) ロック・レジスタ（LOCKR）の初期化のポイント

ロックするとLOCKフラグは0になります。そのあとにスタンバイ状態，SSCGCレジスタへの書き込み，$\overline{\text{RESET}}$端子入力によりアンロック状態になった場合，LOCKフラグは1になります。しかし，それ以外の原因でアンロック状態になった場合は，LOCKフラグは変化しません（LOCKフラグ = 0の状態）。

図4－31　ロック・レジスタ（LOCKR）の設定

アドレス：FFFFF824H

	7	6	5	4	3	2	1	0
初期値	0	0	0	0	0	0	0	1
ビット名	0	0	0	0	0	0	0	LOCK
設定値	0	0	0	0	0	0	0	－

ビット位置	ビット名	意　味
0	LOCK	PLLのロック待ち状態を示す（読み出し専用フラグ）。 ロックアップ状態を維持しているかぎり0を保持。 0：ロック状態であることを示す 1：ロック待ち（アンロック）状態を示す

(6) プログラム例

```
[file:initial.s]
#########################################################
# IIロック・レジスタ(LOCKR)のLOCKビットの確認
#########################################################
-- ( 0 )ロック・レジスタ(LOCKR)を読み出してPLLの安定を確認
_UNLOCK:
     ld.b   LOCKR[r0],r6        -- ロック・レジスタ(LOCKR)よりPLLのロック状態を読み出し
     cmp    r0,       r6        -- まだフェーズ・ロックしてない？
     jnz    _UNLOCK             -- はい

                                -- PLLロック状態が確認できたので，次の手順へ

-- ( i )システム・ウエイト・コントロール・レジスタ(VSWC)へx7Hを設定(x:Iで設定した値を設定)
     mov    0x37,     r6        -- VSWC設定値上位4bitと下位4bitは7H(fx=133MHz)
     st.b   r6,       VSWC[r0]  -- システム・ウエイト・コントロール・レジスタに書き込み

-- ( ii)バス・モード・コントロール・レジスタ(BMC)へ外部バスの周波数分周値設定
     mov    0x01,     r6        -- 外部システム・クロック(fCLK)に対する
                                -- バス・クロック(BUSCLK)の分周比はfCLK/2
     st.b   r6,       BMC[r0]   -- バス・モード・コントロール・レジスタに書き込み

-- (iii)システム・ウエイト・コントロール・レジスタ(VSWC)へIで設定した値に再度設定
     mov    0x33,     r6        -- VSWC設定値の上位,下位を書き戻す(fx=133MHz)
     st.b   r6,       VSWC[r0]  -- システム・ウエイト・コントロール・レジスタに書き込み

-- ( iv)クロック・コントロール・レジスタ(CKC)へ内部システム・クロック周波数分周値設定
####    特定シーケンスによってのみ書き込み    ####
     mov    0xa0,     r6        -- ①NMI割り込み禁止
     ldsr   r6,       5         --   PSWのNP bitを1に設定(NMI禁止)
     mov    0x03,     r6        -- ②汎用レジスタにCKCに設定するデータを用意
                                --   PLLモード時の内部システム・クロック fCLK=fx
     st.b   r6,       PRCMD[r0] -- ③コマンド・レジスタにダミー・データを書き込み
     st.b   r6,       CKC[r0]   -- ④クロック・コントロール・レジスタを設定
     nop                        -- ⑤ダミーNOP命令挿入(5命令発行)
     nop                        -- ⑥ダミーNOP命令挿入
     nop                        -- ⑦ダミーNOP命令挿入
     nop                        -- ⑧ダミーNOP命令挿入
     nop                        -- ⑨ダミーNOP命令挿入
     mov    0x20,     r6        -- ⑩NMI割り込み禁止を解除(NMI解除)
     ldsr   r6,       5         --   PSWのNP bitを0に戻す
####    特定シーケンスによってのみ書き込み    ####
```

```
-- ( v )クロック・ソース選択レジスタ(CKS)へメイン・クロック(fx)供給制御を設定
####    特定シーケンスによってのみ書き込み    ####
     mov    0xa0,      r6          -- ①NMI割り込み禁止
     ldsr   r6,        5           --   PSWのNP bitを1に設定(NMI禁止)
     mov    0x01,      r6          -- ②汎用レジスタにCKSに設定するデータを用意
                                   --   メイン・クロック(fx)供給をSSCG出力クロック(Fx×8)
     st.b   r6,        PRCMD[r0]-- ③コマンド・レジスタにダミー・データを書き込み
     st.b   r6,        CKS[r0]  -- ④クロック・ソース選択レジスタを設定
     nop                        -- ⑤ダミーNOP命令挿入(5命令発行)
     nop                        -- ⑥ダミーNOP命令挿入
     nop                        -- ⑦ダミーNOP命令挿入
     nop                        -- ⑧ダミーNOP命令挿入
     nop                        -- ⑨ダミーNOP命令挿入
     mov    0x20,      r6          -- ⑩NMI割り込み禁止を解除(NMI解除)
     ldsr   r6,        5           --   PSWのNP bitを0に戻す
####    特定シーケンスによってのみ書き込み    ####

# SSCGコントロール・レジスタ(SSCGC)は初期値(4xH:xについては下記参照)
####    特定シーケンスによってのみ書き込み    ####
--SMDL1=0,SMDL1=1:SSCG出力の変調周期26~35KHz
--JIT1端子 JIT0端子 ADJON ADJ2 ADJ1 ADJ0  SSCG出力の周波数変調率(Typ.値)
--   0      0        0    0    0    0          変調なし(周波数固定)
--   0      1        1    0    0    1          -1.0%
--   1      0        1    0    1    1          -3.0%
--   1      1        1    1    0    1          -5.0%
####    特定シーケンスによってのみ書き込み    ####

# USBクロック・コントロール・レジスタ(UCKC)
#    mov    0x80,      r6          -- USBへのクロック供給開始
#    st.b   r6,        UCKC[r0]

# 発振安定時間選択レジスタ(OSTS)は初期値(04H)
--OSTS0=0 ソフトウエアSTOPモード解除後の発振安定時間21.84ms(Fx=12MHz時)

_UNLOCK2:
     ld.b   LOCKR[r0],r6        -- ロック・レジスタ(LOCKR)よりPLLのロック状態を読み出し
     cmp    r0,        r6          -- まだフェーズ・ロックしてない？
     jnz    _UNLOCK2               -- はい

                                   -- PLLロック状態が確認できたので，次の手順へ
```

4. 2. 7　SDRAMの初期化例

SDRAMの初期化を行います。

SDRAMの初期化のポイントとプログラム例を次に示します。

（1）SDRAMを使用する場合は，必ず次に示す初期化シーケンスに従ってください。

- ポートaモード・コントロール・レジスタ（PMCa）を設定する（a = AL, AH, DH, CS, CT, CM, CD）。
- チップ・エリア選択コントロール・レジスタm（CSCm）を設定し，SDRAMを接続するチップ・セレクト信号を確定する（m = 0, 1）。
- バス・サイクル・タイプ・コンフィギュレーション・レジスタm（BCTm）の設定により，SDRAMを接続するチップ・セレクト空間のメモリの種類を確定し，メモリ・コントローラを動作許可にする（m = 0, 1）。
- バス・サイクル・コントロール・レジスタ（BCC）により，アイドル・ステートの挿入を設定する。
- SDRAMリフレッシュ・コントロール・レジスタn（RFSn）を設定する（n = 1, 3, 4, 6）。
- SDRAMコンフィギュレーション・レジスタn（SCRn）を設定する（n = 1, 3, 4, 6）。

（2）SDRAMコンフィギュレーション・レジスタn（SCRn）（n = 1, 3, 4, 6）

- レジスタ・ライト・オペレーションの実行前は，SDRAMリード／ライト・サイクルは発生しません。SCRnレジスタを設定後，SCRnレジスタの値をリードし，WCFnビットがセット（1）されていることを確認してから，SDRAM領域をアクセスしてください。
- SCRnレジスタに書き込む命令を連続して実行しないでください。SCRnレジスタに書き込む命令と命令の間には必ずほかの命令を入れて実行してください。
- レジスタへの書き込みはリセット後に行ってください。書き込み後は値を変更しないでください。

(3) SDRAMリフレッシュ・コントロール・レジスタn（RFSn）（n = 1, 3, 4, 6）

RFSnレジスタへの書き込みはリセット後に行い，そのあとは値を変更しないでください。ただし，CKCレジスタの設定値（内部システム・クロック（f_{CLK}））を変更することで，SDRAMのリフレッシュ間隔も変更する必要がある場合のみ，RFSnレジスタへの設定値を変更できます[注]。また，RFSnレジスタの初期設定が終わるまでは，その初期化ルーチン以外の外部メモリ領域をアクセスしないでください。ただし，初期設定が終了した外部メモリ領域のアクセスは可能です。

注　内部システム・クロック（f_{CLK}）の変更により，SDRAMのリフレッシュ間隔を変更する必要がある場合は，次の手順で行ってください。

①すべての割り込みをマスクする。
マスカブル割り込みに対する割り込み禁止については，割り込みマスク・レジスタ0-5（IMR0-IMR5）（V850E/ME2　**ユーザーズ・マニュアル　ハードウエア編**　7. 3. 5　**割り込みマスク・レジスタ**0-5（IMR0-IMR5）参照）を設定してください。ノンマスカブル割り込みに対する割り込み禁止については，PSWのNPビットをセット（1）して，多重割り込み禁止の状態にしてください（V850E/ME2　**ユーザーズ・マニュアル　ハードウエア編**　3. 2. 2（4）**プログラム・ステータス・ワード**（PSW）参照）。
②BCTmレジスタのMEaビットをクリア（0）する（m = 0, 1，a = 0-7）。
③RENnビットをクリア（0）する（n = 1, 3, 4, 6）。
④BCTmレジスタのMEaビットをセット（1）する（m = 0, 1，a = 0-7）。
⑤RCCn1, RCCn0, RINn5-RINn0ビットに新たな値を設定するとともにRENnビットをセット（1）する（n = 1, 3, 4, 6）。
⑥SCRnレジスタに，現在SCRnレジスタに設定されている値と同じ値を書き込む（n = 1, 3, 4, 6）。
⑦SCRnレジスタのWCFnビットがセット（1）状態であることを確認後にSDRAMアクセスを行う（n = 1, 3, 4, 6）。

なお，リフレッシュ間隔を切り替える場合，切り替えの間でもリフレッシュが十分間に合うような値を考慮して設定してください。リフレッシュ間隔が確実に確保される場合には，上記①の処理を削除できます。また，RFSn, BCTmレジスタは，再書き込み禁止ですが，CKCレジスタの変更によるリフレッシュ間隔の再設定の場合のみ再書き込み可能です。
また，SCRnレジスタの再書き込みにより（上記⑥の処理），SDRAMのレジスタ・ライト・サイクルが発生しますが，SDRAM上のデータは，RFSn, SCRnレジスタの再設定を行う前の値が保持されます。

(4) プログラム例

```
[file:initial.s]
#########################################################
#   SDRAMコンフィギュレーションの指定
#########################################################

### CS1(SDRAM 32bit幅/32MB)空間のリフレッシュ・インターバル設定 ##
     mov    0x801e,   r6         -- リフレッシュ許可(BUSCLK=66MHz)
                                 -- リフレッシュ・カウント・クロック=32/BUSCLK
                                 -- リフレッシュ・インターバル・ファクタ=31(2.000MHz時15.0us)
     st.h   r6,       RFS1[r0]  -- SDRAMリフレッシュ・コントロール・レジスタ1

### CS1(SDRAM 32bit幅/32MB)空間のリフレッシュ環境設定 ##
     mov    0x20a5,   r6         -- リード時CAS Latency 2 Latency(TLATE)
                                 -- ACT-RD,PRE-ACT時に2 wait(TBCW)
                                 -- アドレス・シフト幅         2 bit(On-page)
                                 --                          (外部データ・バス幅:32bit)
                                 -- ロウ・アドレス幅           12 bit
                                 -- アドレス・マルチプレクス幅 9 bit
     st.h   r6,       SCR1[r0]  -- SDRAMコンフィギュレーション・レジスタ1に書き込み
_SCR1NOFIX:
     ld.h   SCR1[r0], r6         -- SDRAMコンフィギュレーション・レジスタ1を読み出し
     movea  0x0100,   r0,  r7    -- WCF1 bit抽出
     and    r7,       r6
     cmp    r6,       r7         -- SCR1設定未完了?
     jnz    _SCR1NOFIX           -- はい

### CS4(SDRAM 16bit幅/16MB)空間のリフレッシュ・インターバル設定 ##
     mov    0x801e,   r6         -- リフレッシュ許可(BUSCLK=66MHz)
                                 -- リフレッシュ・カウント・クロック=32/BUSCLK
                                 -- リフレッシュ・インターバル・ファクタ=31(2.000MHz時15.0us)
     st.h   r6,       RFS4[r0]  -- SDRAMリフレッシュ・コントロール・レジスタ4

### CS4(SDRAM 16bit幅/16MB)空間のリフレッシュ環境設定 ##
     mov    0x2095,   r6         -- リード時CAS Latency 2 Latency(TLATE)
                                 -- ACT-RD,PRE-ACT時に2 wait(TBCW)
                                 -- アドレス・シフト幅         1 bit(On-page)
                                 --                          (外部データ・バス幅:16bit)
                                 -- ロウ・アドレス幅           12 bit
                                 -- アドレス・マルチプレクス幅 9 bit
     st.h   r6,       SCR4[r0]  -- SDRAMコンフィギュレーション・レジスタ4
_SCR4NOFIX:
     ld.h   SCR4[r0], r6         -- SDRAMコンフィギュレーション・レジスタ4を読み出し
     movea  0x0100,   r0,  r7    -- WCF4 bit抽出
     and    r7,       r6
     cmp    r6,       r7         -- SCR4設定未完了?
     jnz    _SCR4NOFIX           -- はい
```

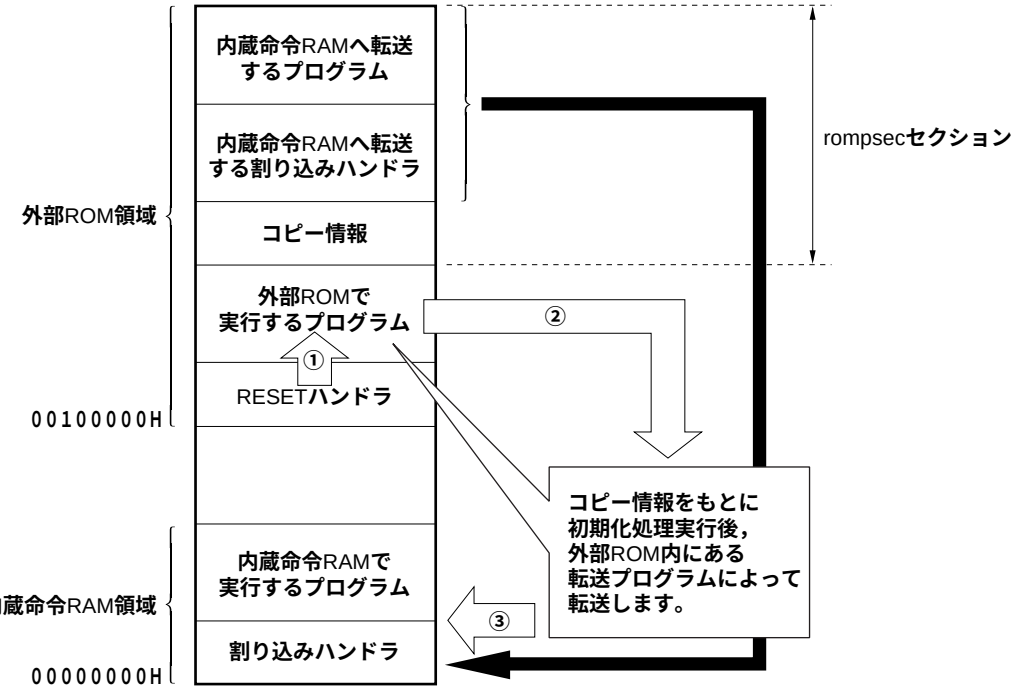
```
### CS6(SDRAM  8bit幅/ 16MB)空間のリフレッシュ・インターバル設定 ##
     mov    0x801e,   r6         -- リフレッシュ許可(BUSCLK=66MHz)
                                 -- リフレッシュ・カウント・クロック=32/BUSCLK
                                 -- リフレッシュ・インターバル・ファクタ=31(2.000MHz時15.0us)
     st.h   r6,       RFS6[r0]  -- SDRAMリフレッシュ・コントロール・レジスタ6

### CS6(SDRAM  8bit幅/ 16MB)空間のリフレッシュ環境設定 ##
     mov    0x2086,   r6         -- リード時CAS Latency 2 Latency(TLATE)
                                 -- ACT-RD,PRE-ACT時に2 wait(TBCW)
                                 -- アドレス・シフト幅          0 bit(On-page)
                                 --                           (外部データ・バス幅: 8bit)
                                 -- ロウ・アドレス幅            12 bit
                                 -- アドレス・マルチプレクス幅  10 bit
     st.h   r6,       SCR6[r0]  -- SDRAMコンフィギュレーション・レジスタ6
_SCR6NOFIX:
     ld.h   SCR6[r0], r6         -- SDRAMコンフィギュレーション・レジスタ6を読み出し
     movea  0x0100,   r0,  r7    -- WCF6 bit抽出
     and    r7,       r6
     cmp    r6,       r7         -- SCR6設定未完了?
     jnz    _SCR6NOFIX           -- はい
```

4. 2. 8　内蔵命令RAMの転送例

CPU初期化処理終了後に実行するプログラムを内蔵命令RAMへ転送します。

内蔵命令RAM転送の基本的な考え方を次に示します。

図4－32　内蔵命令RAM転送の基本的な考え方



備考　図中の①-③は処理の流れを示します。

内蔵命令RAM転送時に注意するポイントを次に示します。

- 転送前後のノンマスカブル割り込み（NMI）および例外割り込みの取り扱いについて
- RESETハンドラのセグメント・ディレクティブ記述について
- 内蔵命令RAMへ転送するプログラムの分離について
- ROM化プロッセッサのオプションについて

(1) 転送前後のノンマスカブル割り込み（NMI）および例外割り込みの取り扱いについて

V850E/ME2では，初期化後にプログラムを内蔵命令RAMへ転送することを想定しています。内蔵命令RAM転送実行中にNMIを含めたあらゆる割り込み，例外の発生は禁止です。このため，通常マスク不可能な割り込みであるNMIに対する仕組みと,プログラム実行によって発生する例外について留意するべきことがあります。

- リセット解除後，NMI入力はハードウエア的にマスクされます。内蔵命令RAM転送シーケンスにおいて，内蔵命令RAMモード・レジスタ（IRAMM）のIRAMM0ビットのクリア（0）と同時にNMIのマスクは解除されます。
- リセット解除後から，内蔵命令RAMをリード・モードに切り替えるまでの間に，NMIが入力されたことを確認する必要がある場合は，NMIリセット・ステータス・レジスタ（NRS）のNMIRSビットをリードしてください。セット（1）されていれば，NMIの有効エッジ入力があったことを示すので，必要に応じてNMIの割り込み処理ルーチンを実行してください。なお，NRSレジスタは，リセット解除後から内蔵命令RAMをリード・モードに設定するまでの間のNMI入力の確認のみに使用され，リセット解除後にクリア（0）されません。

(2) RESETハンドラのセグメント・ディレクティブ記述について

一般にV850シリーズのCPUでは，RESETハンドラ・アドレスが00000000H番地となっています。00000010H番地以降から上位アドレス方向へは，一部を除きほぼ16バイト間隔で，ノンマスカブル割り込み（NMI），例外，マスカブル割り込みハンドラ領域が存在しています。この領域は，内蔵フラッシュ・メモリ，内蔵マスクROM，または外部ROM領域です。これらはROMなので不揮発メモリです。

内蔵命令RAMを採用しても，他のV850シリーズのCPUとの互換性が必要なので，00000010H番地以降から上位アドレス方向へは，一部を除きほぼ16バイト間隔で，NMI，例外，マスカブル割り込みハンドラ領域が存在しています。この領域はRAMなので揮発メモリです。RESETハンドラ・アドレスを揮発メモリである内蔵命令RAM空間の00000000H番地にすると，命令が不定になってしまいプログラムが正常に動きません。RESETハンドラは外部の不揮発メモリ（EPROM, フラッシュ・メモリなど）へ配置する必要があるため，V850E/ME2ではRESETハンドラ・アドレスが外部メモリ空間の00100000H番地になっています。

通常，次のようなアセンブリ・ソース記述をすると，CA850コンパイラ・パッケージがリンク解決する際，内部的なRESETセグメント・ディレクティブとともにRESETハンドラ・アドレスである00000000H番地へプログラム（命令）を配置します。

```
.section "RESET", text
jr  _ _start
```

しかし，前述のようにV850E/ME2ではRESETハンドラ・アドレスが00100000H番地です。そのため，V850E/ME2のRESETハンドラの番地を付加したRESETセグメント・ディレクティブをディレクティブ・ファイル中へ明示的に記述します。

```
RESET   : !LOAD ?RX V0x00100000 {
    RESET        = $PROGBITS ?AX RESET;
};
```

これで，正常にRESETハンドラへプログラム（命令）が配置されます。

注意　このRESETセグメント・ディレクティブに番地を記述しない場合，上記アセンブリ・ソース・コードよりも前にリンクされたプログラム・コードのすぐあと，すなわち，そのときのプログラムによって不定な番地へプログラム（命令）配置されてしまうので注意してください。

(3) 内蔵命令RAMへ転送するプログラムの分離について

図4－32より明らかなように，CPU初期化から転送するまでのプログラムは外部ROMで実行されます。内蔵命令RAMへ転送して実行するプログラムもROM化された形で外部メモリに格納されています。CPU初期化から転送するまでのプログラムは外部ROMで実行されますが，rompsecセクションにパッキングする必要はありません。内蔵命令RAMへ転送して実行するプログラムは，外部ROMでは実行しませんが，rompsecセクションにパッキングする必要があります。

プログラム	外部ROMでの実行	rompsecパッキング
CPU初期化から転送するまでのプログラム	する	必要なし
内蔵命令RAMへ転送して実行するプログラム	しない	必要あり

どちらもプログラムなので，text属性を持つセクションです。rompsecセクションは全text属性セクションを対象とします。したがって，この2つの部分をリンク時に分離します。ディレクティブ・ファイルとソース・ファイルに記述の必要があります。また，rompsecセクションのエントリは何も指定しないと.textセクションの最後尾になり，やはり揮発メモリ上に配置されてしまいます。そこで，新たなrompsecセクションのエントリ・ラベルを用意し，不揮発メモリの外部ROMで実行するプログラムの最後尾にディレクティブによって配置します。

(a) ソース・ファイルの記述について

外部メモリのみで実行するプログラム（CPU初期化からプログラム転送まで）に対しては，セクション名指定のソース・ファイル記述を追加します。

【C言語の場合】

```
#pragama text "セクション名"
  ＜外部メモリのみで実行するプログラム群＞
#pragama text

または，

#pragama text "セクション名" 関数名
  ＜外部メモリのみで実行する関数＞
```

【アセンブリ言語の場合】

```
.section  "セクション名", text
  ＜外部メモリのみで実行するプログラム＞
```

内蔵命令RAMへ転送して実行するプログラムに対しては，特に何も記述しません。

romsecセクションの新たなエントリ・ラベルと，そのエントリ・ラベルの新たなセクション名の記述があるアセンブリ言語ソース・ファイルを追加します。

```
.section "新たなセクション名",text
.align    4
.globl    新たなエントリ・ラベル名, 4
新たなエントリ・ラベル名:
```

(b) ディレクティブ記述について

外部メモリのみで実行するすべてのプログラム（CPU初期化からプログラム転送まで）に対して行ったセクション名指定のディレクティブ記述を追加します。

【C言語の場合】

[cソース・ファイル]
#pragama text "セクション名"
　＜外部メモリのみで実行するプログラム群＞
#pragama text

または，

#pragama text "セクション名" 関数名
　＜外部メモリのみで実行する関数＞

に対しては，

[ディレクティブ・ファイル]
任意のセグメント名 : !LOAD ?RX V割り付けたいアドレス {
　セクション名.text　= $PROGBITS ?AX セクション名.text;
};

次にromfuncセクションを前のセグメントへ続けて配置した場合の例を示します。

[cソース・ファイル]
```
#pragama text "romfunc" test_func
void test_func(void){
```
＜外部メモリのみで実行するプログラム＞
```
}
```

[ディレクティブ・ファイル]
```
ROMFUNC    : !LOAD ?RX  {
  romfunc.text       = $PROGBITS ?AX romfunc.text;
};
```

【アセンブリ言語の場合】

[アセンブラ・ソース・ファイル]
```
  .section  "セクション名", text
```
＜外部メモリのみで実行するプログラム＞

に対しては，

[ディレクティブ・ファイル]
```
任意のセグメント名 : !LOAD ?RX V割り付けたいアドレス {
  セクション名       = $PROGBITS ?AX セクション名;
};
```

次に.crtEセクションを00101000H番地へ配置した場合の例を示します。

[アセンブラ・ソース・ファイル]
```
  .section  ".crtE", text
```
＜外部メモリのみで実行するプログラム＞

[ディレクティブ・ファイル]
```
CRTE    : !LOAD ?RX V0x00101000 {
  .crtE       = $PROGBITS ?AX .crtE;
};
```

内蔵命令RAMへ転送して実行するプログラムに対しては特別な記述はありません。

次に示すTEXTセグメントのアドレス指定で，内蔵命令RAMで実行させたいアドレスへ割り付けるだけです。

次に.textセクションを00001000H番地へ配置した場合の例を示します。

```
TEXT        : !LOAD ?RX V0x00001000 {
  .text       = $PROGBITS ?AX .text;
};
```

rompsecセクションの新しいエントリ・ラベルに対してディレクティブ記述を追加します。ディレクティブ・ファイル中で追加する位置は，外部メモリのみで実行するすべてのプログラム（CPU初期化からプログラム転送まで）に対して行ったセクション名指定のディレクティブ記述群の最後です。

[アセンブラ・ソース・ファイル]

```
.section "新たなセクション名",text
.align    4
.globl    新たなエントリ・ラベル名, 4
新たなエントリ・ラベル名:
```

に対しては，

[ディレクティブ・ファイル]

```
任意のセグメント名 : !LOAD ?RX {
  新たなエントリ・ラベル名   = $PROGBITS ?AX 新たなエントリ・ラベル名;
};
```

.s_rompセクションを外部メモリのみで実行するすべてのプログラム（CPU初期化からプログラム転送まで）に対して行ったセクション名指定のディレクティブ記述群の最後（今回の例ではCRTEセグメントの次）に配置した例を次に示します。

[アセンブラ・ソース・ファイル]

```
.section ".s_romp",text
.align    4
.globl    _ _S_rompack, 4
_ _Srompack;
```

[ディレクティブ・ファイル]

```
SROMP : !LOAD ?RX {
  .s_romp   = $PROGBITS ?AX .s_romp;
};
```

(4) ROM化プロセッサのオプションについて

ディレクティブ・ファイルとソース・ファイルの記述によって内蔵命令RAMへ転送されず外部ROMでのみ実行するプログラムと，外部ROMでは実行されず内蔵命令RAMへ転送して実行するプログラムが分離されました。外部ROMでは実行されず内蔵命令RAMへ転送して実行するプログラムはすべてtext属性セクションを持つ予約セクション名.textとなっています。このセクションと使用する割り込みハンドラのセクションをrompsecセクションにパッキングします。「パッキングする読み出し専用セクション・オプション」-tで.textと使用する割り込みハンドラ名を示します。

```
-t .text -t 割り込みハンドラ名1 -t 割り込みハンドラ名2 ・・・
```

外部ROMでは実行されず内蔵命令RAMへ転送して実行するプログラムをrompsecセクションとしてパッキングするために「エントリ・ラベル・オプション」-bで新たなエントリ・ラベルを定義します。

```
-b エントリ・ラベル
```

なお，リンク時に内蔵ROMエリアを越えている旨の警告が出ますが無視します。

(5) ヘキサ・コンバータのオプションについて

外部ROMのみで実行するプログラム（CPU初期化からプログラム転送まで）は，外部ROMまでオフセットをかける必要があります。「出力アドレスのオフセット・オプション」-dで100000Hを指定します。

```
-d 0x100000
```

(6) プログラム例

注意　プログラム（file:crtE.s）中の記述「-- （ 0 ）内蔵命令RAMへProgramを転送する」の詳細については，4. 2. 10（2）（d）内蔵命令RAMのバンク0へ静的プログラムを転送する方法についてを参照してください。

```
[file:crtE.s]
(前略)
# 通常のcrtE.sファイルのclear bss sectionの後ろ

#########################################################
# 内蔵命令RAMにプログラム・コードを転送
#########################################################
    .extern _ _S_rompack
    .extern _ _S_appli1rom
    .extern _ _ircopy
    .globl _iramboot
_iramboot:
-- ( 0 )内蔵命令RAMへProgramを転送する
.option nowarning
    movhi   hi1(_ _S_rompack), tp, r1   -- tp のオフセットをかける
    movea   lo(_ _S_rompack), r1, r6    -- 静的プログラム・パッキング・セクション先頭アドレス
.option warning
    mov     -0x1,   r7                  -- 転送したいパッキング・セクションはすべて
    jarl    _ _ircopy,r31               -- ROM化された動的プログラムをBANK0へ転送

-- ( i )内蔵命令RAMモード・レジスタ(IRAMM)をリード・モード設定(IRAMM0,1=0)
    mov     0x00,   r6                  -- BANK0 0000000-00FFFFH(64KB)リード・モード
                                        -- BANK1 0010000-01FFFFH(64KB)リード・モード
    st.b    r6,     IRAMM[r0]           -- 内蔵命令RAMモード・レジスタ

-- ( ii )内蔵命令RAMモード・レジスタ(IRAMM)のリード・モード変更を確認(IRAMM0=1?)
_iram_rd_chk:
    ld.b    IRAMM[r0],r6                -- 内蔵命令RAMモード・レジスタ読み出し
    cmp     r0,     r6                  -- BANK0,1はリード・モード?
    bnz     _iram_rd_chk                -- いいえ

-- (iii)分岐命令により内蔵命令RAMへ
.option nowarning
    mov     #_main, r1
    jmp     [r1]
.option warning
_ _exit:
    halt                                -- end of program
_ _startend:
    nop
    nop

#                                                       #
#-------------------- end of start up module -----------#
#                                                       #
```

```
 [file:ircopy.s]
################################################################################
#
# 【関数名】_ircopy
#
# 【概　要】内蔵命令RAMへ外部ROMからプログラムを転送する関数
#
# 【説　明】_ircopy関数は，第1引数r6で渡されるアドレス以降に存在する
#           rompsecセクション内の情報をもとに，第2引数r7で渡されるセクション番号の
#           プログラムを内蔵命令RAM領域へ転送します。
#
# 【引　数】    r6:rompsecセクションの先頭アドレス
#               r7:内蔵命令RAMへコピーしたいセクションID番号
#                  ※セクションID番号は'1'から始まる
#                  ※-1はすべてのセクションをコピー
#
# 【戻り値】   r10:コピー成功(0)/コピー失敗(-1)
#
# 【スタック】未使用
# 【register】r8:コピー開始する1セクション毎コピー情報の単位目
#             r9:コピー終了する1セクション毎コピー情報の単位目
#             r10:転送サイズ/一時変数
#             r11:セクションID番号で指定されたセクションの転送先アドレス/一時変数
#             r12:セクションID番号で指定されたセクションの転送元アドレス/一時変数
#             r13:1セクション毎のコピー情報が格納されている領域の先頭アドレス
#             r14:セクションID番号で指定されたセクションの転送先開始アドレス
#             r15:セクションID番号で指定されたセクションの転送元開始アドレス
#             r16:一時変数/ID番号で指定されたセクションの転送サイズ
#             r17:一時変数
#             r18:一時変数
#             r19:一時変数
#             r20:一時変数
#             r21:一時変数
################################################################################
#           n個のセクションがパッキングされたrompsecセクションの内容図解
#
# rompsecセクションの先頭
# からのオフセット・アドレス
#    ---     + 0-> -+-----------------------------------------+- rompsecセクション
#     |             | マジック・ナンバ                          |  先頭アドレス
#     |             | (0x00504d2)                             |
# 共通情報    + 4-> -+-----------------------------------------|
#     |             | セクション数                              |
#     |             | (単位byte)                                |
#    ---     + 8-> -+=========================================+- + 0
#     |             | 1番目のセクションの転送先アドレス         |
#     |             | (絶対番地)                                |
# 1セクション +12-> -+-----------------------------------------+- + 4
# 毎に              | 1番目のセクションの転送先アドレス         |
#     |             | (絶対番地)                                |
```

```
# 16byte    +16-> -+-----------------------------------------+- + 8
# のコピー          | 1番目のセクションのサイズ                |
#    |            | (単位byte)                               |
# 情報        +20-> -+-----------------------------------------+- +12
#    |            | 1番目のセクションの転送元アドレス        |
#    |            | (rompsecセクション先頭アドレスからのオフセット番地) |
#   ---     +24-> -+=========================================+- + 0
#    |            | 2番目のセクションの転送先アドレス        |
#    |            | (絶対番地)                               |
# 1セクション +28-> -+-----------------------------------------+- + 4
# 毎に              | 2番目のセクションの転送先アドレス        |
#    |            | (絶対番地)                               |
# 16byte    +32-> -+-----------------------------------------+- + 8
# のコピー          | 2番目のセクションのサイズ                |
#    |            | (単位byte)                               |
# 情報        +36-> -+-----------------------------------------+- +12
#    |            | 2番目のセクションの転送元アドレス        |
#    |            | (rompsecセクション先頭アドレスからのオフセット番地) |
#   ---     +40-> -+=========================================+-
#                  |                    :                    |
#                  |                    :                    |
#   +(n-1)*16+8-> -+=========================================+-
#                  | n番目のセクションの転送先アドレス        |
#                  | (絶対番地)                               |
#                 -+-----------------------------------------+-
#                  | n番目のセクションの転送先アドレス        |
#                  | (絶対番地)                               |
#                 -+-----------------------------------------+- - 8
#                  | nバイトのセクションのサイズ              |
#                  | (単位byte)                               |
#                 -+-----------------------------------------+- - 4
#                  | n番目のセクションの転送元アドレス        |
#                  | (rompsecセクション先頭アドレスからのオフセット番地) |
#       +n*16+8-> -+=========================================+- + 0
#                  | 1番目のセクションのパッキング・データ    |
#                 -+-----------------------------------------+-
#                  |                    :                    |
#                 -+-----------------------------------------+-
#                  | 2番目のセクションのパッキング・データ    |
#                 -+-----------------------------------------+-
#                  |                    :                    |
##############################################################################
    .section ".ircopy",text
    .align   16 --    .align 4
    .globl   _ _ircopy
_ _ircopy:

-- rompsecセクション共通情報の正当性をチェック
    ld.w    0x0[r6],  r15       -- rompsecセクション1WORD目データ取得
.option nowarning
```

```
    movhi   0x50,     r0, r1
    movea   0x4d52,   r1, r1
    cmp     r1,     r15           -- Magic Number(0x504d2)と比較
.option warning
    bnz     _ircopy_ng            -- Magic Numberが一致しないので不正終了
    ld.w    0x4[r6],r12           -- rompsecセクション2WORD目データ取得
    cmp     0x1,    r12           -- rompsecセクション内のパッキング・セクション数は0?
    blt     _section_id_chk       -- コピーするセクションがない

-- 第2引数r7(コピーするセクションID番号)の正当性をチェック
    cmp     -0x1,   r7            -- コピーするセクションID番号は-1?
    bz      _all_copy             -- 全セクションをコピーする
_section_id_chk:
    cmp     0x1,    r7            -- コピーする指定セクションID番号＜1?
    blt     _ircopy_ng            -- コピーする指定セクションID番号<1なので不正終了
    cmp     r7,     r12           -- コピーする指定セクションID番号≦パッキング・セクション数?
    bge     _section_copy         -- パッキング・セクション数越のセクションID番号は
    br      _ircopy_ng            -- コピーできないので不正終了
    nop
    nop

-- ID番号で指定されたセクションをコピーするための初期化
_section_copy:
    addi    -0x1,   r7,    r8     -- 1セクション毎コピー情報<ID-1>単位目から
                                  -- コピーを開始するものとしてr8へ
    mov     r7,     r9            -- 1セクション毎コピー情報<ID>単位目で
                                  -- コピーを終了するものとしてr9へ
                                  -- つまりID番号で指定されたセクションのみを
                                  -- コピーすることになる
    br      _ircopy_main          -- コピー処理へ

-- 全セクションをコピーするための初期化
_all_copy:
    mov     r0,     r8            -- 1セクション毎コピー情報<0>単位目から
                                  -- コピーを開始するものとしてr8へ
    mov     r12,    r9            -- 1セクション毎コピー情報<パッキング・セクション数>単位目で
                                  -- コピーを終了するものとしてr9へ


_ircopy_main:
-- パッキング・セクションが格納されている開始アドレスを算出する
    addi    0x8,    r6,    r11    -- rompsecセクション3WORD目のアドレス取得
                                  -- これは1セクション毎コピー情報0単位目のアドレス[A]

    shl     0x4,    r12           -- パッキング・セクション数×16倍(1セクション毎コピー情報1単位
                                  -- 分)
                                  -- これは1セクション毎コピー情報の総byte数を算出[B]

    add     r11,    r12           -- A+B=rompsecセクションのパッキング・データ先頭アドレス
    cmp     r9,     r8            -- コピー終了する1セクション毎コピー情報の単位目まで到達したか?
    mov     r12,    r10           -- rompsecセクションのパッキング・データ先頭アドレスをr10へ
```

```
    bge     _ircopy_ok          -- 到達したのでコピー正常終了

-- パッキング・セクションが格納されている終了アドレスを算出する
    ld.w    -0x4[r10],r17       -- 1セクション毎コピー情報最終単位目の転送元開始アドレス
                                -- すなわちrompsecセクション先頭からのオフセット値を取得
    add     r6,     r17         -- 上記オフセット値にrompsecセクションの先頭アドレスを加算
                                -- することで転送元開始絶対アドレスを求める
    ld.w    -0x8[r10],r18       -- １セクション毎コピー情報最終単位目のコピー転送サイズを取得
    add     r18,    r17         -- 上記絶対アドレスに転送サイズを加算することで
                                -- 転送元終了絶対アドレスを求める

    mov     r17,    r21         -- パッキング・セクションが格納されている終了アドレスを退避
    mov     r11,    r20         -- 1セクション毎コピー情報0単位目のアドレスを退避

_section_loop:
    mov     r8,     r19         -- コピーするセクションID番号をr19へ
    shl     0x4,    r19         -- セクションID番号×16倍(1セクション毎コピー情報1単位分)
                                -- 指定されたセクションID番号のコピー情報の先頭までの
                                -- 総byte数を算出
    mov     r20,    r13         -- １セクション毎コピー情報0単位目のアドレスを取得
    add     r19,    r13         -- 上記アドレスに1セクション毎コピー情報のn単位分(n*16byte)
                                -- を加算することでセクションID番号nの1セクション毎コピー情報が
                                -- 格納されている絶対アドレスを求める
    ld.w    0x0[r13],r14        -- セクションID番号nの転送先開始アドレスを取得
    mov     r14,    r11         -- 上記絶対アドレスに転送サイズを加算することで
                                -- セクションID番号nの転送先開始絶対アドレスを求める

    ld.w    0xc[r13],r15        -- セクションID番号nの転送元開始アドレスを取得
                                -- すなわちrompsecセクション先頭からのオフセット値を取得
    add     r6,     r15         -- 上記アドレスにrompsecセクションの先頭アドレスを加算すること
                                -- で転送元開始絶対アドレスを求める

-- rompsecセクション先頭アドレスの正当性をチェック
    ld.w    0x8[r13],r16        -- 1セクション毎コピー情報最終単位目のコピー転送サイズを取得
    mov     r16,    r10         -- 上記をr10へ一時退避
    shr     2,      r10         -- [WORD]上記を4(WORD size)で割ってr10へ一時退避
    tst1    0,      0x8[r13]    -- [WORD]転送サイズは奇数?
    bnz     _align4adjust       -- [WORD]はい，4byte境界補正へ
    tst1    1,      0x8[r13]    -- [WORD]いいえ，転送サイズに3以下の端数あり?
    bz      _align4ok           -- [WORD]いいえ，補正の必要なし。
_align4adjust:                  -- [WORD]はい，Align Error発生を回避するため
    add     1,      r10         -- [WORD]転送サイズに+1
_align4ok:                      -- [WORD]4byte境界なので転送サイズそのまま

    add     r16,    r14         -- 指定セクションの転送先開始アドレスに転送サイズを加算することで
                                -- 指定セクションの転送先終了アドレスを算出
    cmp     r6,     r14         -- 指定セクションの転送先終了アドレスとrompsecセクション先頭
                                -- アドレスがオーバラップ?
    mov     r15,    r12         -- 指定セクションの転送元開始アドレスをr12へ
    bnh     _size_chk           -- いいえ，コピーするデータ・サイズのチェックへ
```

```
-- rompsecセクション終了アドレスの正当性をチェック
    mov     r21,    r18             -- 退避していたrompsecセクション終了アドレスを読み出す
    cmp     r18,    r11             -- rompsecセクション終了アドレスと転送先開始アドレスが
                                    -- オーバラップ?
    bc      _ircopy_ng              -- はい，コピーされない危険があるので不正終了
_size_chk:
    cmp     r0,     r10             -- コピーするセクションのサイズ≦0?
    ble     _next_section           -- はい，コピーする必要がないので次のセクションへ
    add     -0x1,   r10             -- セクションID番号で指定されたセクションの転送サイズ

_data_loop:
    mov     r11,    r14
    addi    0x4,    r14,    r11     -- セクションID番号で指定されたセクションの転送先アドレス+4

    mov     r12,    r7
    addi    0x4,    r7,     r12     -- セクションID番号で指定されたセクションの転送元アドレス+4

    ld.w    0x0[r7],r19             --セクションID番号で指定されたセクションのパッキング・データ取得
    st.w    r19,    0x0[r14]        -- 展開先へ格納

    mov     r10,    r13
    addi    -0x1,   r13,    r10     -- カウンタ減算

    cmp     r0,     r13             -- カウンタ＞0?
    bgt     _data_loop              -- はい，データの残りがあるので次のデータへ
_next_section:
    add     0x1,    r8              -- いいえ，次のセクションID番号(単位)へ
    cmp     r9,     r8              -- パッキング・セクション数＜次のセクションID番号(単位)?
    blt     _section_loop           -- いいえ，セクションの残りがあるので次のセクションへ

_ircopy_ok:
    mov     r0,     r10             -- はい，正常にコピー終了(成功)
    br      _ircopy_exit
_ircopy_ng:
    mov     -0x1,   r10             -- コピー失敗
_ircopy_exit:
    jmp     [lp]

#### end of ircopy.s ####
```

4. 2. 9　内蔵命令RAMに転送するプログラム例

約1秒ごとにDot LED1からLED4までを使いバイナリで1から8を繰り返し点灯し続けるプログラムです。

```
    .text
    .globl _main
_main:
    mov    0x01,   r11
    mov    0x8,    r12
_ledloop:
    mov    r11,    r6
    jarl   _ledout,lp

    movhi  0x00cb, r0, r6
    movea  0xc6cc, r6, r6
    jarl   _softwait,lp

    add    0x1,    r11
    add    -1,     r12
    cmp    r0,     r12
    bnz    _main
    jr     _ledtest

    .globl _ledout
_ledout:
    mov    r6,     r10
    shr    0x02,   r10
    and    0x3f,   r10
    st.b   r10,    P5

    mov    r6,     r10
    shl    0x04,   r10
    and    0x30,   r10
    st.b   r10,    P2
    jmp    [lp]

    .globl _softwait
_softwait:
    add    -1,     r6
    cmp    r0,     r6
    bnz    _softwait
    jmp    [lp]
```

4. 2. 10　内蔵命令RAMで動的実行するプログラム例

（1）動的プログラム・サンプルの実行概要

V850E/ME2では，RESETハンドラと内蔵命令RAMはそれぞれ次の番地から配置されています。

RESET　　：00100000H番地
内蔵命令RAM　：00000000H番地

内蔵命令RAMでは書き込みと読み出しが可能なため，内蔵命令RAMで動作させるプログラムを動的に置き換えることができます。

動的実行するプログラム1-3について説明します。動的プログラム1-3の動作はほとんど同じです。

main関数内では，UARTB1より次のように出力します。

```
Running application No.n（n = 1-3）
```

main関数内で永久ループしています。次のように8桁の数字を出力し続けます。

```
<pseudo 7segLED>=00000000
```

TB-V850E/ME2に7segLEDはありませんが，UARTB1からの出力数字8桁を擬似的に8桁の7segLEDに見立てたプログラムとなっています。

8桁のうち2桁だけ，アイドル・カウントにより，0-9へと数字回転表示を繰り返します。

```
[file:appli1.c]
   printf("¥r¥nRunning application No.1¥r¥n");/* 動的プログラムのID番号を表示 */
   for( ;; ){
         wait();
         if((place & DIGIT1) == GO)bcd01++;     /* 指定桁の回転は続行か? */
         if(bcd01 > 9)bcd01=0;                  /* 回転リールの数字は0-9でループ */
         if((place & DIGIT2) == GO)bcd02++;     /* 指定桁の回転は続行か? */
         if(bcd02 > 9)bcd02=0;                  /* 回転リールの数字は0-9でループ */
         /* 8桁のうち2桁回転 */
         printf("¥r<pseudo 7segLED>=000000%d%d",bcd02,bcd01);
   }
```

INTスイッチを押すとINTP11割り込みが受け付けられ，割り込み処理内で次の処理を実行します。

```
[file:appli1.c]
void intsw_for_app1(void){
   if(flag ^= ALTERNATIVE){  /* flagは割り込む毎に交互に0/1反転する    */
        /* 回転リール1桁目停止,2桁目回転 */
        seg7led(&place, DIGIT1, STOP);    /* 回転リール1桁目の表示停止 */
        seg7led(&place, DIGIT2, GO);      /* 回転リール2桁目の表示動作 */
        dotled(DOTLED1, ALLOFF);          /* Dot LED 全消灯 */
        dotled(DOTLED1, ON);              /* Dot LED1 点灯 */
   }else{
        /* 回転リール2桁目停止,1桁目回転 */
        seg7led(&place, DIGIT2, STOP);    /* 回転リール2桁目の表示停止 */
        seg7led(&place, DIGIT1, GO);      /* 回転リール1桁目の表示動作 */
        dotled(DOTLED2, ALLOFF);          /* Dot LED 全消灯 */
        dotled(DOTLED2, ON);              /* Dot LED2 点灯 */
   }
}
```

・数字回転表示している2桁のうち，左側の数字回転を許可，右側の数字回転を禁止

DotLEDnを点灯，DotLEDn+1を消灯

再び，INTスイッチを押すとINTP11割り込みが受け付けられ，割り込み処理内で次の処理を実行します。

・数字回転表示している2桁のうち，左側の数字回転を禁止，右側の数字回転を許可

DotLEDnを消灯，DotLEDn+1を点灯

INTスイッチを押すたびに，上記動作を交互にくりかえします。

動的プログラム1-3の動作で異なるのは，回転する2桁の数字位置と交互に点灯するDotLEDの位置のみです。次に相違点を示します。

条　件		動的プログラム1	動的プログラム2	動的プログラム3
回転する2桁の数字位置		000000??	000??000	??000000
INT SW奇数回目押し下げ時	停止する桁	000000?x	000?x000	?x000000
	点灯するDotLEDの位置	LED1	LED4	LED7
		LED8○　○LED4	LED8○　●LED4	LED8○　○LED4
		LED7○　○LED3	LED7○　○LED3	LED7●　○LED3
		LED6○　○LED2	LED6○　○LED2	LED6○　○LED2
		LED5○　●LED1	LED5○　○LED1	LED5○　○LED1
INT SW偶数回目押し下げ時	停止する桁	000000x?	000x?000	x?000000
	点灯するDotLEDの位置	LED2	LED5	LED8
		LED8○　○LED4	LED8○　○LED4	LED8●　○LED4
		LED7○　○LED3	LED7○　○LED3	LED7○　○LED3
		LED6○　●LED2	LED6○　○LED2	LED6○　○LED2
		LED5○　○LED1	LED5●　○LED1	LED5○　○LED1

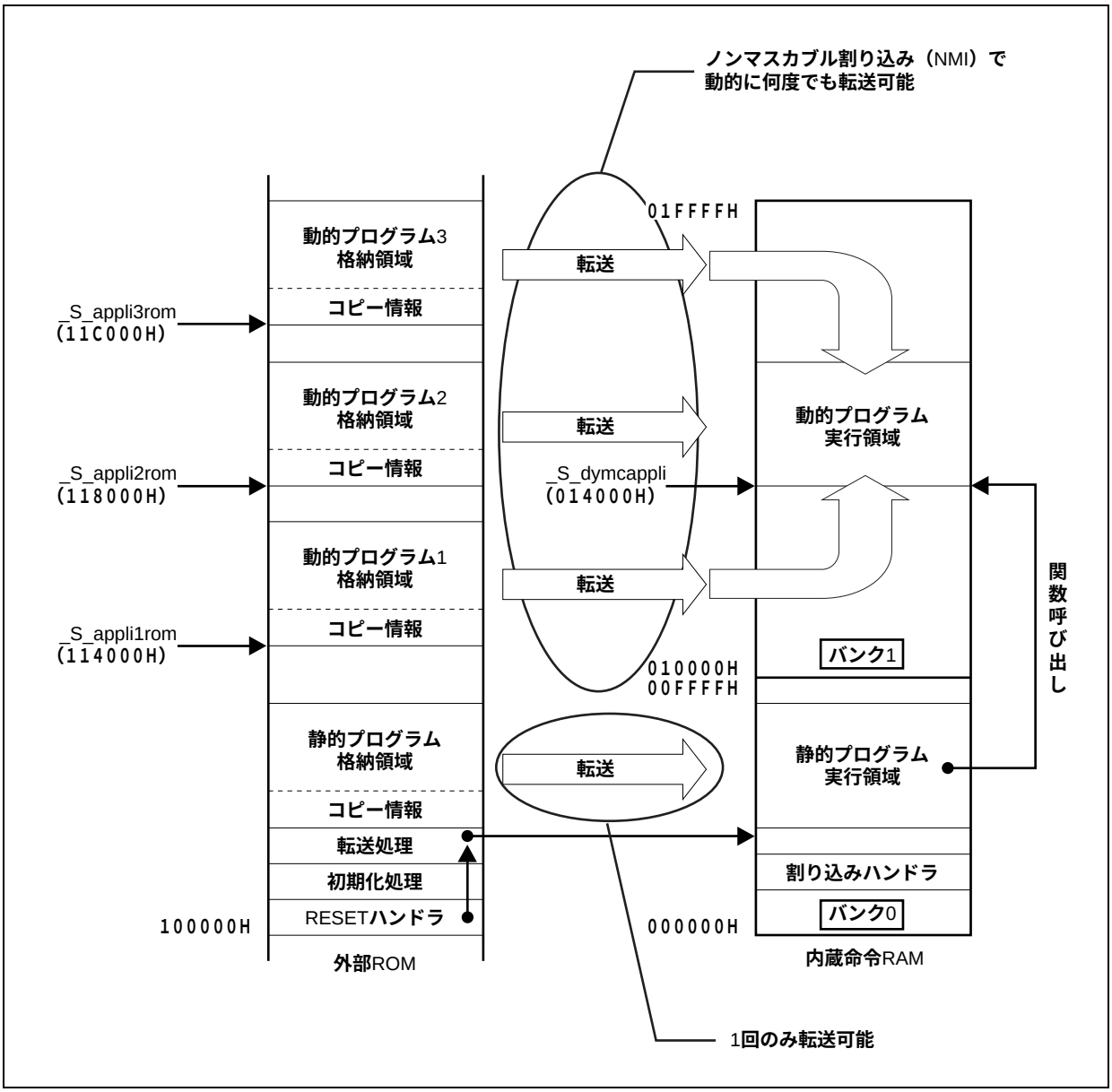
備考　??：数字回転している2桁の位置

x　：INT SWにより数字回転が停止する位置

RESETハンドラ → 初期化処理 → 転送処理によって，静的プログラムを外部ROMより内蔵命令RAMのバンク0へ転送後，静的プログラムの先頭へジャンプします。静的プログラムは動的プログラム_S_dymcappli( )関数を呼び出しています。動的プログラムはノンマスカブル割り込み（NMI）で動的に変更できます。

次に動的プログラム・サンプルの実行概要を示します。

図4－33　動的プログラム・サンプルの実行概要



(2) 静的プログラム・リンク時のイメージ

静的プログラムと，動的プログラムは，まったく別にコンパイル，リンクされます。

(a) 静的プログラムの中で扱う動的プログラム・リンクについて

動的プログラムの格納領域1-3および動的プログラムの実行領域は，すべて不定のため，空（先頭アドレスのみでコード実体0バイト）でリンクします。

静的プログラムでは，動的プログラムが格納されている外部ROM空間のラベル（固定アドレス）を外部参照します。このとき，text属性セクションのため，C言語での外部参照は外部変数名としてではなく，外部関数名として参照するので注意してください。

```
[file:progmain.c]
extern void _S_appli1rom(void);
extern void _S_appli2rom(void);
extern void _S_appli3rom(void);
```

```
[file:secentry.s]
# 動的に動かしたいプログラム1が格納されている外部EPROM領域1
        .section ".appli1rom",text
        .align   4
        .globl   _ _S_appli1rom, 4
_ _S_appli1rom:

# 動的に動かしたいプログラム2が格納されている外部EPROM領域2
        .section ".appli2rom",text
        .align   4
        .globl   _ _S_appli2rom, 4
_ _S_appli2rom:

# 動的に動かしたいプログラム3が格納されている外部EPROM領域3
        .section ".appli3rom",text
        .align   4
        .globl   _ _S_appli3rom, 4
_ _S_appli3rom:
```

```
[file:progmain.dir]
#動的に動作するプログラム1を格納する空セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
APPLI1ROM   : !LOAD ?RX V0x00114000 {
       .appli1rom      = $PROGBITS     ?AX A0x4 .appli1rom;
};

#動的に動作するプログラム2を格納する空セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
APPLI2ROM   : !LOAD ?RX V0x00118000 {
       .appli2rom      = $PROGBITS     ?AX A0x4 .appli2rom;
};

#動的に動作するプログラム3を格納する空セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
APPLI3ROM   : !LOAD ?RX V0x0011c000 {
       .appli3rom      = $PROGBITS     ?AX A0x4 .appli3rom;
};
```

ROM化したプログラムを内蔵命令RAMへ転送する考え方や方法については，4. 2. 8　**内蔵命令**RAM**の転送例**を参照してください。

(b) 静的プログラム中のROM**化されている静的実行プログラム・リンクについて**

内蔵命令RAMで静的実行する次のセクションはROM化する必要があります。

- .const**セクション**
- **ノンマスカブル割り込み（**NMI**）セクション**
- .text**セクション**

① .const**セクションは標準入出力ライブラリ・リンク時に生成されます。**

```
[file:progmain.dir]
#静的に動作するプログラムの CONSTセグメント領域
#内蔵命令RAM領域BANK0 0x00001000-0x0000ffff( 60KB)
CONST   : !LOAD ?R V0x0000f000 {
       .const          = $PROGBITS     ?A .const;
};
```

② ノンマスカブル割り込み（NMI）セクションは，NMI割り込み処理関数にC言語記述の拡張として，割り込み／例外ハンドラ記述を行うことにより生成されます。

```
[file:progmain.c]
#pragma interrupt NMI int_nmi
_ _interrupt void int_nmi(void){
(中略)
}
```

```
[Cコンパイラが内部的に付加するNMI割り込みディレクティブ]
NMI   : !LOAD ?RX V0x00000010 {
   NMI    = $PROGBITS ?AX NMI;
};
```

③ crE.s, initial.s, ledtest.s, ircopy.s以外のソース・ファイルでセクション名を指定し，かつ，ディレクティブ・ファイルでセクションを定義しなかったすべてのプログラム・コードは.textセクションとして生成されます。

```
[file:progmain.dir]
#静的に動作するプログラム・セグメント領域
#内蔵命令RAM領域BANK0 0x00001000-0x0000ffff( 60KB)
MAINTEXT    : !LOAD ?RWX V0x00001000{
       .text            = $PROGBITS     ?AX .text;
};
```

これらのセクションをCコンパイラ・パッケージ内付属のROM化ユーティリティromp850でパッキングするように-tオプションで指示します。

```
-t NMI -t .text -t .const
```

PM+では，メイン・ウインドウから[ツール(T)] → [ROM化プロセッサオプションの設定(R)...]メニューを選択し，ROM化プロセッサ・オプションの設定ダイアログを表示させ，[オプション]タブで[パッキングする読み出し専用セクション[-t](R)]のリスト・ボックスに“NMI;.text;.const”と「;」（セミコロン）で区切って入力します。

(c) ROM化用のオブジェクト作成方法について

ROM化用のオブジェクト作成では，外部EPROMに配置されるセクション最後尾にリンクする必要があります。

まず，rompsecセクションのエントリ名を_ _Srompackという新しい名前にし，アセンブリ・ソース・ファイルに記述します。また，そのエントリ名に対して.s_rompというセクション名を指定します。

エントリ名は，ROM化されない.textセクションの終端を越えて最初の（4バイトの整列条件で整列された）アドレスを指すようにします。

```
[file:rompack.s]
        .file "rompack.s"
        .section ".s_romp",text
        .align   4
        .globl   _ _S_rompack, 4
_ _S_rompack:
```

_ _Srompackという名前にしたrompsecセクションを.s_rompセクションとしてディレクティブ・ファイルで配置を指示します。

外部EPROMに配置されるセグメント・ディレクティブ最後尾に追加記述します。

セグメント・ディレクティブのディフォールトの整列条件は8バイトです。

```
[progmain.dir]
#rompsec seciton セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
SROMP       : !LOAD ?RX  {
   .s_romp             = $PROGBITS     ?AX .s_romp;
};
```

Cコンパイラ・パッケージ内付属のROM化ユーティリティromp850で_ _S_rompラベルをROM化パッキング開始アドレスにするように-bオプションで指示します。

PM+では，メイン・ウインドウから[ツール(T)] → [ROM化プロセッサオプションの設定(R)...]メニューを選択し，ROM化プロセッサ・オプションの設定ダイアログを表示させ，[オプション]タブで[エントリラベル[-b](E)]のリスト・ボックスに“_ _S_rompack”と入力します。

ROM化時に，ディフォールトのrompsecセクション用の領域確保コード（ファイル名：rompcrt.o）がリンクされ，_ _S_rompというラベルのセクションが，.textセクションの後ろに0バイト生成されても，無視してください。

(d) 内蔵命令RAMのバンク0へ静的プログラムを転送する方法について

転送には_ircopy関数を使用します。

_irocpy関数の詳細については，4. 2. 8　内蔵命令RAMの転送例を参照してください。

_irocpy関数呼び出し時の第1引数であるROM化されたセクションの先頭アドレスの渡し方には注意が必要です。

V850E/ME2では，RESETハンドラ・アドレスが00100000H番地です。また，サンプルの場合，ノンマスカブル割り込み（NMI）を使用しています。NMI割り込みハンドラ・アドレスは00000010H番地です。そのため，テキスト・ポインタtpの値は，.textセクション最上位番地であるNMI割り込みハンドラ・アドレスの00000010H番地でリンク解決されます。

プログラム・コード上のラベル（アドレス）は，テキスト・ポインタtp = 00000010H分が減算されて算出されます。

プログラム・コード上のラベル（アドレス）にアクセスする場合，tpポインタに注目し，必ずオフセットをかけ，00000010H分加算します。

この場合，直接rompsecセクションのエントリ名_ _S_rompackを引数指定すると，tpオフセットがかかっていない10番地ずれた番地を_ircopy関数へ引き渡すことになってしまいます。

```
   mov    _ _S_rompack, r6      -- 静的プログラム・パッキング・セクション先頭アドレス
   mov    -0x1,         r7      -- 転送したいパッキング・セクションはすべて
   jarl   _ _ircopy,    r31     -- ROM化されたセクションを内蔵命令RAM BANK0へ転送
```

この情報では，正常に内蔵命令RAMへROM化したプログラムを転送できません。次に示すようにtpオフセット値，00000010H番地を加算して，rompsecセクションのエントリ名を_ircopy関数へ引き渡します。

```
[file:crtE.s]
.option nowarning
   movhi  hi1(_ _S_rompack),   tp,  r1    -- tp のオフセットをかける
   movea  lo(_ _S_rompack),    r1,  r6    -- 静的プログラム・パッキング・
                                          -- セクション先頭アドレス
.option warning
   mov    -0x1,      r7                   -- 転送したいパッキング・セクション
                                          -- はすべて
   jarl   _ _ircopy, r31                  -- ROM化された動的プログラムを
                                          -- BANK0へ転送
```

なお，このようにtpオフセットに留意しなければならないのは，アセンブリ言語上でラベル（アドレス）を操作する場合のみです。

C言語上でラベル（アドレス）操作する場合は，Cコンパイラがtpオフセットを考慮した上記のようなアセンブラ命令を生成するので，そのままラベルが使えます。

たとえば，上記のアセンブリ・ソース記述は次のようなCソースとなります。

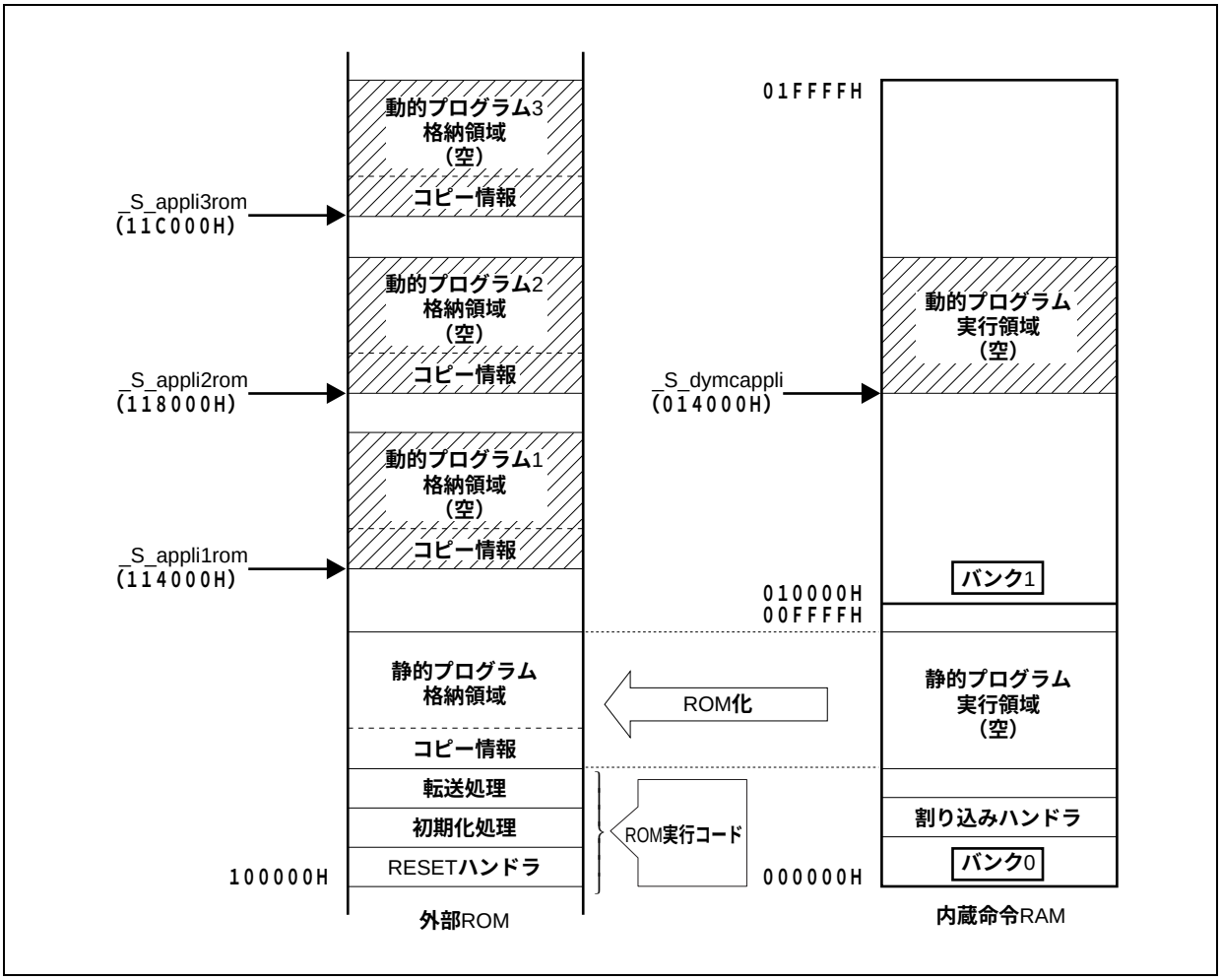
```
_ircopy(_Srompack,1);
```

また，内蔵命令RAMのバンク0で動作する静的プログラムでは，動的プログラムの呼び出しと動的プログラムの置き換えのみ行います。そのため，次に示すものは指定アドレスのみの空（コード実体なしの0バイト）でリンクします

- 動的プログラム実行領域先頭番地　：　_S_dymcappli
- 動的プログラム1格納領域先頭番地　：　_S_appli1rom
- 動的プログラム2格納領域先頭番地　：　_S_appli2rom
- 動的プログラム3格納領域先頭番地　：　_S_appli3rom

次に静的プログラム・リンク時のイメージを示します。

図4－34　静的プログラム・リンク時のイメージ



(3) 動的プログラム・リンク時のイメージ

動的プログラム1-3はすべて同じメモリ空間で実行されます。

プログラムがリンクされるアドレスCONSTセグメントがリンクされるアドレスはすべて同じです。

```
[file:progapp1.dir]
#内蔵命令RAM BANK1空間   0x00014000-0x00016fff( 12KB)
#動的プログラム群実行領域
TEXT       : !LOAD ?RWX V0x00014000 {
       .text           = $PROGBITS     ?AX .text;
};
```

動的プログラム1-3はすべて同じメモリ空間で実行されます。

```
#内蔵命令RAM BANK1空間   0x00017000-0x00017fff(  4KB)
#動的実行プログラム定数データ格納領域
CONST      : !LOAD ?R   V0x00017000 {
       .const          = $PROGBITS     ?A .const;
};
```

動的プログラム1-3を格納するROM化領域のメモリ空間は異なります。

```
[file:progapp1.dir]
#外部EPROM(16bit幅)空間 0x00114000-0x00116fff( 12KB)
#動的実行プログラム1格納領域
APPLI1ENTRY  : !LOAD ?RX V0x00114000 {
       .app1entry      = $PROGBITS     ?AX .app1entry;
};

[file:progapp2.dir]
#外部EPROM(16bit幅)空間 0x00118000-0x0011bfff( 12KB)
#動的実行プログラム2格納領域
APPLI2ENTRY  : !LOAD ?RX V0x00118000 {
       .app2entry      = $PROGBITS     ?AX .app2entry;
};

 [file:progapp3.dir]
#外部EPROM(16bit幅)空間 0x0011c000-0x0011ffff( 12KB)
#動的実行プログラム3格納領域
APPLI1ENTRY  : !LOAD ?RX V0x0011c000 {
       .app3entry      = $PROGBITS     ?AX .app3entry;
};
```

静的プログラムと動的プログラムのグローバル変数領域およびスタック領域のメモリ空間は重ならないようにします。

静的プログラムのグローバル変数領域は，.sdataと.sbss領域で0FFFD000H番地からおよそ300H程度使用しています。動的プログラムのグローバル変数領域は，同様に.sdataと.sbss領域で0FFFD800H番地からおよそ300H程度使用しています。静的プログラムも動的プログラムもスタック領域はその後の.bss領域から200H（512バイト）分下位番地から使用しています。

```
[file:progmain.dir]
#内蔵データRAMセグメント領域 0x0fffb000-0x00fffefff(20KB)--V850E/ME2
#内蔵データRAM予約領域 0xfff8000-0xfffafff(12KB)(アクセス禁止)
#内蔵データRAM実装領域 0xfffb000-0xfffcfff( 8KB)(ROM Monitor予約)
DATA  : !LOAD ?RW V0x0fffd000 {
       .data           = $PROGBITS     ?AW  .data;
       .sdata          = $PROGBITS     ?AWG .sdata;
       .sbss           = $NOBITS       ?AWG .sbss;
       .bss            = $NOBITS       ?AW  .bss;
};
```

★

```
[file:progapp1.dir,file:progapp2.dir,file:progapp3.dir]
#内蔵データRAM空間 0x0fffb000-0x00fffefff( 20KB)-- V850E/ME2 uPD703111A
#内蔵データRAM空間 0x0fffb000-0x00fffcfff(  8KB) -- V850E/ME2 ROM Monitor
                                                  -- 予約領域
DATA    : !LOAD ?RW V0x0fffd800 {
       .data           = $PROGBITS     ?AW  .data;
       .sdata          = $PROGBITS     ?AWG .sdata;
       .sbss           = $NOBITS       ?AWG .sbss;
       .bss            = $NOBITS       ?AW  .bss;
};
```

[静的プログラムのfile:crtE.s]

```
#------------------------------------------------------------------------------
#  system stack
#------------------------------------------------------------------------------
       .set  STACKSIZE, 0x200
       .globl    _ _stack
       .bss
       .lcomm    _ _stack, STACKSIZE, 4
```

[動的プログラムのfile:crtE.s]

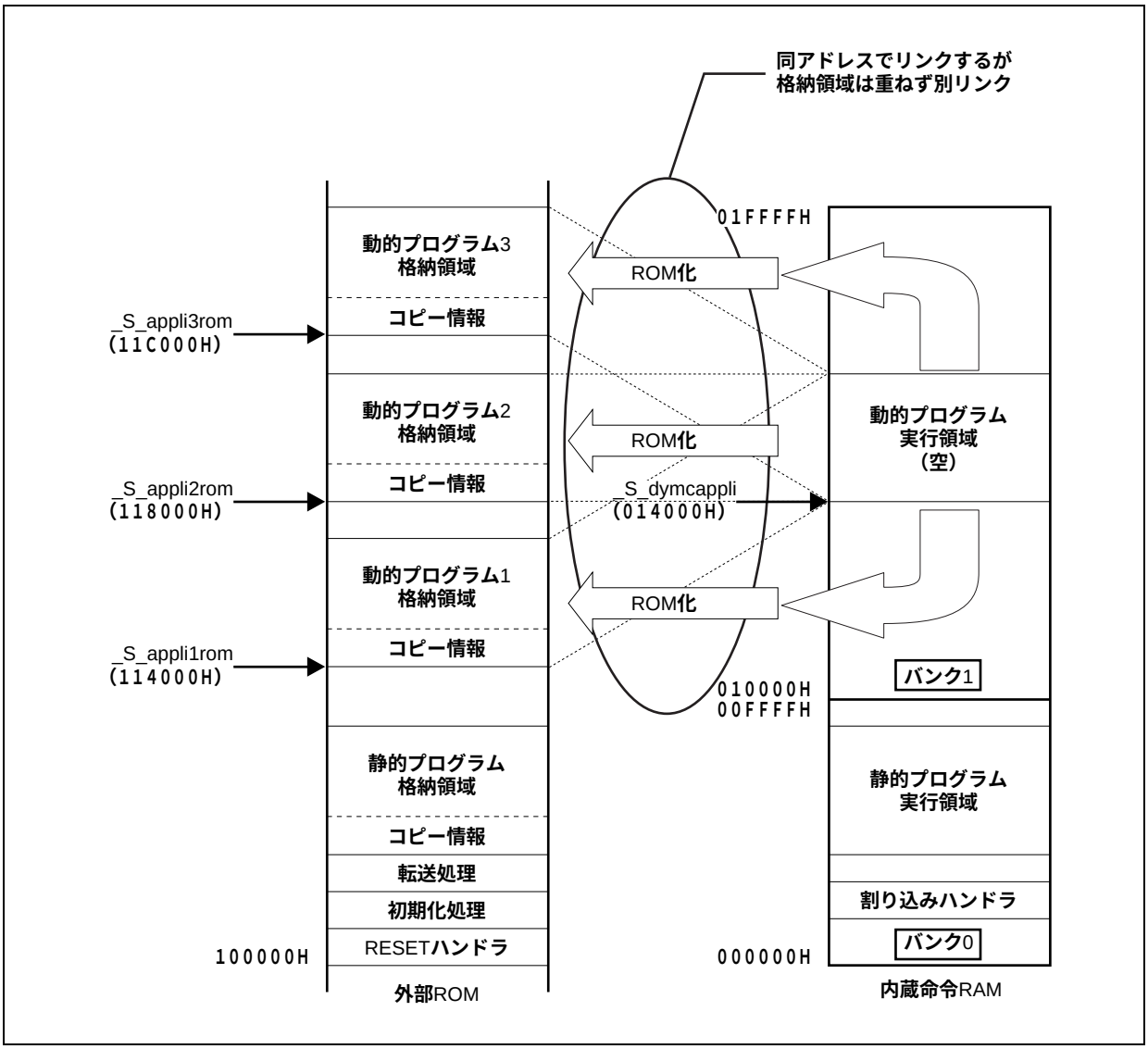
```
#------------------------------------------------------------------------------
#  system stack
#------------------------------------------------------------------------------
       .set  STACKSIZE, 0x200
       .bss
       .lcomm    _ _stack, STACKSIZE, 4
```

動的プログラム1-3はすべて，同じ先頭番地に配置しますが別々にリンクします。

ROM化領域（動的プログラム1-3格納領域先頭番地）は，各領域が重ならないように配置します。また，動的プログラム実行時に使用するグローバル変数領域は，静的プログラムが使用するグローバル変数領域と重ならないようにします。

次に動的プログラム・リンク時のイメージを示します。

図4－35　動的プログラム・リンク時のイメージ



(4) ROMに書き込むイメージ

(a) 動的プログラム1-3のリンクについて

動的プログラム1-3は，すべて同じメモリ空間上00114000H番地で実行されるため，ほとんど同じイメージでリンクされます。

動的プログラム1-3で異なるのは，プログラムを格納する領域のみです。

動的プログラム1を例にとります。

プログラム実行領域は内蔵命令RAMの00014000H番地に割り付けます。

```
[file:progapp1.dir]
#内蔵命令RAM BANK1空間   0x00014000-0x00016fff( 12KB)
#動的プログラム群実行領域
TEXT       : !LOAD ?RWX V0x00014000 {
       .text           = $PROGBITS     ?AX .text;
};
```

標準入出力ライブラリ・リンク時に使用する.constセクションは内蔵命令RAMの00017000H番地に割り付けます。

```
[file:progapp1.dir]
#内蔵命令RAM BANK1空間   0x00017000-0x00017fff(  4KB)
#動的実行プログラム定数データ格納領域
CONST1     : !LOAD ?R   V0x00017000 {
       .const          = $PROGBITS     ?A .const;
};
```

プログラム格納領域，すなわちrompsecセクションのエントリは00114000H番地に割り付けます。

```
[file:progapp1.dir]
#外部EPROM(16bit幅)空間 0x00114000-0x0016ffff( 12KB)
#動的実行プログラム1格納領域
APPLI1ENTRY  : !LOAD ?RX V0x00114000 {
       .app1entry      = $PROGBITS     ?AX .app1entry;
};
```

なお，プログラム格納領域，すなわちrompsecセクションのエントリは，動的プログラム2では00118000H番地に，動的プログラム3では0011C000H番地に割り付けます。

```
[file:secentry.s]
        .file    "secentry.s"
        .section ".app1entry",text
        .align   4
        .globl   _ _S_app1entry, 4
_ _S_app1entry:
```

最後に，ヘキサ・ファイルに変換しROMに書き込みます。

一般的なV850シリーズのCPUのROM領域は00000000H番地に存在しますが，V850E/ME2では00100000H番地からROM領域となります。ヘキサ・コード変換時にオフセットをかける必要があります。

Cコンパイラ・パッケージ内付属のヘキサ・コード変換ユーティリティhx850の-dオプションで指示します。

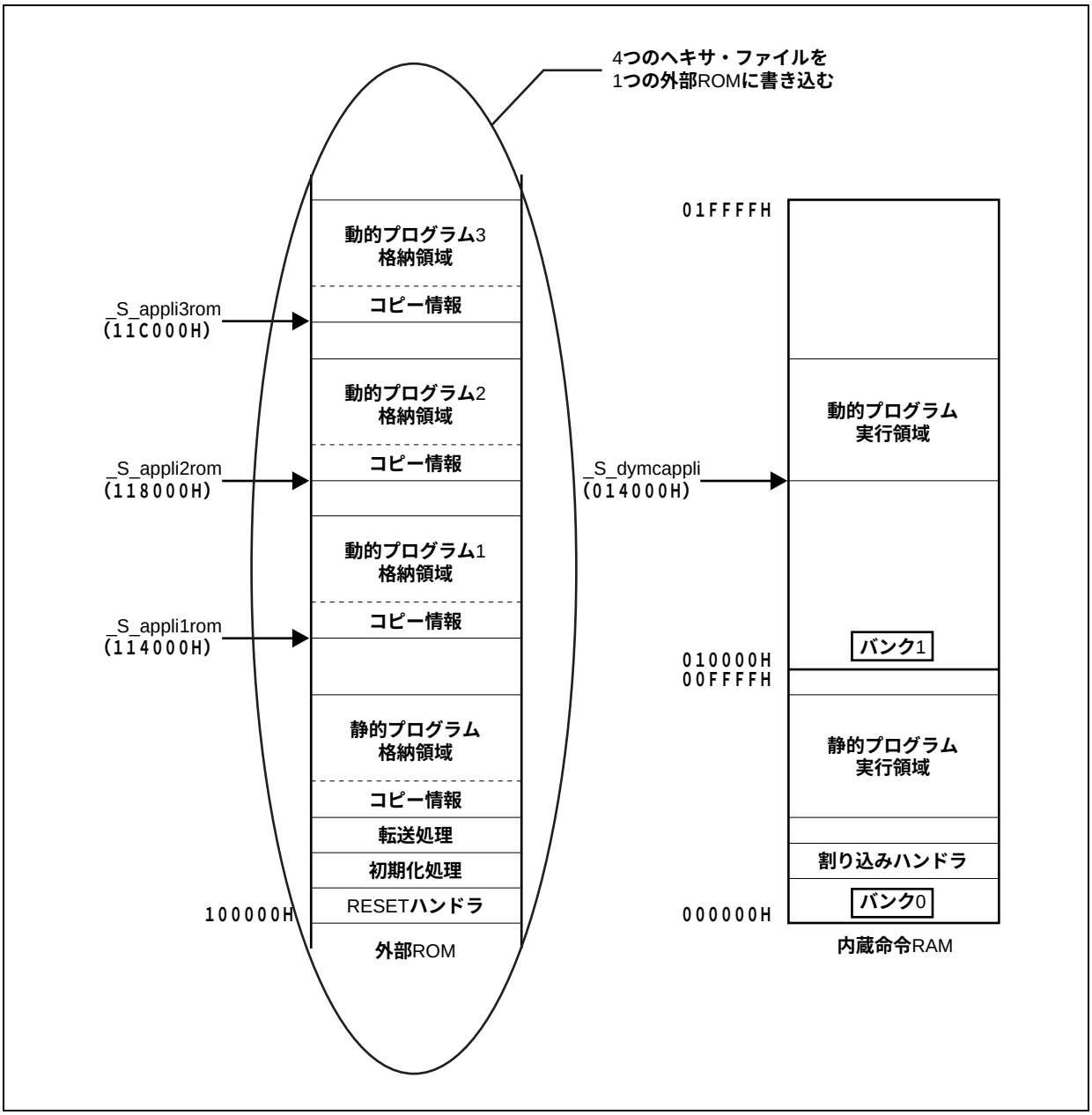
```
-d 0x100000
```

PM+では，メイン・ウインドウから[ツール(T)] → [ヘキサコンバータオプションの設定(H)...]メニューを選択し，ヘキサ・コンバータ・オプションの設定ダイアログを表示させ，[オプション]タブで[出力アドレスのオフセット[-d](O)]のコンボ・ボックスに“0x100000”と入力します。

静的プログラム，動的プログラム1，動的プログラム2，動的プログラム3をヘキサ・ファイルに変換し，ROMに書き込みます。これで動的プログラムの実行ROMが完成します。

次にROMに書き込むイメージを示します。

図4－36　ROMに書き込むイメージ



(b) 静的プログラムのリセットから静的プログラム転送までの動作

V850E/ME2のRESETハンドラ・アドレスは外部EPROM空間の100000H番地です。RESETハンドラ・アドレスはディレクティブで明示する必要があります。ディレクティブの詳細については，4. 2. 8 (3) (b) ディレクティブ記述についてを参照してください。

```
[file:crtE.s]

       .section "RESET", text
       jr        _ _start

[file:progmain.dir]
#CPU RESET セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
RESET   : !LOAD ?RX V0x00100000 {
   RESET               = $PROGBITS     ?AX RESET;
};
```

まず，RESETハンドラ・アドレスから，スタートアップ・モジュールの先頭__startラベルへジャンプします。ここでは，静的プログラムの初期化を行います。初期化の内容を次に示します。

- gp, tp, sp, epの設定
- V850E/ME2のCPU初期化処理
- CPU動作確認用DotLED点灯処理
- .sbssセクションの初期化
- .bssセクションの初期化
- 内蔵命令RAMへのプログラム転送処理

これらソース部分はCPUリセット後，1回のみ実行すれば良いので，内蔵命令RAMへ転送する必要はありません。そこで，ソース・ファイルへのセクション名記述とディレクティブ・ファイルへの設定により，内蔵命令RAMへ転送する静的プログラムから分けて配置します。

```
[file:crtE.s]
.section ".crtE",text
       .align   4
       .globl   _ _start
       .globl   _ _exit
       .globl   _ _startend
#      .extern  _ _ _PROLOG_TABLE
_ _start:
```

```
[file:progmain.dir]
#スタートアップ・モジュール・プログラム・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
CRTE       : !LOAD ?RX V0x00101000 {
   .crtE               = $PROGBITS     ?AX .crtE;
};
```

上記プログラム例では，ROM Monitorディバグ環境のため“ V0x00101000 ”番地になっていますが，アドレス位置は，RESETセグメントに連続してもかまいません。

次にV850E/ME2のCPU初期化処理とCPU動作確認用DotLED点灯処理のプログラム例を示します。

```
[file:crtE.s]
       jarl     _initial,lp            -- V850E/ME2 CPU initial
_initial_end:

       jr       _ledtest
       .globl   _ledtest_end
_ledtest_end:
```

```
[file:ledtest.s]
       .section ".ledtest",text
       .align   4
       .globl   _ledtest

       .extern  _ledtest_end

# 初期化指標処理
# Dot LED バイナリ・シフト
_ledtest:
```

```
[file:initial.s]
       .section ".initial", text
       .align   4
       .globl   _initial
_initial:
```

```
[file:progmain.dir]
#CPU初期化プログラム・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
INITIAL : !LOAD ?RX {
   .initial            = $PROGBITS     ?AX .initial;
};

#LED点灯プログラム・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
LEDTEST : !LOAD ?RX {
   .ledtest            = $PROGBITS     ?AX .ledtest;
};
```

次に内蔵命令RAMへのプログラム転送処理のプログラム例を示します。

```
[file:progmain.dir]
#内蔵命令RAM領域転送プログラム・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
IRCOPY    : !LOAD ?RX  {
   .ircopy             = $PROGBITS     ?AX .ircopy;
};
```

(5) 静的プログラム転送

プログラム転送には_ircopy関数を使用します。

_ircopy関数では，次の処理を実行します。

- 第1引数にROM化されたセクションの先頭アドレスを渡します。
- 第2引数にどのセクションを転送するかのセクションID番号を渡します。

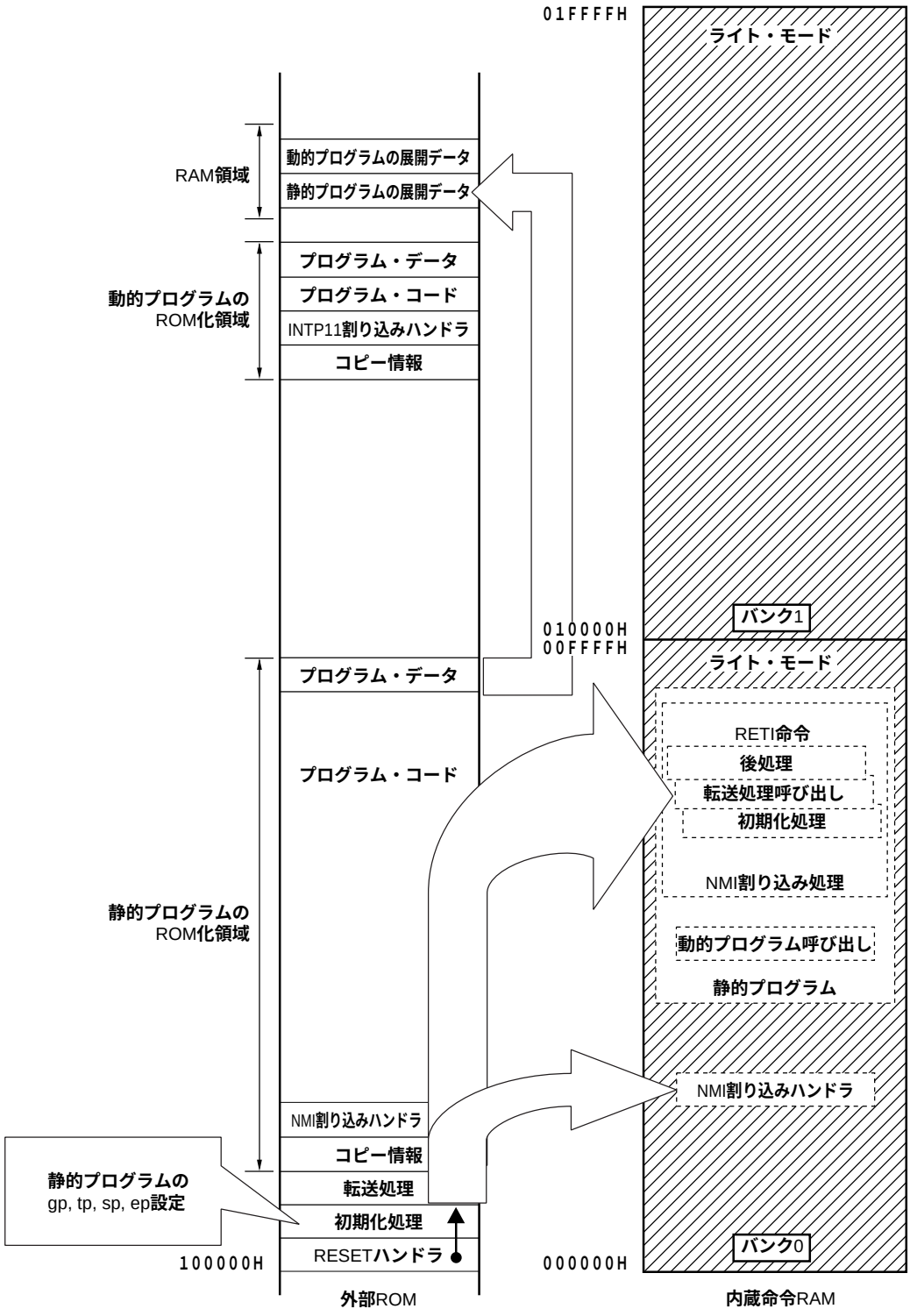
また，_ircopy関数の特徴を次に示します。

- スタックは使用しません。
- マスク・レジスタに使用されるr20とr21を破壊します。
- 転送単位はワード（4バイト）単位
- 転送サイズの最下位4ビットがワード（4バイト）単位の倍数（0x0, 0x4, 0x8, 0xc）ではなく，ハーフワード（2バイト）単位の倍数（0x2, 0x6, 0xa, 0xe）の場合，ワードに整列して転送します。そのため，転送先末尾にハーフワード分の不定データを転送します。

CPUリセット後は，内蔵命令RAMのバンク0，バンク1ともにライト・モードです。初期化終了後，外部ROM上の転送プログラムが，静的プログラムを外部ROMより読み出し，バンク0へ書き込みます。

次に処理の流れを示します。

図4－37　静的プログラム転送処理の流れ



備考　内蔵命令RAM内の各プログラム処理状態は次のようになっています。

：現在実行していない処理

(6) 最初に実行する動的プログラム転送

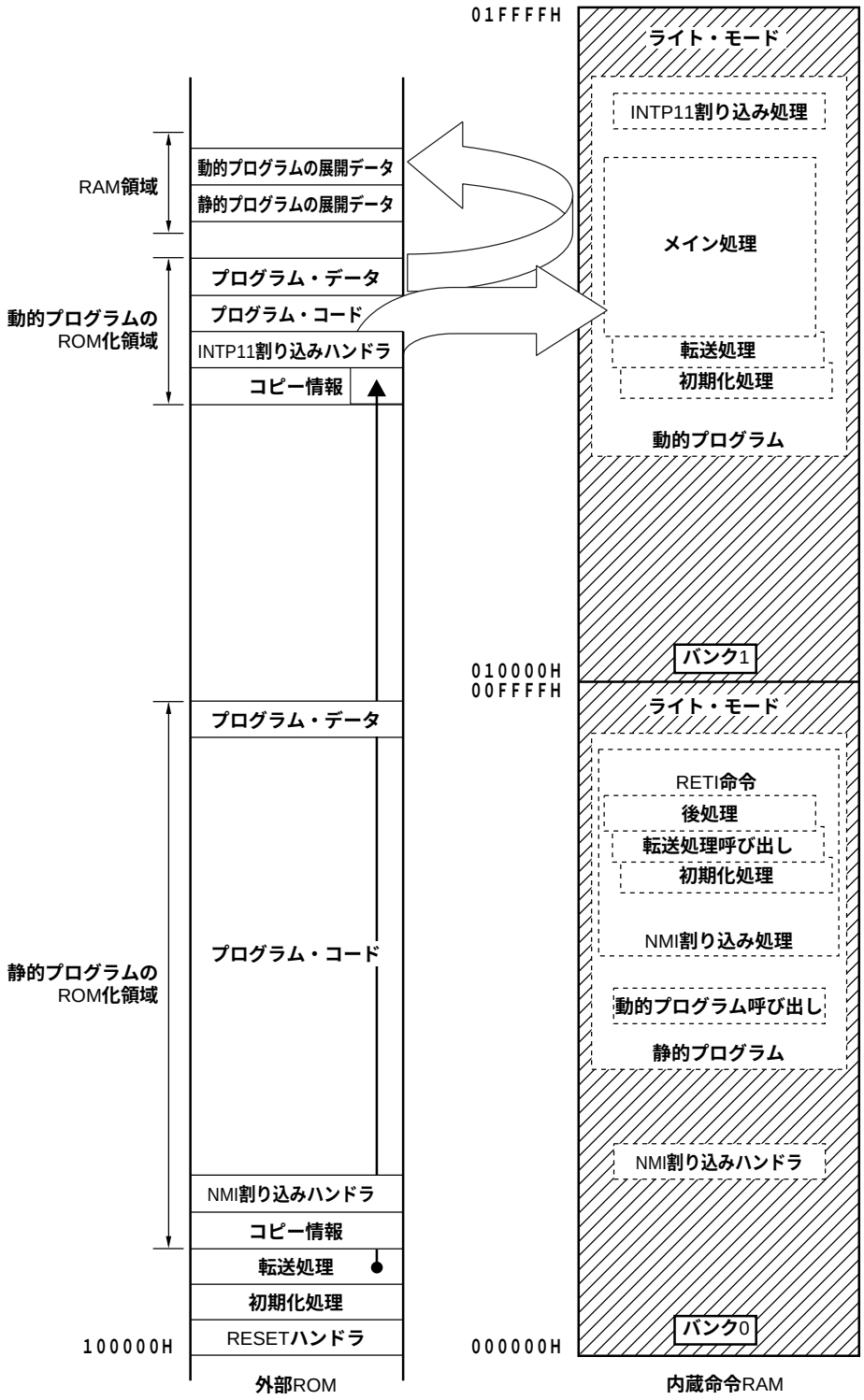
空（先頭アドレスのみでコード実体0バイト）でリンクした動的プログラム1のラベル（固定アドレス）と_ircopy関数を使用して最初に実行する動的プログラムを内蔵命令RAMへ転送します。ID番号に-1を渡すことによって，ROM化したコードとデータをすべて転送します。

```
[file:crtE.s]
.extern _ _S_appli1rom
.option nowarning
   movhi  hi1( _ _S_appli1rom), tp, r1    -- tpのオフセットをかける
   movea  lo(_ _S_appli1rom),   r1, r6    -- 動的プログラム1パッキング・
                                          -- セクション先頭アドレス
.option warning
   mov    -0x1,      r7                   -- 転送したいパッキング・セクション
                                          -- はすべて
   jarl   _ _ircopy, r31                  -- ROM化された動的プログラムを
                                          -- BANK1へ転送
```

静的プログラムを内蔵命令RAMのバンク0へ転送終了後，最初に実行する動的プログラムを外部ROMより読み出し，内蔵命令RAMのバンク1へ転送します。

次に処理の流れを示します。

図4－38　最初に実行する動的プログラム転送時の処理の流れ



備考　内蔵命令RAM内の各プログラム処理状態は次のようになっています。

：現在実行していない処理

（7）内蔵命令RAMでプログラム実行開始

（1）-（6）までの処理で，内蔵命令RAMでプログラム実行する準備が整ったので，最後にC言語のmain関数へジャンプします。

```
[file:crtE.s]
-- (iii)分岐命令により内蔵命令RAMへ
.option nowarning
   mov    #_main,    r1
   jmp    [r1]
.option warning
```

±2 Mバイト（±21ビット）までの範囲ならばjr命令でジャンプしてもかまいません。
main関数では，すぐに動的実行するプログラムを呼び出します。

```
[file:progmain.c]
extern void _S_dymcappli(void);
(中略)
   for( ;; ){
       _S_dymcappli();
   }
```

_S_dymcappli関数は動的にプログラム・コードが置き換わるため，空（コード実体のない0バイト）です。
ラベル（アドレス）だけ定義しています。

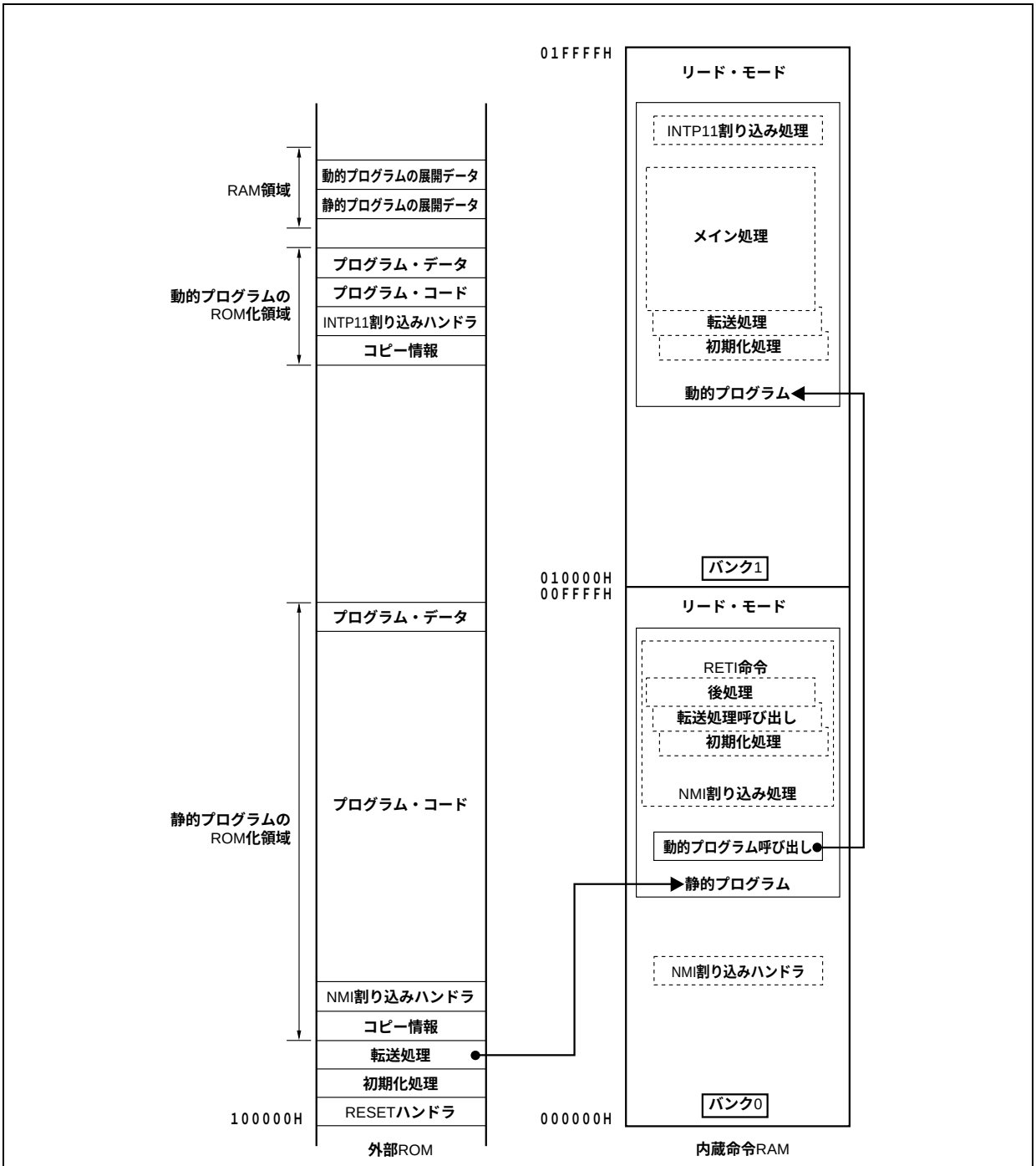
```
[file:secentry.s]
# 外部EPROM領域から転送した動的にプログラムを動かす内蔵命令RAM(Bank1)領域
       .section ".dymcappli",text
       .align   4
       .globl   _ _S_dymcappli, 4
_ _S_dymcappli:
```

```
[file:progmain.dir]
#動的に動作するプログラム・セグメント領域
#内蔵命令RAM領域BANK1 0x00010000-0x0001ffff( 64KB)
DYMCAPPL  : !LOAD ?RWX V0x00014000 {
       .dymcappli      = $PROGBITS     ?AX A0x4 .dymcappli;
};
```

すべてのプログラムを内蔵命令RAMへ転送終了後，内蔵命令RAMのバンク0，バンク1ともにリード・モードにして，静的プログラムのmain関数へジャンプします。main関数内では，動的プログラムを呼び出します。

次に処理の流れを示します。

図4－39　内蔵命令RAMでプログラム実行開始時の処理の流れ



備考　内蔵命令RAM内の各プログラム処理状態は次のようになっています。

：現在実行していない処理

：実行中の処理

(8) 動的プログラムの初期化

_S_dymcappli関数が呼び出されると，動的プログラムのスタートアップ・モジュールを通過し，次の処理を実行します。

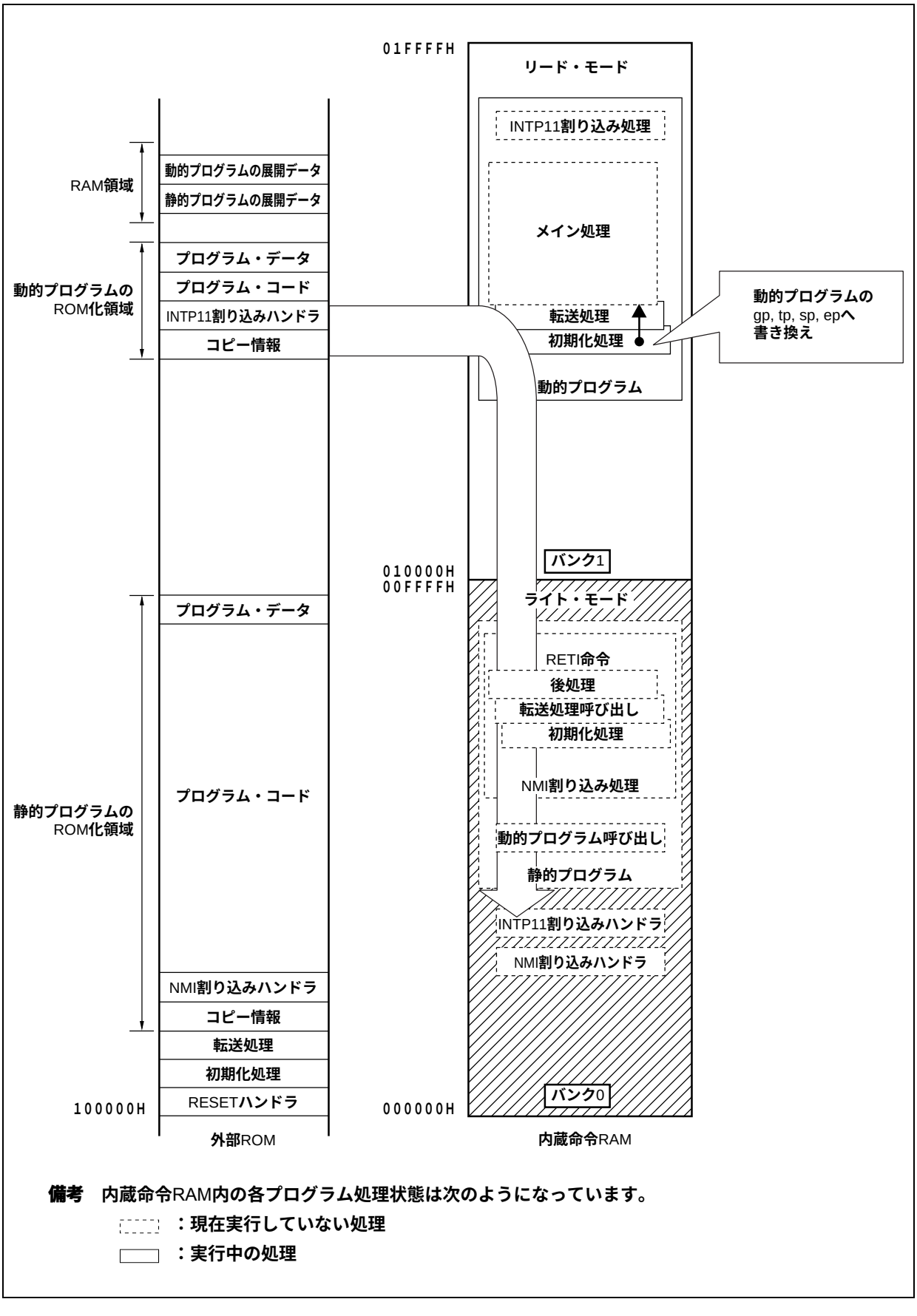
- gp, tp, sp, epの設定
- .sbssセクションの初期化
- .bssセクションの初期化
- INTP11割り込みセクション（INTP11割り込みハンドラ）の内蔵命令RAMのバンク0への転送

処理実行後，動的プログラムのmain関数へジャンプします。

動的プログラムはバンク1側で実行します。動的プログラムのための初期化後，内蔵命令RAMのバンク0を再びライト・モードにして，動的プログラム用のINTP11割り込みハンドラをROMより読み出し，バンク0へ書き込みます。

次に処理の流れを示します。

図4－40　動的プログラム初期化時の処理の流れ



備考　内蔵命令RAM内の各プログラム処理状態は次のようになっています。

：現在実行していない処理

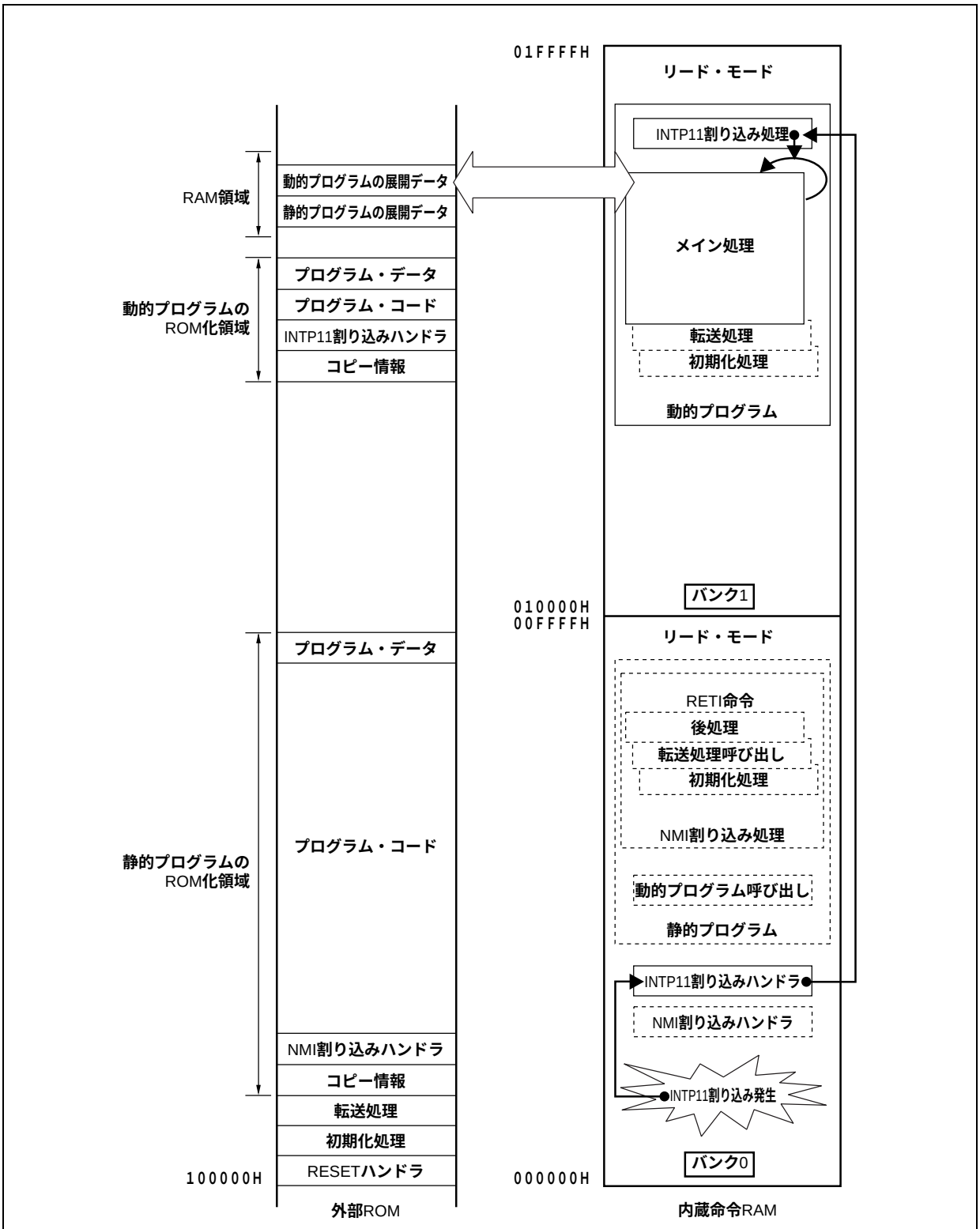
：実行中の処理

(9) 動的プログラムの稼動

動的プログラムのINTP11割り込みハンドラをバンク0へ書き込んだあと，バンク0をリード・モードに戻します。INTP11割り込みが発生すれば，バンク0のINTP11割り込みハンドラ経由で動的プログラムのINTP11割り込み処理を実行できます。

次に処理の流れを示します。

図4－41　動的プログラム稼動時の処理の流れ



備考　内蔵命令RAM内の各プログラム処理状態は次のようになっています。

：現在実行していない処理

：実行中の処理

（10）ノンマスカブル割り込み（NMI）による別の動的プログラムの置き換え

NMI割り込みのC言語拡張記述を次に示します。

なお，C言語拡張記述の詳細については，CA850　**ユーザーズ・マニュアル　C言語編**（U17291J）**の第4章　拡張言語仕様**を参照してください。

```
[file:progmain.c]
#pragma interrupt NMI0 int_nmi
_ _interrupt
```

NMI割り込みが発生すると，動的プログラムから静的プログラム側へ制御が戻ります。tp, gp, sp, epの初期化を行い，静的プログラムのC言語実行可能な環境へ復帰します。

```
[file:progmain.c]
   _ _asm("mov #_ _tp_TEXT, tp");      /* tp register */
   _ _asm("mov #_ _gp_DATA, gp");      /* gp register offset */
   _ _asm("add tp, gp");               /* gp register */
   _ _asm("mov #_ _stack+0x200, sp");  /* sp register */
   _ _asm("mov #_ _ep_DATA, ep");      /* ep register */
```

動的プログラム1-3が格納されている外部参照ラベル（固定アドレス）は配列にしてあります。NMI割り込みが発生するたびに，この配列から順番にアドレスを参照します。

```
[file:progmain.c]

extern void _S_dymcappli(void);
extern void _S_appli1rom(void);
extern void _S_appli2rom(void);
extern void _S_appli3rom(void);

/* グローバル変数定義 */
int id = 0;                                  /* 動的実行プログラムID番号 */
void (*appllitbl[4])(void)  ={               /* 動的実行プログラム・アドレス配列 */
_S_appli1rom,_S_appli2rom,_S_appli3rom, (void (*)())NULL };
（中略）
   /* 動的に実行するプログラムを順次変えて転送 */
   if(++id > 2)
        id = 0;
```

動的プログラム実行空間である内蔵命令RAMのバンク1をライト・モードにし，そのアドレスを_ircopy関数に引き渡して，新しい動的プログラム全体を転送します。

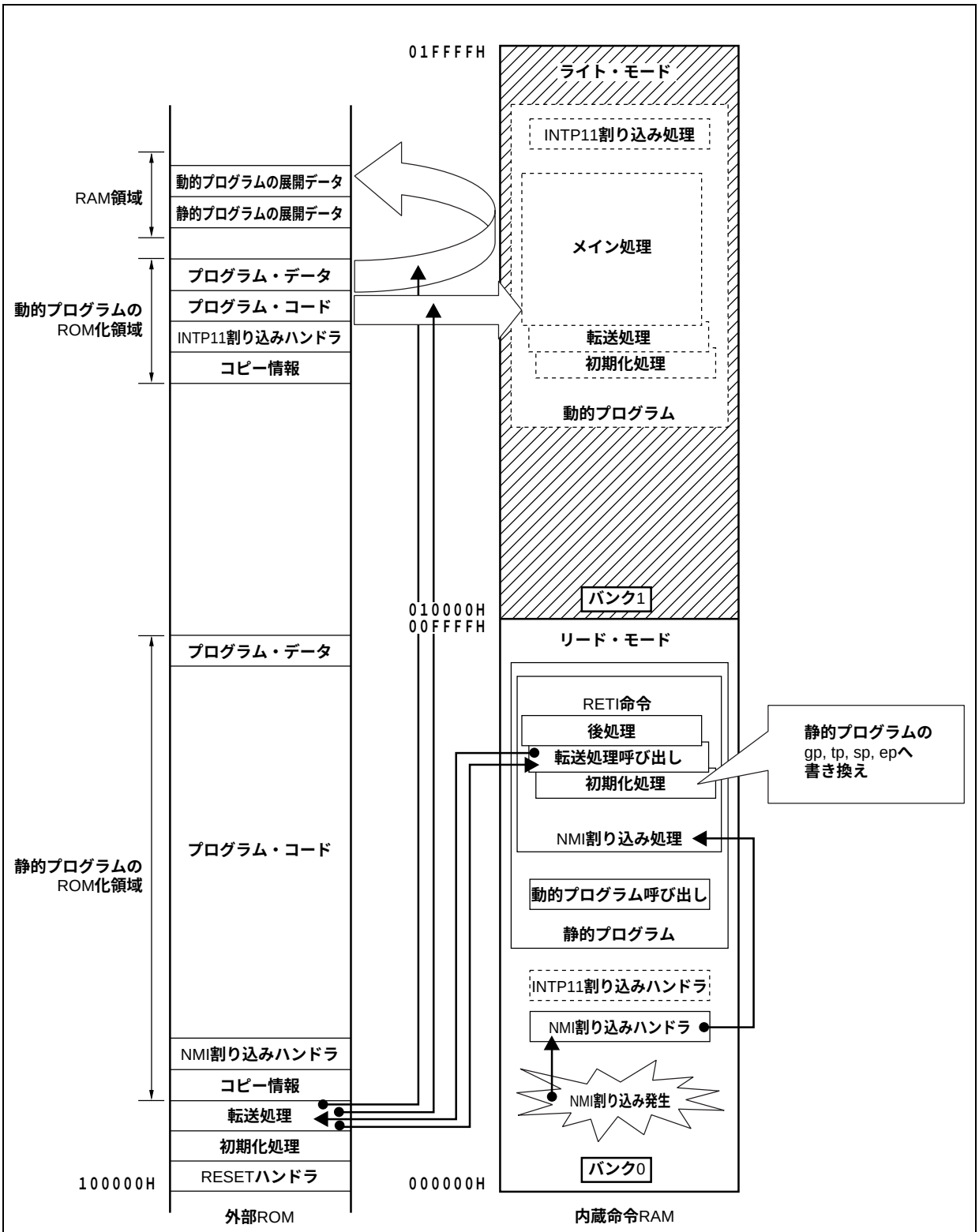
転送終了後は内蔵命令RAMのバンク1をリード・モードに戻します。

```
[file:progmain.c]
/* 内蔵命令RAM BANK1(0x00010000-0x0001ffff)をライト・モードへ */
IRAMM = 0x02; /* 内蔵命令RAMモード・レジスタ(IRAMM)をライト・モード設定(IRAMM1=1) */
do{
  ; /* 内蔵命令RAMモード・レジスタ(IRAMM)のライト・モード変更を確認(IRAMM1=1?) */
}while(IRAMM != 2);

_ircopy(applitbl[id],-1);
/* 内蔵命令RAM BANK1(0x00010000-0x0001ffff)をリード・モードへ */
IRAMM = 0x00; /* 内蔵命令RAMモード・レジスタ(IRAMM)をリード・モード設定(IRAMM1=0) */
do{
  ; /* 内蔵命令RAMモード・レジスタ(IRAMM)のリード・モード変更を確認(IRAMM1=0?) */
}while(IRAMM != 0);
```

NMI割り込み処理は内蔵命令RAMのバンク0の静的プログラム側で実行します。静的プログラムのための初期化後，NMI割り込み処理内で内蔵命令RAMのバンク1をライト・モードにして，動的プログラムをROMより読み出し，バンク1へ書き込みます。NMI割り込みハンドラは内蔵命令RAMのバンク0へ書き込むので，この段階では，まだ書き込めません。

次に処理の流れを示します。

図4－42　ノンマスカブル割り込み（NMI）による別の動的プログラムの置き換え処理の流れ



備考　内蔵命令RAM内の各プログラム処理状態は次のようになっています。

：現在実行していない処理

：実行中の処理

（11）ノンマスカブル割り込み（NMI）より復帰して別の動的プログラムへ

NMI割り込みの戻り先アドレスは，システム・レジスタのFEPCに格納されています。

RETI命令と同時にFEPCの値がPCに書き込まれます。置き換わる前のNMI割り込みが入った動的プログラムに戻るのではなく，新たな動的プログラムの最初から実行する必要があります。

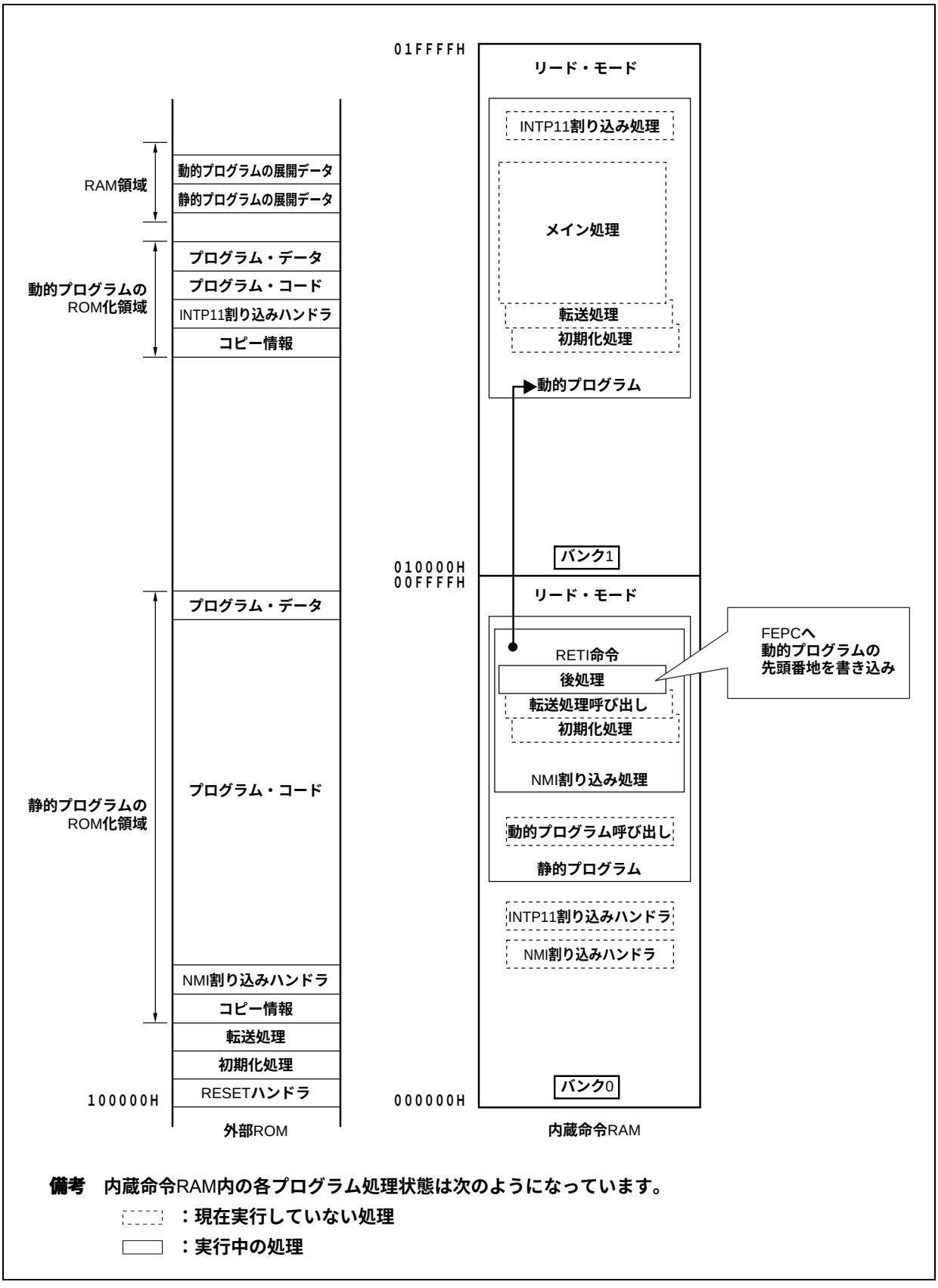
動的プログラム実行領域の先頭ラベル（固定アドレス）である_S_dymcappliをFEPCに上書きしてNMI割り込みから復帰すれば，置き換わったプログラムの先頭から動き始めます。

```
[file:progmain.c]
   /* 動的に実行するプログラムを転送する領域の先頭アドレスへNMI0割り込みの戻り先指定 */
   _ _asm("movhi hi1(#_ _S_dymcappli),r0,r6");
   _ _asm("movea lo(#_ _S_dymcappli), r6,r6");
   _ _asm("ldsr  r6, 2");
```

NMI割り込み処理を抜ける前に，動的プログラム実行領域である内蔵命令RAMのバンク1をリード・モードに，NMI割り込みの戻り先番地を動的プログラム実行領域の先頭に設定します。

次に処理の流れを示します。

図4－43　ノンマスカブル割り込み（NMI）より復帰して別の動的プログラムを実行時の処理の流れ



備考　内蔵命令RAM内の各プログラム処理状態は次のようになっています。

：現在実行していない処理

：実行中の処理

（12）プログラム・リスト

（a）静的プログラム・リスト

次に静的プログラム・リストのソース・ファイル一覧と静的プログラム・リストを示します。

表4－11　静的プログラム・リストのソース・ファイル一覧

ソース・ファイル名	概　　要
progmain.c	main関数，NMI関数
initial.s	V850E/ME2初期化処理
ledtest.s	DotLED点灯処理
printf.c	UARTB1出力処理関数群
uart.h	UARTB関連ヘッダ・ファイル
crtE.s	静的プログラム用スタートアップ・モジュール処理
secentry.s	空リンク・エントリ
rompack.s	rompsecセクション・エントリ
progmain.dir	静的プログラム用リンク・ディレクティブ
ircopy.s	_ircopy関数（4. 2. 8（6）プログラム例参照）

```
 [progmain.c]
#pragma ioreg
#ifndef NULL
#ifndef NULL 0
#endif
#define DOTLED8_3 P5

/* 外部変数参照宣言 */
extern int _ _ _stack;
extern void _S_rompack();
extern void _S_dymcappli(void);
extern void _S_appli1rom(void);
extern void _S_appli2rom(void);
extern void _S_appli3rom(void);

/* グローバル変数定義 */
int id = 0;                              /* 動的実行プログラムID番号 */
void (*applitbl[4])(void) ={             /* 動的実行プログラム・アドレス配列 */
_S_appli1rom,_S_appli2rom,_S_appli3rom, (void (*)())NULL };

/************************************************************************
 *
 *  割り込み要求名：NMI0
 *  機　　能：静的プログラムで稼動するNMI SW割り込み処理
 *
 ************************************************************************/
#pragma interrupt NMI0 int_nmi
_ _interrupt void int_nmi(void){

     _ _asm("mov #_ _tp_TEXT, tp");      /* tp register */
     _ _asm("mov #_ _gp_DATA, gp");      /* gp register offset */
     _ _asm("add tp, gp");               /* gp register */
     _ _asm("mov #_ _stack+0x200, sp"); /* sp register */
     _ _asm("mov #_ _ep_DATA, ep");      /* ep register */

  /* 動的に実行するプログラムを転送する領域の先頭アドレスへNMI0割り込みの戻り先指定  */
     _ _asm("movhi hi1(#_ _S_dymcappli),r0,r6");
     _ _asm("movea lo(#_ _S_dymcappli), r6,r6");
     _ _asm("ldsr  r6, 2");

    /* 動的に実行するプログラムを順次変えて転送 */
    if(++id > 2)
        id = 0;

    /* 内蔵命令RAM BANK1(0x00010000-0x0001ffff)をライト・モードへ */
    IRAMM = 0x02;/* 内蔵命令RAMモード・レジスタ(IRAMM)をライト・モード(IRAMM1=1) */
    do{
      ; /* 内蔵命令RAMモード・レジスタ(IRAMM)のライト・モード変更を確認(IRAMM1=1?) */
    }while(IRAMM != 2);
```

```
    _ircopy(appllitbl[id],-1);

    /* 内蔵命令RAM BANK1(0x00010000-0x0001ffff)をリード・モードへ */
    IRAMM = 0x00;/* 内蔵命令RAMモード・レジスタ(IRAMM)をリード・モード(IRAMM1=0) */
    do{
      ; /* 内蔵命令RAMモード・レジスタ(IRAMM)のリード・モード変更を確認(IRAMM1=0?) */
    }while(IRAMM != 0);
}

/************************************************************************
 *
 *  関数名: main
 *  機　能: 静的プログラム・メイン処理
 *  引　数: なし
 *  戻り値: なし
 *
 ************************************************************************/
void main(void)
{
/************************************************************************/
/**** このプログラムは内蔵命令RAM BANK0領域(0x00000000-0x0000ffff)で動く     ****/
/************************************************************************/

     init_uartb1(19200); /* UARTB1 チャネル初期化  */
     printf("¥r¥n**** Dynamic Inplement Sample Program ****¥r¥n¥r¥n");

     for(;;){
        _S_dymcappli();
     }
}
```

```
 [initial.s]
#############################################################
# -------- Internal I/O register initialize -----------
# V850E/ME2の初期化シーケンスは下記の手順
#
# Ⅰ外部バス・アクセス性能に影響するレジスタの設定
# Ⅱロック・レジスタ(LOCKR)のLOCKビットの確認
-- ( 0 )ロック・レジスタ(LOCKR)を読み出してPLLの安定を確認
-- ( i )システム・ウエイト・コントロール・レジスタ(VSWC)へx7Hを設定(x:Ⅰで設定した値を設定)
-- ( ii)バス・モード・コントロール・レジスタ(BMC)へ外部バスの周波数分周値設定
-- (iii)システム・ウエイト・コントロール・レジスタ(VSWC)へⅠで設定値に再度設定
-- ( iv)クロック・コントロール・レジスタ(CKC)へ内部システム・クロック周波数分周値設定
-- ( v )クロック・ソース選択レジスタ(CKS)へメイン・クロック(fx)供給制御を設定
# Ⅲ内蔵命令RAMにプログラム・コードを転送
#
#############################################################
     .set    TBV850E, 1
     .section ".initial", text
     .align  4
     .globl  _initial
_initial:
#########################################################
# Ⅰ外部バス・アクセス性能に影響するレジスタの設定
#########################################################
#   内蔵周辺I/Oアクセスのウエイト設定
     mov     0x33,   r6              -- 125.00MHz＜fx≦150.00MH時のVSWC設定値
     st.b    r6,     VSWC[r0]        -- システム・ウエイト・コントロール・レジスタ

#########################################################
#   各CS空間ごとのデータ・ウエイトを指定
#########################################################
     mov     0x0003, r6              -- CS0:EPROM    空間 3 wait(TW)挿入
                                     -- CS1:SDRAM    空間 0 wait(TW)
                                     -- CS2:SRAM     空間 0 wait(TW)
                                     -- CS3:SRAM     空間 0 wait(TW)
     mov     0x0000, r7              -- CS4:SDRAM    空間 0 wait(TW)
                                     -- CS5:SRAM     空間 0 wait(TW)
                                     -- CS6:SDRAM    空間 0 wait(TW)
                                     -- CS7:なし
     st.h    r6,     DWC0[r0]        -- データ・ウエイト・コントロール・レジスタ0
     st.h    r7,     DWC1[r0]        -- データ・ウエイト・コントロール・レジスタ1

     mov     0x0000, r6              -- CS0:EPROM    空間 0 wait(TASW)
                                     -- CS1:SDRAM    空間 0 wait(TASW)
                                     -- CS2:SRAM     空間 0 wait(TASW)
                                     -- CS3:SRAM     空間 0 wait(TASW)
                                     -- CS4:SDRAM    空間 0 wait(TASW)
                                     -- CS5:SRAM     空間 0 wait(TASW)
                                     -- CS6:SDRAM    空間 0 wait(TASW)
                                     -- CS7:なし
```

```
     st.h    r6,     ASC[r0]          -- アドレス・セットアップ・ウエイト・コントロール・レジスタ

# DMAフライバイ転送ウエイト・コントロール・レジスタ(FWC)は初期値(7777H)
--FW00~02=111 DMA ch.0使用時 SDRAM対象:7wait/SDRAM非対象:8wait挿入
--FW10~12=111 DMA ch.1使用時 SDRAM対象:7wait/SDRAM非対象:8wait挿入
--FW20~22=111 DMA ch.2使用時 SDRAM対象:7wait/SDRAM非対象:8wait挿入

#########################################################
#   各CS空間ごとのアイドル・ステートを指定
#########################################################
     mov     0x0452, r6               -- CS0:EPROM    空間 2 idle(TI)挿入
                                      -- CS1:SDRAM    空間 0 idle(TI)
                                      -- CS2:SRAM     空間 1 idle(TI)挿入
                                      -- CS3:SRAM     空間 1 idle(TI)挿入
                                      -- CS4:SDRAM    空間 0 idle(TI)
                                      -- CS5:SRAM     空間 1 idle(TI)挿入
                                      -- CS6:SDRAM    空間 0 idle(TI)
                                      -- CS7:なし
     st.h    r6,     BCC[r0]          -- バス・サイクル・コントロール・レジスタ

# DMAフライバイ転送アイドル・コントロール・レジスタ(FIC)は初期値(3333H)
--FI00~01=11 DMA Ch.0使用時 3idle挿入
--FI10~11=11 DMA Ch.1使用時 3idle挿入
--FI20~21=11 DMA Ch.2使用時 3idle挿入

#########################################################
#   256MBのメモリ空間のうち，
#   下位8MB(00000000H-07FFFFFFH)
#   上位8MB(0F800000H-0FFFFFFFH)
#   を2MB単位でのメモリ・ブロックに分割し
#   チップ・セレクト信号を指定
#########################################################
     mov     0x0e01, r6               -- エリア0(64MB)
                                      -- ブロック0 x0100000H-x01fffffH:( 1MB)CS0
                                      -- ブロック1 x0200000H-x03fffffH:( 2MB)CS2
                                      -- ブロック2 x0400000H-x05fffffH:( 2MB)CS2
                                      -- ブロック3 x0600000H-x07fffffH:( 2MB)CS2
                                      --          x0800000H-x3fffffH:(56MB)CS1固定

                                      -- エリア1(64MB)
                                      --          x4000000H-x7fffffH:(64MB)CS3固定
                                      -- エリア2(64MB)
                                      --          x8000000H-xbfffffH:(64MB)CS4固定
```

```
    mov     0x0f00, r7              -- エリア3(64MB)
                                    --          xc000000H-xf7fffffH:(56MB)CS6固定
                                    -- ブロック4 xf800000H-xf9fffffH:( 2MB)CS5
                                    -- ブロック5 xfa00000H-xfbfffffH:( 2MB)CS5
                                    -- ブロック6 xfc00000H-xfdfffffH:( 2MB)CS5
                                    -- ブロック7 xfe00000H-xffffffH:( 2MB)CS5
                                    -- CS7 未      使      用
    st.h    r6,     CSC0[r0]        -- チップ・エリア選択コントロール・レジスタ0
    st.h    r7,     CSC1[r0]        -- チップ・エリア選択コントロール・レジスタ1

########################################################
#   各CS空間ごとの接続する外部デバイスを指定
########################################################
    mov     0x88b8, r6              -- CS0:EPROM x0100000H-x011ffffH(128KB)
                                    -- CS1:SDRAM x1000000H-x2fffffH( 32MB)
                                    -- CS2:SRAM  x0200000H-x027ffffH(512KB)
                                    -- CS3:SRAM  x4000000H-x403ffffH(256KB)
    mov     0x0b8b, r7              -- CS4:SDRAM x8000000H-x8fffffH( 16MB)
                                    -- CS5:SRAM  xF800000H-xF81ffffH(128KB)
                                    -- CS6:SDRAM xC000000H-xCfffffH( 16MB)
                                    -- CS7:なし
    st.h    r6,     BCT0[r0]        -- バス・サイクル・タイプ・コンフィギュレーション・レジスタ0
    st.h    r7,     BCT1[r0]        -- バス・サイクル・タイプ・コンフィギュレーション・レジスタ1

########################################################
#   各CS空間ごとのデータ・バス幅を指定
########################################################
    mov     0x0169, r6              -- CS0:EPROM x0100000H-x011ffffH(128KB)16bit
                                    -- CS1:SDRAM x1000000H-x2fffffH( 32MB)32bit
                                    -- CS2:SRAM  x0200000H-x027ffffH(512KB)32bit
                                    -- CS3:SRAM  x4000000H-x403ffffH(256KB)16bit
                                    -- CS4:SDRAM x8000000H-x8fffffH( 16MB)16bit
                                    -- CS5:SRAM  xF800000H-xF81ffffH(128KB) 8bit
                                    -- CS6:SDRAM xC000000H-xCfffffH( 16MB) 8bit
                                    -- CS7:なし                             8bit
    st.h    r6,     LBS[r0]         -- ローカル・バス・サイジング・コントロール・レジスタ

########################################################
#   各CS空間ごとのエンディアンを指定
########################################################
    mov     0x0000, r6              -- CS0-CS7全空間 リトル・エンディアン
    st.h    r6,     BEC[r0]         -- エンディアン・コンフィギュレーション・レジスタ

########################################################
#   各CS空間ごとのデータ・リード制御機能を指定
########################################################
    mov     r0,     r6              -- CS0-CS3全空間 先読みなし(リード・バッファ動作の禁止)
    mov     r0,     r7              -- CS4-CS7全空間 先読みなし(リード・バッファ動作の禁止)
    st.h    r6,     LBC0[r0]        -- ライン・バッファ・コントロール・レジスタ0
    st.h    r7,     LBC1[r0]        -- ライン・バッファ・コントロール・レジスタ1
```

```
#########################################################
#   オンページ・アクセス時のウエイトの指定など(未実装)
#########################################################
--    mov      0x7000,  r6                -- 2×32bit,4×16bit,8×8bit
                                          -- オンページ・アクセス時に 7 wait(TW)挿入
--    st.h     r6,      PRC[r0]           -- ページROMコンフィギュレーション・レジスタ

#########################################################
#   D16-D31端子の使用有無を指定
#########################################################
-- PMDH,PMCDH,PFCDHの設定は16bitバス・モード時の設定です。
-- MODE1端子=1,MODE0端子=0設定時16bitバス・モードになります。
      mov      0x0000,  r6                -- PDH15-PDH0のインアクティブ・レベル・ロウを設定
      st.h     r6,      PDH[r0]
      mov      0x0000,  r6                -- PDH15-PDH0は出力モード選択
      st.h     r6,      PMDH[r0]
      mov      0x8000,  r6                -- PDH15は制御端子選択(INTPD15/PWM1)
      st.h     r6,      PMCDH[r0]
-- PFCDHのBMODCN=1とすると32bitデータ・バスとの混在使用が可能になります。
--    mov      0x8000,  r6                -- PDH15はPWM1端子選択
      mov      0x0001,  r6                -- 16bitモードでの起動時に
                                          -- PDH15-PDH0をD31-D16端子として使用
      st.h     r6,      PFCDH[r0]         -- ポートDHファンクション・コントロール・レジスタを書き込み
-- PFCDHのBMODCN=1設定時PDH15-PDH0はD31-D16端子となり，PDH,PMDH,PMCDHの上記設定は無効です。

_BMODCN_NOFIX:
      ld.h     PFCDH[r0],r6               -- ポートDHファンクション・コントロール・レジスタを読み出し
      movea    0x0001,  r0,   r7          -- BMODCN bit抽出
      and      r7,      r6
      cmp      r6,      r7                -- BMODCN設定未完了?
      jnz      _BMODCN_NOFIX              -- はい
--MODE1端子=0,MODE0端子=0設定時32bitバス・モードとなり，PMDH,PMCDH,PFCDHの上記設定は無効です。

#########################################################
#   バス制御端子を指定
#########################################################
      mov      0x0003,  r6                -- 制御端子選択
      st.h     r6,      PMCAL[r0]
      mov      0x03,    r6                -- A0-A1出力端子選択
      st.b     r6,      PFCALL[r0]

      mov      0x03ff,  r6                -- A16-A25出力端子選択
      st.h     r6,      PMCAH[r0]

      mov      0xff,    r6                -- CS0-CS7出力信号のインアクティブ・レベル・ハイを設定
      st.b     r6,      PCS[r0]
      mov      0xff,    r6                -- CS0-CS7端子選択
      st.b     r6,      PMCCS[r0]
```

```
    mov     0xbf,   r6              -- BCYST,WE/WR,RD,
                                    -- UUWR/UUBE/UUDQM,
                                    -- ULWR/ULBE/ULDQM,
                                    -- LUWR/LUBE/LUDQM,
                                    -- LLWR/LLBE/LLDQM出力信号の
    st.b    r6,     PCT[r0]         -- インアクティブ・レベル・ハイ設定
    mov     0xbf,   r6              -- BCYST,ASTB,WE/WR,RD,
                                    -- UUWR/UUBE/UUDQM,
                                    -- ULWR/ULBE/ULDQM,
                                    -- LUWR/LUBE/LUDQM,
                                    -- LLWR/LLBE/LLDQM出力端子選択
    st.b    r6,     PMCCT[r0]

    mov     0xc0,   r6              -- PCM5-PCM0は出力モード
    st.b    r6,     PMCM[r0]
    mov     0x00,   r6              -- PCM5-PCM0はすべて汎用ポート
    st.b    r6,     PMCCM[r0]

    mov     0x0c,   r6              -- SDRAS,SDCAS出力信号のインアクティブ・レベル・ハイ
    st.b    r6,     PCD[r0]         -- BUSCLK,SDCKE出力信号のインアクティブ・レベル・ロウ設定
    mov     0x0f,   r6              -- SDRAS,SDCAS,BUSCLK,SDCKE出力端子選択
    st.b    r6,     PMCCD[r0]

    mov     0x00,   r6              -- SRAM,外部ROM,外部I/Oサイクル時の
                                    -- IORD,IOWR動作禁止
    st.b    r6,     BCP[r0]         -- バス・サイクル・ピリオド・コントロール・レジスタ

#########################################################
#   バス制御端子以外の機能を指定
#########################################################

# 外部割り込み立ち上がりエッジ指定レジスタ1(INTR1)は初期値(00H)
# 外部割り込み立ち下がりエッジ指定レジスタ1(INTF1)は初期値(00H)
                                    -- INTP11,INTP10端子は立ち下がりエッジ入力
# 外部割り込み立ち上がりエッジ指定レジスタ2(INTR2)は初期値(00H)
# 外部割り込み立ち下がりエッジ指定レジスタ2(INTF2)は初期値(00H)
                                    -- INTP2n(n=1-5),NMI端子は立ち下がりエッジ入力
# 外部割り込み立ち上がりエッジ指定レジスタ6(INTR6)は初期値(20H)
# 外部割り込み立ち下がりエッジ指定レジスタ6(INTF6)は初期値(20H)
                                    -- INTP65端子は立ち上がり／立ち下がり両エッジ入力

    st.b    r0,     P1[r0]          -- TxD0出力信号のインアクティブ・レベル・ロウを設定
    mov     0x0f,   r6              -- P13-P10はすべて制御端子モード
    st.b    r6,     PMC1[r0]
    mov     0x0d,   r6              -- P13=TxD0出力,P12=RxD0入力,
                                    -- P11=INTP11入力,P10=UCLK入力
    st.b    r6,     PFC1[r0]
```

```
    mov     0x03,   r6              -- P25-P24(LED)のインアクティブ・レベル・ロウ(消灯)
    st.b    r6,     P2[r0]          -- P23(USB制御),TxD1端子にインアクティブ・レベル・ロウ
    mov     0xc7,   r6
    st.b    r6,     PM2[r0]         -- P25-P24(LED),P23(USB制御)出力モード
    mov     0x07,   r6
    st.b    r6,     PMC2[r0]        -- P25-P23汎用ポート,P22-P20は制御端子
    mov     0x06,   r6
    st.b    r6,     PFC2[r0]        -- P22=TxD1,P21=RxD1,P20=NMI

    mov     0x00,   r6              -- P55-P50(LED)のインアクティブ・レベル・ロウ(消灯)
    st.b    r6,     P5[r0]
    mov     0xc0,   r6
    st.b    r6,     PM5[r0]         -- P55-P50(LED)出力モード
    st.b    r0,     PMC5[r0]        -- P55-P50すべて汎用ポート

    mov     0xff,   r6
    st.b    r6,     PM6[r0]         -- P67-P66(TOGLE SW),P65(USB給電監視)入力モード
    mov     0x20,   r6
    st.b    r6,     INTR6[r0]
    st.b    r6,     INTF6[r0]
    st.b    r6,     PMC6[r0]        -- P67-P66汎用ポート,P65=INTP65

    mov     0xff,   r6
    st.b    r6,     P7[r0]
    st.b    r6,     PM7[r0]         -- P77-P72(TOGLE SW)入力モード
    st.b    r0,     PMC7[r0]        -- P77-P72すべて汎用ポート

#########################################################
# IIロック・レジスタ(LOCKR)のLOCKビットの確認
#########################################################
-- ( 0 )ロック・レジスタ(LOCKR)を読み出してPLLの安定を確認
_UNLOCK:
    ld.b    LOCKR[r0],r6            --ロック・レジスタ(LOCKR)よりPLLのロック状態を読み出し
    cmp     r0,     r6              -- まだフェーズ・ロックしてない？
    jnz     _UNLOCK                 -- はい

                                    -- PLLロック状態が確認できたので，次手順へ

-- ( i )システム・ウエイト・コントロール・レジスタ(VSWC)へx7Hを設定(x: I で設定した値を設定)
    mov     0x37,   r6              -- VSWC設定値上位4bitと下位4bitは7H(fx=133MHz)
    st.b    r6,     VSWC[r0]        -- システム・ウエイト・コントロール・レジスタに書き込み

-- ( ii)バス・モード・コントロール・レジスタ(BMC)へ外部バスの周波数分周値設定
    mov     0x01,   r6              -- 外部システム・クロック(fCLK)に対する
                                    -- バス・クロック(BUSCLK)の分周比はfCLK/2
    st.b    r6,     BMC[r0]         -- バス・モード・コントロール・レジスタに書き込み
```

```
-- (iii)システム・ウエイト・コントロール・レジスタ(VSWC)へ I で設定した値に再度設定
     mov     0x33,    r6                -- VSWC設定値の上位,下位を書き戻す(fx=133MHz)
     st.b    r6,      VSWC[r0]          -- システム・ウエイト・コントロール・レジスタに書き込み

-- ( iv)クロック・コントロール・レジスタ(CKC)へ内部システム・クロック周波数分周値設定
####    特定シーケンスによってのみ書き込み    ####
     mov     0xa0,    r6                -- ①NMI割り込み禁止
     ldsr    r6,      5                 --   PSWのNP bitを1に設定(NMI禁止)
     mov     0x03,    r6                -- ②汎用レジスタにCKCに設定するデータを用意
                                        --   PLLモード時の内部システム・クロック fCLK=fx
     st.b    r6,      PRCMD[r0]         -- ③コマンド・レジスタにダミー・データを書き込み
     st.b    r6,      CKC[r0]           -- ④クロック・コントロール・レジスタを設定
     nop                                -- ⑤ダミーNOP命令挿入(5命令発行)
     nop                                -- ⑥ダミーNOP命令挿入
     nop                                -- ⑦ダミーNOP命令挿入
     nop                                -- ⑧ダミーNOP命令挿入
     nop                                -- ⑨ダミーNOP命令挿入
     mov     0x20,    r6                -- ⑩NMI割り込み禁止を解除(NMI解除)
     ldsr    r6,      5                 --   PSWのNP bitを0に戻す
####    特定シーケンスによってのみ書き込み    ####

-- ( v )クロック・ソース選択レジスタ(CKS)へメイン・クロック(fx)供給制御を設定
####    特定シーケンスによってのみ書き込み    ####
     mov     0xa0,    r6                -- ①NMI割り込み禁止
     ldsr    r6,      5                 --   PSWのNP bitを1に設定(NMI禁止)
     mov     0x01,    r6                -- ②汎用レジスタにCKSに設定するデータを用意
                                        --   メイン・クロック(fx)供給をSSCG出力クロック(Fx×8)
     st.b    r6,      PRCMD[r0]         -- ③コマンド・レジスタにダミー・データを書き込み
     st.b    r6,      CKS[r0]           -- ④クロック・ソース選択レジスタを設定
     nop                                -- ⑤ダミーNOP命令挿入(5命令発行)
     nop                                -- ⑥ダミーNOP命令挿入
     nop                                -- ⑦ダミーNOP命令挿入
     nop                                -- ⑧ダミーNOP命令挿入
     nop                                -- ⑨ダミーNOP命令挿入
     mov     0x20,    r6                -- ⑩NMI割り込み禁止を解除(NMI解除)
     ldsr    r6,      5                 --   PSWのNP bitを0に戻す
####    特定シーケンスによってのみ書き込み    ####

# SSCGコントロール・レジスタ(SSCGC)は初期値(4xH:xについては下記参照)
####    特定シーケンスによってのみ書き込み    ####
--SMDL1=0,SMDL1=1:SSCG出力の変調周期26~35KHz
--JIT1端子 JIT0端子 ADJON ADJ2 ADJ1 ADJ0  SSCG出力の周波数変調率(Typ.値)
--  0       0       0    0    0    0       変調なし(周波数固定)
--  0       1       1    0    0    1       -1.0%
--  1       0       1    0    1    1       -3.0%
--  1       1       1    1    0    1       -5.0%
####    特定シーケンスによってのみ書き込み    ####
```

```
# USBクロック・コントロール・レジスタ(UCKC)
#     mov      0x80,    r6                -- USBへのクロック供給開始
#     st.b     r6,      UCKC[r0]

# 発振安定時間選択レジスタ(OSTS)は初期値(04H)
--OSTS0=0 ソフトウエアSTOPモード解除後の発振安定時間21.84ms(Fx=12MHz時)


_UNLOCK2:
      ld.b     LOCKR[r0],r6              --ロック・レジスタ(LOCKR)よりPLLのロック状態を読み出し
      cmp      r0,      r6               -- まだフェーズ・ロックしてない？
      jnz      _UNLOCK2                  -- はい

                                         -- PLLロック状態が確認できたので，次手順へ

#########################################################
#   SDRAMコンフィギュレーションの指定
#########################################################

### CS1(SDRAM 32bit幅/32MB)空間のリフレッシュ・インターバル設定 ##
      mov      0x801e, r6                -- リフレッシュ許可(BUSCLK=66MHz)
                                         -- リフレッシュ・カウント・クロック=32/BUSCLK
                                         -- リフレッシュ・インターバル・ファクタ=31(2.000MHz時15.0us)
      st.h     r6,      RFS1[r0]         -- SDRAMリフレッシュ・コントロール・レジスタ1

### CS1(SDRAM 32bit幅/32MB)空間のリフレッシュ環境設定 ##
      mov      0x20a5, r6                -- リード時CAS Latency 2 Latency(TLATE)
                                         -- ACT-RD,PRE-ACT時に2 wait(TBCW)
                                         -- アドレス・シフト幅          2 bit(On-page)
                                         --                          (外部データ・バス幅:32bit)
                                         -- ロウ・アドレス幅            12 bit
                                         -- アドレス・マルチプレクス幅  9 bit
      st.h     r6,      SCR1[r0]         -- SDRAMコンフィギュレーション・レジスタ1に書き込み
_SCR1NOFIX:
      ld.h     SCR1[r0], r6              -- SDRAMコンフィギュレーション・レジスタ1を読み出し
      movea    0x0100, r0,  r7           -- WCF1 bit抽出
      and      r7,      r6
      cmp      r6,      r7               -- SCR1設定未完了?
      jnz      _SCR1NOFIX                -- はい

### CS4(SDRAM 16bit幅/16MB)空間のリフレッシュ・インターバル設定 ##
      mov      0x801e, r6                -- リフレッシュ許可(BUSCLK=66MHz)
                                         -- リフレッシュ・カウント・クロック=32/BUSCLK
                                         -- リフレッシュ・インターバル・ファクタ=31(2.000MHz時15.0us)
      st.h     r6,      RFS4[r0]         -- SDRAMリフレッシュ・コントロール・レジスタ4
```

```
### CS4(SDRAM 16bit幅/16MB)空間のリフレッシュ環境設定 ##
     mov      0x2095, r6                -- リード時CAS Latency 2 Latency(TLATE)
                                        -- ACT-RD,PRE-ACT時に2 wait(TBCW)
                                        -- アドレス・シフト幅          1 bit(On-page)
                                        --                           (外部データ・バス幅:16bit)
                                        -- ロウ・アドレス幅            12 bit
                                        -- アドレス・マルチプレクス幅  9 bit
     st.h     r6,      SCR4[r0]         -- SDRAMコンフィギュレーション・レジスタ4
_SCR4NOFIX:
     ld.h     SCR4[r0], r6              -- SDRAMコンフィギュレーション・レジスタ4を読み出し
     movea    0x0100, r0,  r7           -- WCF4 bit抽出
     and      r7,      r6
     cmp      r6,      r7               -- SCR4設定未完了?
     jnz      _SCR4NOFIX                -- はい

### CS6(SDRAM  8bit幅/ 16MB)空間のリフレッシュ・インターバル設定 ##
     mov      0x801e, r6                -- リフレッシュ許可(BUSCLK=66MHz)
                                        -- リフレッシュ・カウント・クロック=32/BUSCLK
                                        -- リフレッシュ・インターバル・ファクタ=31(2.000MHz時15.0us)
     st.h     r6,      RFS6[r0]         -- SDRAMリフレッシュ・コントロール・レジスタ6

### CS6(SDRAM  8bit幅/ 16MB)空間のリフレッシュ環境設定 ##
     mov      0x2086, r6                -- リード時CAS Latency 2 Latency(TLATE)
                                        -- ACT-RD,PRE-ACT時に2 wait(TBCW)
                                        -- アドレス・シフト幅          0 bit(On-page)
                                        --                           (外部データ・バス幅: 8bit)
                                        -- ロウ・アドレス幅            12 bit
                                        -- アドレス・マルチプレクス幅 10 bit
     st.h     r6,      SCR6[r0]         -- SDRAMコンフィギュレーション・レジスタ6
_SCR6NOFIX:
     ld.h     SCR6[r0], r6              -- SDRAMコンフィギュレーション・レジスタ6を読み出し
     movea    0x0100, r0,  r7           -- WCF6 bit抽出
     and      r7,      r6
     cmp      r6,      r7               -- SCR6設定未完了?
     jnz      _SCR6NOFIX                -- はい

     jmp      [lp]
```

```
 [ledtest.s]
     .section ".ledtest",text
     .align   4
     .globl   _ledtest

     .extern  _ledtest_end
```

 # 初期化指標処理

 # Dot LED バイナリ・シフト

```
_ledtest:
     mov      0x01,    r11
     mov      0x8,     r12
_ledloop2:
     mov      r11,     r6
     jarl     _ledout, lp
     movhi    0x0001, r0,  r6
     movea    0x0000, r6,  r6
     jarl     _softwait, lp

     shl      0x1,     r11
     add      -1,      r12
     cmp      r0,      r12
     bnz      _ledloop2
     jr       _ledtest_end
```

 # Dot LED バイナリ出力処理

```
     .globl   _ledout
_ledout:
     mov      r6,      r10
     shr      0x02,    r10
     and      0x3f,    r10
     st.b     r10,     P5
     mov      r6,      r10
     shl      0x04,    r10
     and      0x30,    r10
     st.b     r10,     P2

     jmp      [lp]
```

 # ソフトウエア・アイドル・ループ処理

```
     .globl   _softwait
_softwait:
     add      -1,      r6
     cmp      r0,      r6
     bnz      _softwait
     jmp      [lp]
```

```
 [printf.c]
#include <stdio.h>
#include <stdarg.h>

#include "uart.h"
#define TRUE  1
#define FALSE 0

/* グローバル変数定義 */
static volatile unsigned int    TxCh1; /* ポーリング送信フラグ */

/*****************************************************************************
 *
 *  関数名: init_uartb1
 *  機　能: UARTB1の初期化
 *  引　数: bps 転送速度(uart.h参照)
 *  戻り値: なし
 *
 *****************************************************************************/
void init_uartb1(int bps)
{
   int i;

   TxCh1 = FALSE;

   /* UARTB1関連端子初期化 */
   P2     = 0x03;
   PM2   |= 0xc7;
   PMC2  |= 0x07;
   PFC2  |= 0x06;

   /* ボー・レート設定 */
   switch(bps){
   case UART_115200BPS:
          UB1CTL2 = UART_BAUD_115200;
          break;
   case UART_57600BPS:
          UB1CTL2 = UART_BAUD_57600;
          break;
   case UART_38400BPS:
          UB1CTL2 = UART_BAUD_38400;
          break;
   case UART_19200BPS:
          UB1CTL2 = UART_BAUD_19200;
          break;
   case UART_9600BPS:
          UB1CTL2 = UART_BAUD_9600;
          break;
```

```
   case UART_4800BPS:
          UB1CTL2 = UART_BAUD_4800;
          break;
   case UART_2400BPS:
          UB1CTL2 = UART_BAUD_2400;
          break;
   default:
          /* default 9600bps */
          UB1CTL2 = UART_BAUD_9600;
          break;
   }

   UB1CTL0  = UART_PWR_OFF;    /* UARTB1を非同期にリセット */
   for(i = 0 ; i < 10 ; i++); /* fx/4の2周期分以上経過 */
   UB1CTL0 |= UART_PWR_ON;     /* UARTB1をへクロック供給 */
   /* 送受信許可,LSB先頭,Parityなし, StopBit 1,8bit長 */
   UB1CTL0 |= (UART_TXE_ON | UART_RXE_ON | UART_CL_8);

}

/*****************************************************************************
 *
 *  関数名: write_ch1
 *  機　能: UARTB1で1バイト送信
 *  引　数: ch送信する値
 *  戻り値: なし
 *
 *****************************************************************************/
void write_ch1(unsigned char ch)
{
   volatile unsigned char stic1;
   volatile int  delay;
   /* 最初のデータの送信漏れがないように送信レジスタの空きをチェック */
   stic1 = UTIC1;
   if(!TxCh1 && (stic1 & 0x80))
          UTIC1 = stic1 ^ 0x80;
   /* 送信レジスタの空き待ち */
   if (TxCh1){
          while (1){
            stic1 = UTIC1;
            if (stic1 & 0x80){
                 /* 送信割り込み要求クリア */
                 UTIC1 = stic1 ^ 0x80;
                 TxCh1 = FALSE;
                 break;
            }
          }
   }
```

```
   /* データ送信 */
   if (!TxCh1){
          UB1TX = ch;
          TxCh1 = TRUE;
   }
}

/*****************************************************************************
 *
 *  関数名: write_str1
 *  機　能: UARTB1で文字列を送信
 *  引　数: str送信する文字列(NULLで終了していなくてはならない)へのポインタ
 *  戻り値: なし
 *
 *****************************************************************************/
void write_str1( char *str)
{
   while (*str != (char)NULL){
          write_ch1(*str);
          str++;
   }
}
/*****************************************************************************
 *
 *  関数名: write_data1
 *  機　能: UARTB1でデータを送信
 *  引　数: str 送信するデータへのポインタ
 *          len データ長
 *  戻り値: なし
 *
 *****************************************************************************/
void write_data1(char *str, int len)
{
   int i;
   for (i = 0; i < len; i++){
          write_ch1((unsigned char)*str);
          str++;
   }
}
```

```
/************************************************************************
 *
 *  関数名: send_message
 *  機　能: UARTB1を使ってメッセージを送信する
 *  引　数: strメッセージ文字列(NULLで終了する必要あり)へのポインタ
 *  戻り値: なし
 *
 ************************************************************************/
void send_message(char *str)
{
  write_str1(str);
}

/************************************************************************
 *
 *  関数名: printf
 *  機　能: UARTB1を使ってprintf出力する
 *  引　数: 通常のprintfと同じ
 *  戻り値: 通常のprintfと同じ
 *
 ************************************************************************/
int
printf(const char *fmt, ...)
{
  char buf[256];
  va_list ap;
  int ret = 0;

  /* 引数リスト走査用変数の初期化 */
  va_start(ap, fmt);

  /* 書式付データをバッファへ出力 */
  ret = vsprintf(buf, fmt, ap);

  /* 引数リスト走査の終了 */
  va_end(ap);

  /* UARTB1でバッファの中身を出力 */
  send_message(buf);

  return ret;
}
```

[uart.h]

```
/*---------------------------------------------------------------------*/
/* 転送速度定義                                                        */
/*---------------------------------------------------------------------*/
#define UART_115200BPS  115200
#define UART_57600BPS   57600
#define UART_38400BPS   38400
#define UART_19200BPS   19200
#define UART_9600BPS    9600
#define UART_4800BPS    4800
#define UART_2400BPS    2400

#if 0
/* UB0CTL1,UB0CTL2 設定値 fx=133MHz,Clock=fx/4基準 */
#define UART_BAUD_2400    6927    /* 誤差 +0.001% */
#define UART_BAUD_4800    3464    /* 誤差 -0.013% */
#define UART_BAUD_9600    1732    /* 誤差 -0.013% */
#define UART_BAUD_19200   866     /* 誤差 -0.013% */
#define UART_BAUD_38400   433     /* 誤差 -0.013% */
#define UART_BAUD_57600   289     /* 誤差 -0.128% */
#define UART_BAUD_115200  144     /* 誤差 +0.218% */
#else
/* UB0CTL1,UB0CTL2 設定値 fx= 96MHz,Clock=fx/4基準 */
#define UART_BAUD_2400    5000    /* 誤差 -0.000% */
#define UART_BAUD_4800    2500    /* 誤差 -0.000% */
#define UART_BAUD_9600    1250    /* 誤差 -0.000% */
#define UART_BAUD_19200   625     /* 誤差 -0.000% */
#define UART_BAUD_38400   313     /* 誤差 -0.160% */
#define UART_BAUD_57600   208     /* 誤差 +0.160% */
#define UART_BAUD_115200  104     /* 誤差 +0.160% */
#endif

/*---------------------------------------------------------------------*/
/* UARTBn制御レジスタ0(UBnCTL0)ビット定義                              */
/*---------------------------------------------------------------------*/
#define UART_PWR_ON  (1 << 7)     /* UARTへのクロック供給停止 */
#define UART_PWR_OFF (0 << 7)     /* UARTへのクロック供給開始 */

#define UART_TXE_ON  (1 << 6)     /* 送信許可 */
#define UART_TXE_OFF (0 << 6)     /* 送信禁止 */

#define UART_RXE_ON  (1 << 5)     /* 受信許可 */
#define UART_RXE_OFF (0 << 5)     /* 受信禁止 */

#define UART_DIR_LSB (1 << 4)     /* 転送データ先頭ビットはLSB */
#define UART_DIR_MSB (0 << 4)     /* 転送データ先頭ビットはMSB */
```

```
#define UART_PS_NO   (0 << 3) | (0 << 2)        /* パリティなし */
#define UART_PS_0    (0 << 2) | (1 << 2)        /* パリティ0 */
#define UART_PS_ODD  (1 << 3) | (0 << 2)        /* 奇数パリティ */
#define UART_PS_EVN  (1 << 3) | (1 << 2)        /* 偶数パリティ */

#define UART_CL_8    (1 << 1) /* 転送データ1フレームは8ビット */
#define UART_CL_7    (0 << 1) /* 転送データ1フレームは7ビット */

#define UART_SL_2    (1 << 0) /* 転送データ・ストップ・ビットは2ビット */
#define UART_SL_1    (0 << 0) /* 転送データ・ストップ・ビットは1ビット */
```

[crtE.s]

#（**著作権表示等省略**）

```
#------------------------------------------------------------------------------
#       special symbols
#------------------------------------------------------------------------------
     .extern _ _tp_TEXT, 4
     .extern _ _gp_DATA, 4
     .extern _ _ep_DATA, 4
     .extern _ _ssbss, 4
     .extern _ _esbss, 4
     .extern _ _sbss, 4
     .extern _ _ebss, 4

#------------------------------------------------------------------------------
#        C program main function
#------------------------------------------------------------------------------
     .extern _main

#------------------------------------------------------------------------------
#   for argv (not used, So deleted data definitions)
#------------------------------------------------------------------------------

#------------------------------------------------------------------------------
#       dummy data declaration for creating sbss section
#------------------------------------------------------------------------------
     .sbss
     .lcomm  _ _sbss_dummy,     0,      0

#------------------------------------------------------------------------------
#       system stack
#------------------------------------------------------------------------------
     .set    STACKSIZE, 0x200
     .globl  _ _stack
     .bss
     .lcomm  _ _stack,     STACKSIZE,4


#------------------------------------------------------------------------------
#       RESET handler
#------------------------------------------------------------------------------
     .section "RESET", text
     jr      _ _start
```

```
#------------------------------------------------------------------------------
#       start up
#               pointers: tp - text pointer
#                         gp - global pointer
#                         sp - stack pointer
#                         ep - element pointer
#               mask reg: r20 - 0xff
#                         r21 - 0xffff
#       exit status is set to r10
#------------------------------------------------------------------------------
.section ".crtE",text
     .align  4
     .globl  _ _start
     .globl  _ _exit
     .globl  _ _startend
#    .extern _ _PROLOG_TABLE
_ _start:
     jarl    _initial,lp              -- V850E/ME2 CPU initial
_initial_end:

     jr      _ledtest
     .globl  _ledtest_end
_ledtest_end:

     mov     #_ _tp_TEXT, tp          -- set tp register
     mov     #_ _gp_DATA, gp          -- set gp register offset
     add     tp,     gp               -- set gp register
     mov     #_ _stack+STACKSIZE, sp  -- set sp register
     mov     #_ _ep_DATA, ep          -- set ep register
#
     .option nowarning
     mov     0xff,   r20              -- set mask register
     mov     0xffff, r21              -- set mask register
     .option warning
#
     mov     #_ _ssbss, r13           -- clear sbss section
     mov     #_ _esbss, r12
     cmp     r12,    r13
     jnl     .L11
.L12:
     st.w    r0,     [r13]
     add     4,      r13
     cmp     r12,    r13
     jl      .L12
.L11:
#
     mov     #_ _sbss, r13            -- clear bss section
     mov     #_ _ebss, r12
     cmp     r12,    r13
     jnl     .L14
```

```
.L15:
     st.w     r0,      [r13]
     add      4,       r13
     cmp      r12,     r13
     jl       .L15
.L14:

#########################################################
# 内蔵命令RAMにプログラム・コードを転送
#########################################################
     .extern  _ _S_rompack
     .extern  _ _S_appli1rom
     .extern  _ _ircopy
     .globl   _iramboot
_iramboot:
-- ( 0 )内蔵命令RAMへProgramを転送する
.option nowarning
     movhi    hi1(_ _S_rompack), tp, r1          -- tp のオフセットをかける
     movea    lo(_ _S_rompack), r1, r6           -- 静的プログラム・パッキング・セクション先頭アドレス
.option warning
     mov      -0x1,    r7                        -- 転送したいパッキング・セクションはすべて
     jarl     _ _ircopy,r31                      -- ROM化された動的プログラムをBANK0へ転送

.extern _ _S_appli1rom
.option nowarning
     movhi    hi1(_ _S_appli1rom), tp, r1        -- tp のオフセットをかける
     movea    lo(_ _S_appli1rom), r1, r6         -- 動的プログラム1パッキング・セクション先頭アドレス
.option warning
     mov      -0x1,    r7                        -- 転送したいパッキング・セクションはすべて
     jarl     _ _ircopy,r31                      -- ROM化された動的プログラムをBANK1へ転送

                                                 -- プログラム・コードが転送できたので
                                                 -- 内蔵命令RAMをリード・モードに切り替える

-- ( i )内蔵命令RAMモード・レジスタ(IRAMM)をリード・モード設定(IRAMM0,1=0)
     mov      0x00,    r6                        -- BANK0 0000000-00FFFFH(64KB)リード・モード
                                                 -- BANK1 0010000-01FFFFH(64KB)リード・モード
     st.b     r6,      IRAMM[r0]                 -- 内蔵命令RAMモード・レジスタ

-- ( ii )内蔵命令RAMモード・レジスタ(IRAMM)のリード・モード変更を確認(IRAMM0=1?)
_iram_rd_chk:
     ld.b     IRAMM[r0], r6                     -- 内蔵命令RAMモード・レジスタ読み出し
     cmp      r0,      r6                       -- BANK0,1はリード・モード?
     bnz      _iram_rd_chk                      -- いいえ
```

```
-- (iii)分岐命令により内蔵命令RAMへ
.option nowarning
     mov      #_main, r1
     jmp      [r1]
.option warning
_ _exit:
     halt                              -- end of program
_ _startend:
     nop
     nop
#                                                                    #
#-------------------- end of start up module -------------------#
#                                                                    #
```

```
 [secentry.s]
      .file    "secentry.s"

# 外部EPROM領域から転送した動的にプログラムを動かす内蔵命令RAM(BANK1)領域
      .section ".dymcappli",text
      .align   4
      .globl   _ _S_dymcappli, 4
_ _S_dymcappli:

# 動的に動かしたいプログラム1が格納されている外部EPROM領域1
      .section ".appli1rom",text
      .align   4
      .globl   _ _S_appli1rom, 4
_ _S_appli1rom:

# 動的に動かしたいプログラム2が格納されている外部EPROM領域2
      .section ".appli2rom",text
      .align   4
      .globl   _ _S_appli2rom, 4
_ _S_appli2rom:

# 動的に動かしたいプログラム3が格納されている外部EPROM領域3
      .section ".appli3rom",text
      .align   4
      .globl   _ _S_appli3rom, 4
_ _S_appli3rom:
```

```
 [rompack.s]
      .file    "rompack.s"
      .section ".s_romp",text
      .align   4
      .globl   _ _S_rompack, 4
_ _S_rompack:
```

```
 [progmain.dir]
#(著作権表示等省略)
#静的に動作するプログラム・セグメント領域
#内蔵命令RAM領域BANK0 0x00001000-0x0000ffff( 60KB)
MAINTEXT    : !LOAD ?RWX V0x00001000{
     .text        = $PROGBITS   ?AX .text;
};

#静的に動作するプログラムのCONSTセグメント領域
#内蔵命令RAM領域BANK0 0x00001000-0x0000ffff( 60KB)
CONST   : !LOAD ?R V0x0000f000 {
     .const       = $PROGBITS   ?A .const;
};

#動的に動作するプログラム・セグメント領域
#内蔵命令RAM領域BANK1 0x00010000-0x0001ffff( 64KB)
DYMCAPPL  : !LOAD ?RWX V0x00014000 {
     .dymcappli   = $PROGBITS   ?AX A0x4 .dymcappli;
};

#CPU RESETセグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
RESET    : !LOAD ?RX V0x00100000 {
     RESET        = $PROGBITS   ?AX RESET0;
};


#スタートアップ・モジュール・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
CRTE       : !LOAD ?RX V0x00101000 {
     .crtE        = $PROGBITS   ?AX .crtE;
};

#内蔵命令RAM領域転送プログラム・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
IRCOPY    : !LOAD ?RX  {
     .ircopy      = $PROGBITS   ?AX .ircopy;
};

#CPU初期化プログラム・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
INITIAL : !LOAD ?RX {
     .initial     = $PROGBITS   ?AX .initial;
};

#LED点灯プログラム・セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
LEDTEST : !LOAD ?RX {
     .ledtest     = $PROGBITS   ?AX .ledtest;
};
```

```
#rompsec seciton セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
SROMP       : !LOAD ?RX  {
     .s_romp      = $PROGBITS   ?AX .s_romp;
};

#動的に動作するプログラム1を格納する空セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
APPLI1ROM   : !LOAD ?RX V0x00114000 {
     .appli1rom   = $PROGBITS   ?AX A0x4 .appli1rom;
};

#動的に動作するプログラム2を格納する空セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
APPLI2ROM   : !LOAD ?RX V0x00118000 {
     .appli2rom   = $PROGBITS   ?AX A0x4 .appli2rom;
};

#動的に動作するプログラム3を格納する空セグメント領域
#外部EPROM(16bit幅)領域 0x100000-0x11ffff(128KB)
APPLI3ROM   : !LOAD ?RX V0x0011c000 {
     .appli3rom   = $PROGBITS   ?AX A0x4 .appli3rom;
};

#内蔵データRAMセグメント領域 0x0fffb000-0x00fffefff(20KB)--V850E/ME2
#内蔵データRAM予約領域 0xfff8000-0xfffafff(12KB)(アクセス禁止)
#内蔵データRAM実装領域 0xfffb000-0xfffcfff( 8KB)(ROM Monitor予約)
DATA  : !LOAD ?RW V0x0fffd000 {
     .data            = $PROGBITS  ?AW  .data;
     .sdata           = $PROGBITS  ?AWG .sdata;
     .sbss            = $NOBITS    ?AWG .sbss;
     .bss             = $NOBITS    ?AW  .bss;
};

_ _tp_TEXT @ %TP_SYMBOL;
_ _gp_DATA @ %GP_SYMBOL &_ _tp_TEXT{DATA};
_ _ep_DATA @ %EP_SYMBOL;
```

[ircopy.s]

4. 2. 8 **(6) プログラム例参照**

(b) 動的プログラム・リスト

次に動的プログラム・リストのソース・ファイル一覧と動的プログラム・リストを示します。

注意　動的プログラム・リストでは，動的プログラム1の場合のみ掲載しています。他のプログラムとの違いは，main関数内の回転リールの桁とINT SW割り込み内でリールを止める処理，およびROM化時のリンク・アドレスが異なるだけです。詳細については，4. 2. 10（1）**動的プログラム・サンプルの実行概要を参照してください。**

表4－12　動的プログラム・リストのソース・ファイル一覧

ファイル名	概　　要
appli1.c	wait関数，main関数，intsw_for_app1関数，seg7led関数，dotled関数
secentry.s	空リンク・エントリ
printf.c	UARTB1出力処理関数群（静的プログラムと同じ）
uart.h	UARTB関連ヘッダ・ファイル（静的プログラムと同じ）
appli1.h	動的プログラム用ヘッダ・ファイル
crtE.s	動的プログラム用スタートアップ・モジュール処理
progapp1.dir	動的プログラム用リンク・ディレクティブ

```
 [appli1.c]
#include "appli1.h"
#pragma ioreg   /* 内蔵I/O使用宣言 */

/* 動的実行するプログラムが格納されている外部EPROM領域アドレス名外部宣言 */
extern void _S_app1entry();

/* グローバル変数 */
unsigned char place;                    /* 回転リール表示位置の指定 */
unsigned char flag;                     /* 割り込み処理トグル・フラグ */

/* プロトタイプ宣言 */
void intsw_for_app1(void);
void port_init(void);
void dotled(int n, int op);
/************************************************************************
 *
 *  関数名: wait
 *  機　能: アイドル・ループ(WAIT定義数分アイドル)
 *  引　数: なし
 *  戻り値: なし
 *
 ************************************************************************/
void wait(void)
{
   int a=0;
   while(a!=10000)
   {
      a=a+1;
   }
}

/************************************************************************
 *
 *  関数名: main
 *  機　能: 動的プログラム1 メイン処理
 *  引　数: なし
 *  戻り値: なし
 *
 ************************************************************************/
void main(void){
     int i;
     int bcd02 = 0,bcd01 = 0;           /* 回転リール変数 */

     init_uartb1(19200);                /* UARTB1 channel 初期化 */
     port_init();                       /* ポート初期化 */
```

```
    /* INTP11割り込み制御レジスタ設定 */
    _ _set_il(0, "INTP11);              /* INTP11割り込みマスク解除 */
    _ _set_il(3, "INTP11);              /* INTP11割り込み優先順位4 */

    /* 内蔵命令RAM BANK0(0x00000000-0x0000ffff)をライト・モードへ */
    IRAMM = 0x01;/* 内蔵命令RAMモード・レジスタ(IRAMM)をライト・モード(IRAMM0=1) */
    do{
     ; /* 内蔵命令RAMモード・レジスタ(IRAMM)のライト・モード変更を確認(IRAMM0=1?) */
    }while(IRAMM != 1);

    for(i = 1 ; i <= USEINTMAX ; i++){
     _ircopy(_S_app1entry,i); /* 動的プログラムで使用する割り込みハンドラ書き込み */
    }

    /* 内蔵命令RAM BANK0(0x00000000-0x0000ffff)をリード・モードへ */
    IRAMM = 0x00;/* 内蔵命令RAMモード・レジスタ(IRAMM)をリード・モード(IRAMM0=0) */
    do{
     ; /* 内蔵命令RAMモード・レジスタ(IRAMM)のリード・モード変更を確認(IRAMM0=0?) */
    }while(IRAMM != 0);

    /* 割り込み許可 */
    _ _EI();

    printf("¥r¥nRunning application No.1¥r¥n");/* 動的プログラムのID番号を表示 */
    for(;;){
       wait();
       if((place & DIGIT1) == GO)bcd01++;      /* 指定桁の回転は続行か? */
       if(bcd01 > 9)bcd01=0;                   /* 回転リールの数字は0-9でループ */
       if((place & DIGIT2) == GO)bcd02++;      /* 指定桁の回転は続行か? */
       if(bcd02 > 9)bcd02=0;                   /* 回転リールの数字は0-9でループ */
       /* 8桁のうち2桁回転 */
       printf("¥r<pseudo 7segLED>=000000%d%d",bcd02,bcd01);
    }
}

/*****************************************************************************
 *
 *  関数名: port_init
 *  機　能: 汎用PORT初期化処理
 *  引　数: なし
 *  戻り値: なし
 *
 *****************************************************************************/
void port_init (void){
    /* INTSW用端子初期化 */
    PMC1 = 0x0f;   /* P13-P10はすべて制御端子モード */
    PFC1 = 0x0d;   /* P13=TxD0,P12=RxD0,P11=INTP11,P10=UCLK入力 */
```

```
    /* Dot LED用ポート初期化 */
    P2    = 0x03;                       /* LED1-2(P25-P24)を消灯 */
                                        /* P23(USB制御),TxD1端子にインアクティブ・レベル・ロウ */
    PM2  |= 0xc7;                       /* P25-P24=Dot LED,P23(USB制御)を出力モード */
    PMC2 |= 0x07;                       /* P25-P23を入出力ポート */
    PFC2 |= 0x06;                       /* P22=TxD1,P21=RxD1,P20=NMI */

    P5    = 0x00;                       /* LED3-8(P55-P50)を消灯 */
    PM5  |= 0xc0;                       /* P55-P50=Dot LEDを出力モード */
    PMC5 |= 0x00;                       /* P55-P50を入出力ポート */

    /* TOGLE SW用ポート初期化 */
    PM6  |= 0xff;                       /* P67-P66=TOGLE SW,P65(USB給電監視)を入力モード */
    PMC6 |= 0x20;                       /* P67-P66を入出力ポート,P65=INTP65 */

    PM7  |= 0xff;                       /* P77-P72=TOGLE SWを入力モード */
    PMC7 |= 0x00;                       /* P77-P72を入出力ポート */

}

/************************************************************************
 *
 *  関数名: seg7led
 *  機　能: 回転リール 操作処理
 *  引　数: *p リール操作桁位置変数へのポインタ
 *          op リール停止／動作
 *  戻り値: なし
 *
 ************************************************************************/
void seg7led( unsigned char *p, int n, int op ){
    if(op == STOP){
       *p |=  n;                        /* 回転リール n桁目の表示を止める */
    }else{
       *p &= ~n;                        /* 回転リール n桁目の表示を動かす */
    }
}
```

```
/**** Dot LED 操作処理 ****/
/************************************************************************
 *
 *  関数名：dotled
 *  機　能：回転リール 操作処理
 *  引　数：pattern DotLEDの位置
 *          op      DotLEDの点灯／消灯
 *  戻り値：なし
 *
 ************************************************************************/
void dotled(int pattern, int op)
{
     if(op == ALLON){
        DOTLED2_1 = ((DOTLED2 | DOTLED1) << 4) & 0x30;
        DOTLED8_3 = ((DOTLED8 | DOTLED7 | DOTLED6 | DOTLED5 |
                        DOTLED4 | DOTLED3 & 0xfc) >> 2) & 0x3f;
     }else if(op == ON){
        DOTLED2_1 =  ((pattern & 0x03) << 4) & 0x30;
        DOTLED8_3 =  ((pattern & 0xfc) >> 2) & 0x3f;
     }else if(op == OFF){
        DOTLED2_1 = ~((pattern & 0x03) << 4) & 0x30;
        DOTLED8_3 = ~((pattern & 0xfc) >> 2) & 0x3f;
     }else if(op == ALLOFF){
        DOTLED2_1 = ~((DOTLED2 | DOTLED1) << 4) & 0x30;
        DOTLED8_3 = ~((DOTLED8 | DOTLED7 | DOTLED6 | DOTLED5 |
                        DOTLED4 | DOTLED3 & 0xfc) >> 2) & 0x3f;
     }
}

/************************************************************************
 *
 *  割り込み要求名：INTP11
 *  機　　能：動的プログラムで稼動するINTSW割り込み処理
 *
 ************************************************************************/
#pragma interrupt INTP11 intsw_for_app1
_ _interrupt
void intsw_for_app1(void){
     if(flag ^= ALTERNATIVE){  /* flagは割り込む毎に交互に0/1反転する */
        /* 回転リール1桁目停止,2桁目回転 */
        seg7led(&place, DIGIT1, STOP);     /* 回転リール1桁目の表示停止 */
        seg7led(&place, DIGIT2, GO);       /* 回転リール2桁目の表示動作 */
        dotled(DOTLED1, ALLOFF);           /* Dot LED 全消灯 */
        dotled(DOTLED1, ON);               /* Dot LED1 点灯 */
```

```
     }else{
        /* 回転リール2桁目停止,1桁目回転 */
        seg7led(&place, DIGIT2, STOP);     /* 回転リール2桁目の表示停止 */
        seg7led(&place, DIGIT1, GO);       /* 回転リール1桁目の表示動作 */
        dotled(DOTLED2, ALLOFF);           /* Dot LED 全消灯 */
        dotled(DOTLED2, ON);               /* Dot LED2 点灯 */
    }
}
```

[secentry.s]

```
     .file    "secentry.s"
     .section ".app1entry",text
     .align   4
     .globl   _ _S_app1entry, 4
_ _S_app1entry:
```

[printf.c]

静的プログラムと同じ。

[uart.h]

静的プログラムと同じ。

[appli1.h]

```
/* 定数定義 */
#define USEINTMAX 1                     /* 動的プログラムで使用する割り込みの最大値 */
#define HIGH 1                          /* ハイ・レベル */
#define LOW 0                           /* ロウ・レベル */
#define ACTIVE_LEVEL HIGH               /* アクティブ・レベル選択 */
#define ALTERNATIVE 0x01                /* 割り込みで使用するトグル・フラグ */
#define DIGIT1 0x01                     /* 回転リールの1桁目位置 */
#define DIGIT2 0x02                     /* 回転リールの2桁目位置 */
#define DIGIT3 0x04                     /* 回転リールの3桁目位置 */
#define DIGIT4 0x08                     /* 回転リールの4桁目位置 */
#define DIGIT5 0x10                     /* 回転リールの5桁目位置 */
#define DIGIT6 0x20                     /* 回転リールの6桁目位置 */
#define DIGIT7 0x40                     /* 回転リールの7桁目位置 */
#define DIGIT8 0x80                     /* 回転リールの8桁目位置 */

#define DOTLED1 0x01                    /* Dot LEDの1桁目位置 */
#define DOTLED2 0x02                    /* Dot LEDの2桁目位置 */
#define DOTLED3 0x04                    /* Dot LEDの3桁目位置 */
#define DOTLED4 0x08                    /* Dot LEDの4桁目位置 */
#define DOTLED5 0x10                    /* Dot LEDの5桁目位置 */
#define DOTLED6 0x20                    /* Dot LEDの6桁目位置 */
#define DOTLED7 0x40                    /* Dot LEDの7桁目位置 */
#define DOTLED8 0x80                    /* Dot LEDの8桁目位置 */

#define DOTLED8_3 P5                    /* Dot LEDの8-3桁目に割り付けられたポート */
#define DOTLED2_1 P2                    /* Dot LEDの2-1桁目に割り付けられたポート */

#define STOP 1                          /* 回転リールの停止 */
#define GO 0                            /* 回転リールの回転 */

#define ALLON 3                         /* LED全点灯 */
#define ON 1                            /* LED点灯 */
#define OFF 0                           /* LED消灯 */
#define ALLOFF -1                       /* LED全消灯 */

#define HANDLER_INTSW 0x090             /* NMI割り込みハンドラ・アドレス */
```

[crtE.s]

#（**著作権表示等は省略**）

```
#------------------------------------------------------------------------------
#   special symbols
#------------------------------------------------------------------------------
     .extern _ _tp_TEXT, 4
     .extern _ _gp_DATA, 4
     .extern _ _ep_DATA, 4
     .extern _ _ssbss, 4
     .extern _ _esbss, 4
     .extern _ _sbss, 4
     .extern _ _ebss, 4

#------------------------------------------------------------------------------
#   C program main function
#------------------------------------------------------------------------------
     .extern _main

#------------------------------------------------------------------------------
#   for argv (not used, So deleted data definitions)
#------------------------------------------------------------------------------

#------------------------------------------------------------------------------
#   dummy data declaration for creating sbss section
#------------------------------------------------------------------------------
     .sbss
     .lcomm  _ _sbss_dummy, 0, 0

#------------------------------------------------------------------------------
#   system stack
#------------------------------------------------------------------------------
     .set    STACKSIZE,0x200
     .bss
     .lcomm  _ _stack, STACKSIZE, 4
```

```
#------------------------------------------------------------------------------
#   RESET handler(not used,So deleted codes)
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#       start up
#               pointers: tp - text pointer
#                         gp - global pointer
#                         sp - stack pointer
#                         ep  - element pointer
#               mask reg: r20 - 0xff
#                         r21 - 0xffff
#       exit status is set to r10
#------------------------------------------------------------------------------
     .text
     .align  4
     .globl  _ _start
     .globl  _ _exit
     .globl  _ _startend
_ _start:
     .option nowarning
     movhi   hi1(#_ _tp_TEXT), r0, r1          -- set tp register
     movea   lo(#_ _tp_TEXT), r1, tp           -- set tp register
     movhi   hi1(#_ _gp_DATA), r0, r1          -- set gp register offset
     movea   lo(#_ _gp_DATA), r1, gp           -- set gp register offset
     .option warning
     add     tp, gp                            -- set gp register
     .option nowarning
     movhi   hi1(#_ _stack+STACKSIZE),r0, r1   -- set sp register
     movea   lo(#_ _stack+STACKSIZE), r1, sp   -- set sp register
     movhi   hi1(#_ _ep_DATA), r0, r1          -- set ep register
     movea   lo(#_ _ep_DATA), r1, ep           -- set ep register
     .option warning
#
     .option nowarning
     mov     0xff,   r20                       -- set mask register
     mov     0xffff, r21                       -- set mask register
     .option warning
#
     mov #_ _ssbss, r13                        -- clear sbss section
     mov #_ _esbss, r12
     cmp r12, r13
     jnl .L11
.L12:
     st.w    r0,     [r13]
     add     4,      r13
     cmp     r12,    r13
     jl      .L12
.L11:
#
     mov #_ _sbss, r13                                 -- clear bss section
```

```
     mov #_ _ebss, r12
     cmp r12, r13
     jnl .L14
.L15:
     st.w    r0,      [r13]
     add     4,       r13
     cmp     r12,     r13
     jl      .L15
.L14:

     jarl    _main,  lp                      -- call main function
_ _exit:
     halt                                    -- end of program
_ _startend:
             nop
             nop
#                                                                  #
#-------------------- end of start up module -------------------#
#                                                                  #
```

```
 [progapp1.dir]
#(著作権表示等省略)

#内蔵命令RAM BANK1空間   0x00014000-0x00016fff( 12KB)
#動的プログラム群実行領域
TEXT       : !LOAD ?RWX V0x00014000 {
     .text        = $PROGBITS   ?AX .text;
};

#内蔵命令RAM BANK1空間   0x00017000-0x00017fff(  4KB)
#動的実行プログラム定数データ格納領域
CONST1     : !LOAD ?R   V0x00017000 {
     .const       = $PROGBITS   ?A .const;
};

#外部EPROM(16bit幅)空間 0x00114000-0x0016ffff( 12KB)
#動的実行プログラム1格納領域
APPLI1ENTRY     : !LOAD ?RX V0x00114000 {
     .app1entry   = $PROGBITS   ?AX .app1entry;
};

★   #内蔵データRAM空間 0x0fffb000-0x00fffefff( 20KB)--V850E/ME2 uPD703111A
    #内蔵データRAM空間 0x0fffb000-0x00fffcfff(  8KB)--V850E/ME2 ROM Monitor 予約領域
DATA    : !LOAD ?RW V0x0fffd800 {
     .data        = $PROGBITS   ?AW .data;
     .sdata       = $PROGBITS   ?AWG .sdata;
     .sbss        = $NOBITS     ?AWG .sbss;
     .bss         = $NOBITS     ?AW .bss;
};

_ _tp_TEXT @ %TP_SYMBOL;
_ _gp_DATA @ %GP_SYMBOL &_ _tp_TEXT{DATA};
_ _ep_DATA @ %EP_SYMBOL;
```

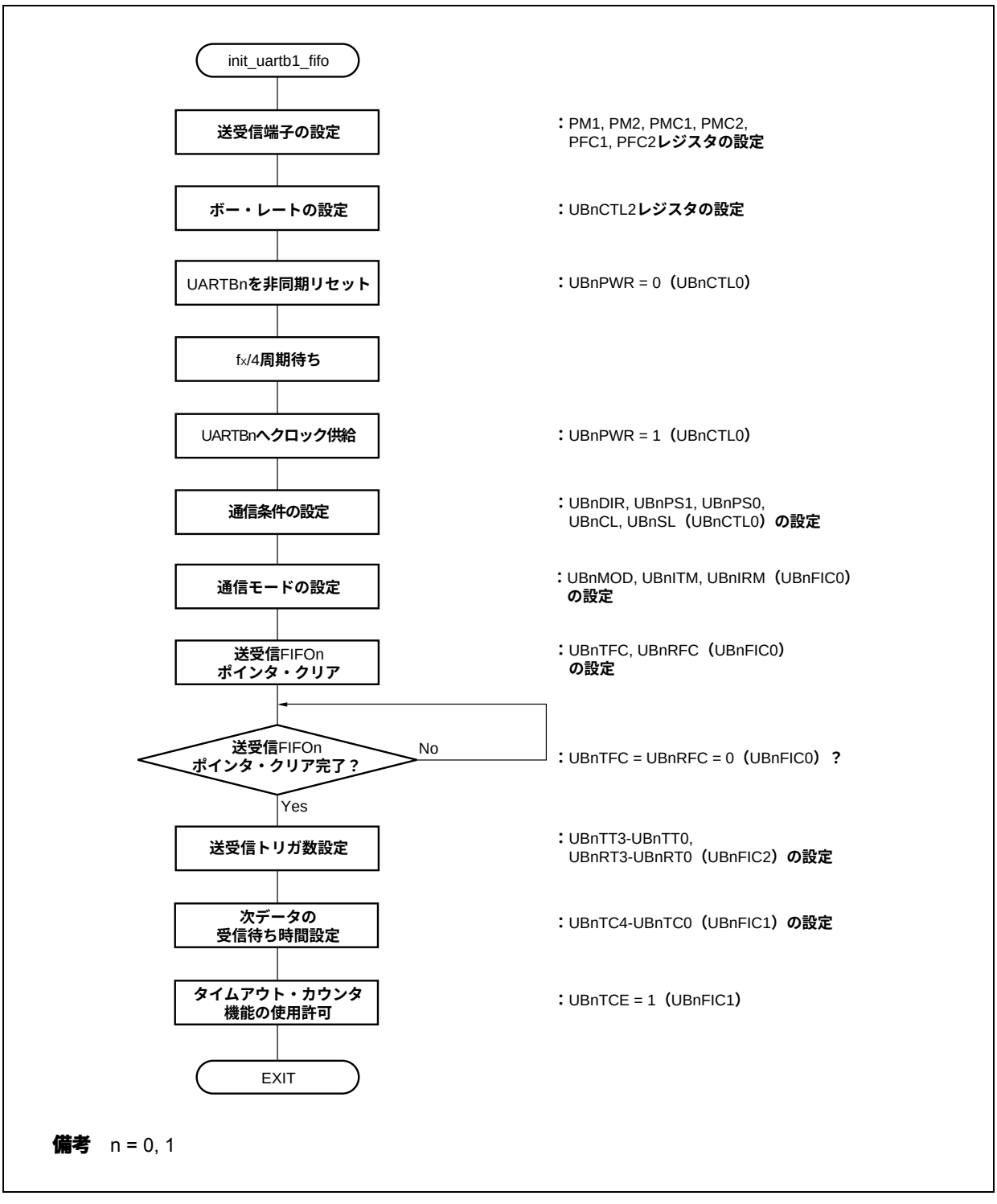
4. 2. 11　内蔵UARTBnのFIFOプログラム例

内蔵UARTBnのFIFOモードで送受信をするプログラム例について説明します（n = 0, 1）。このプログラム例では，固定文字列の送信を繰り返しながら，256バイトの受信リング・バッファを使って，外部からの連続受信を行います。

(1) フロー・チャート

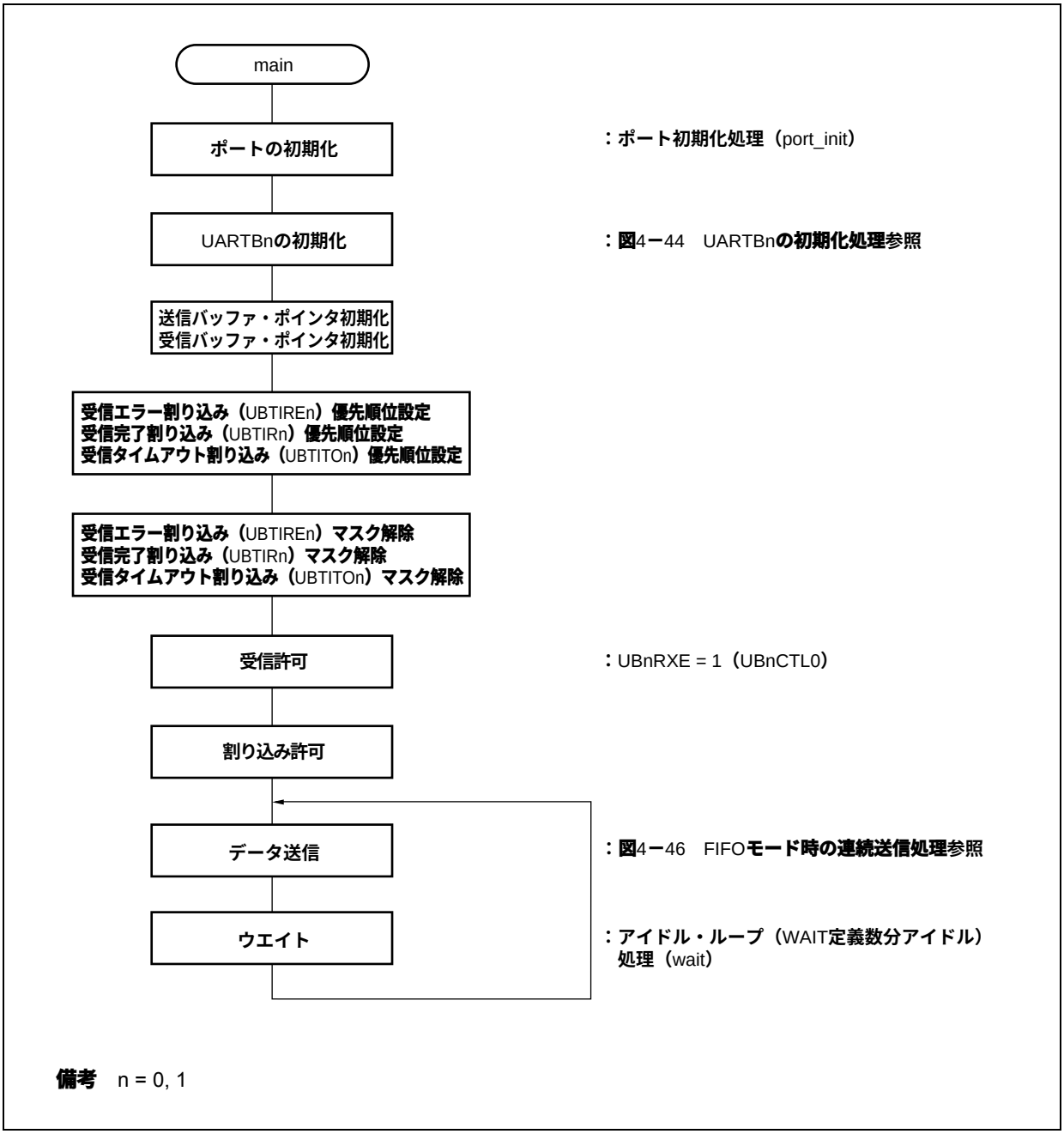
次にフロー・チャートを示します。

図4－44　UARTBnの初期化処理



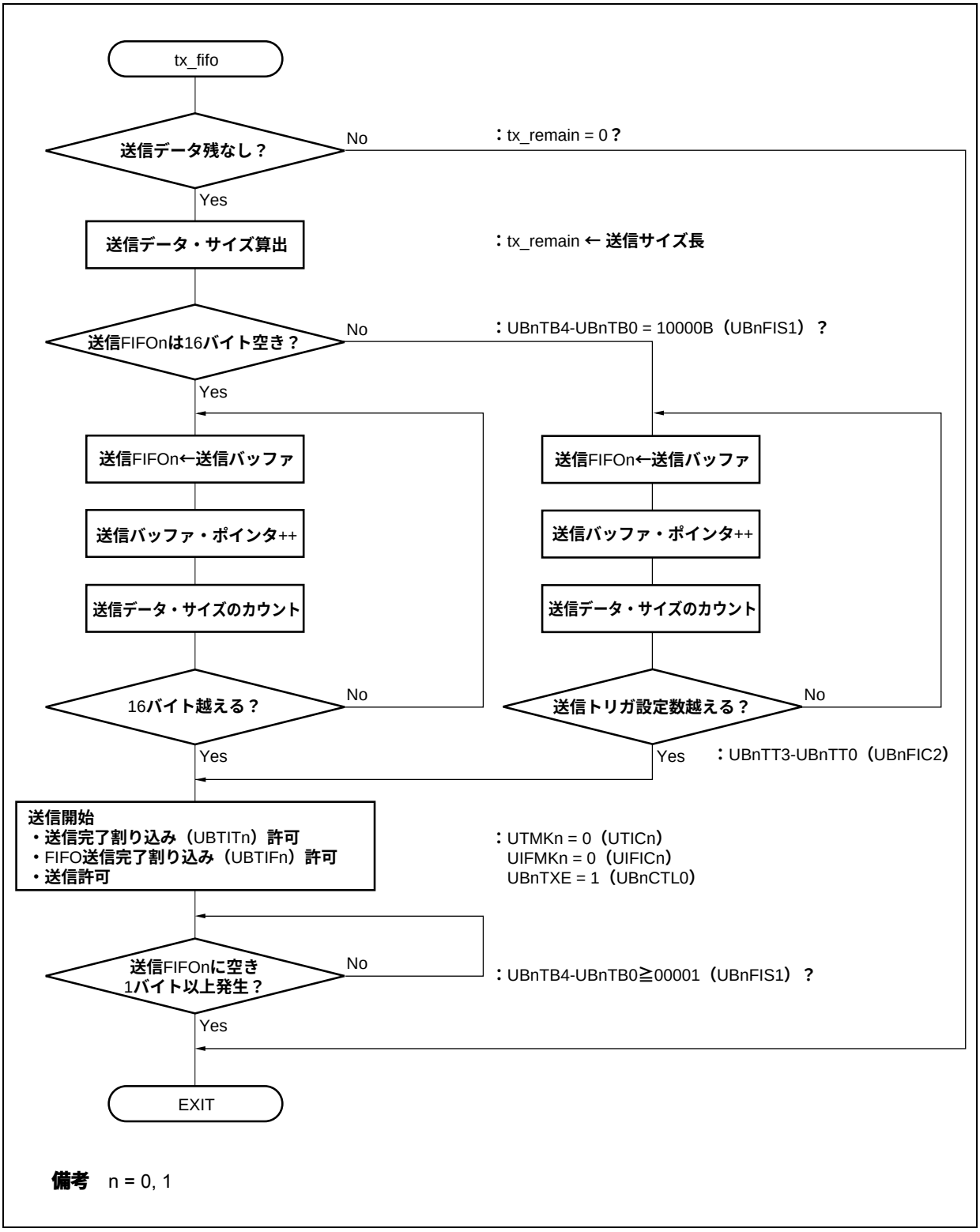
備考　n = 0, 1

図4－45　メイン処理



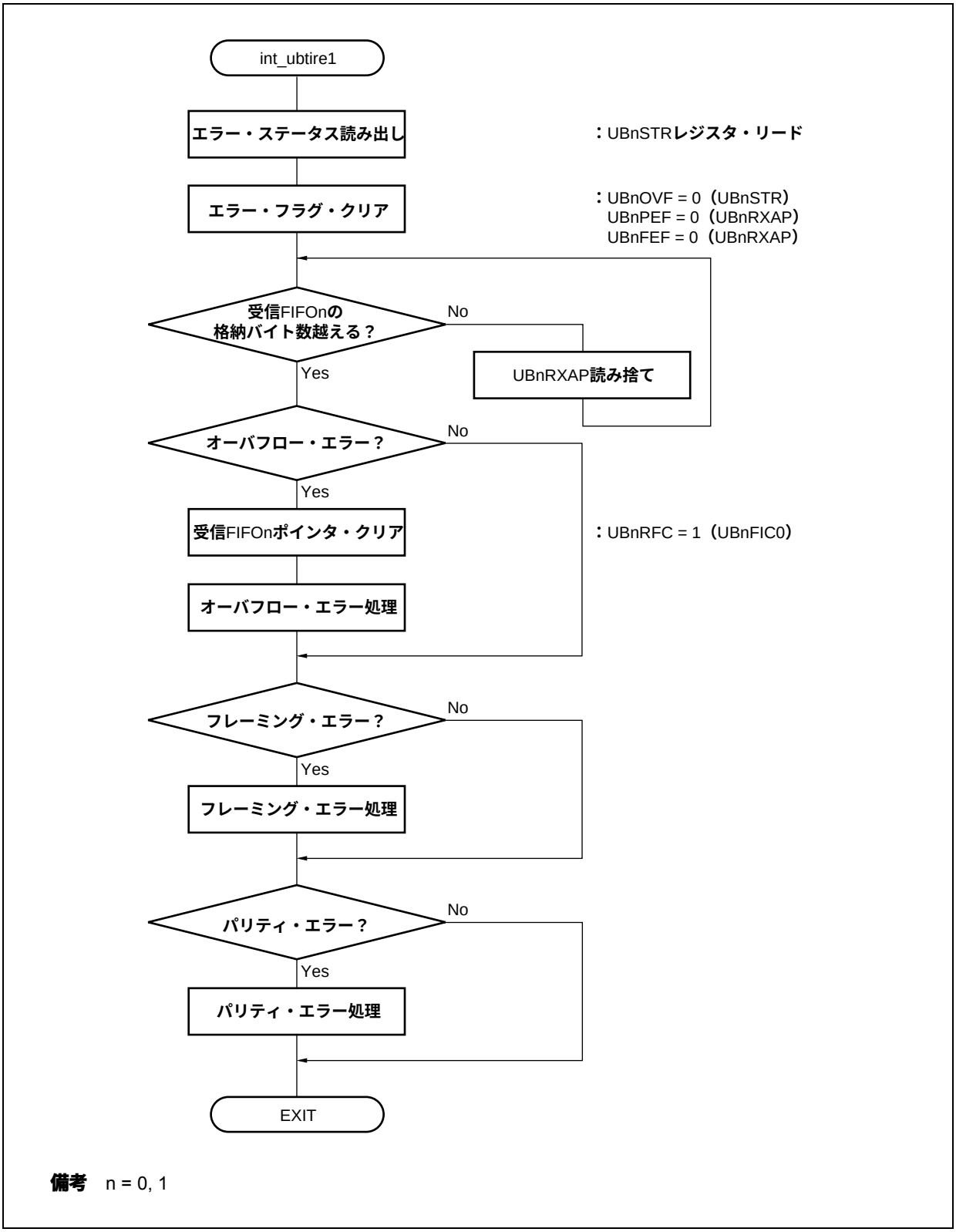
備考　n = 0, 1

図4－46　FIFOモード時の連続送信処理



備考　n = 0, 1

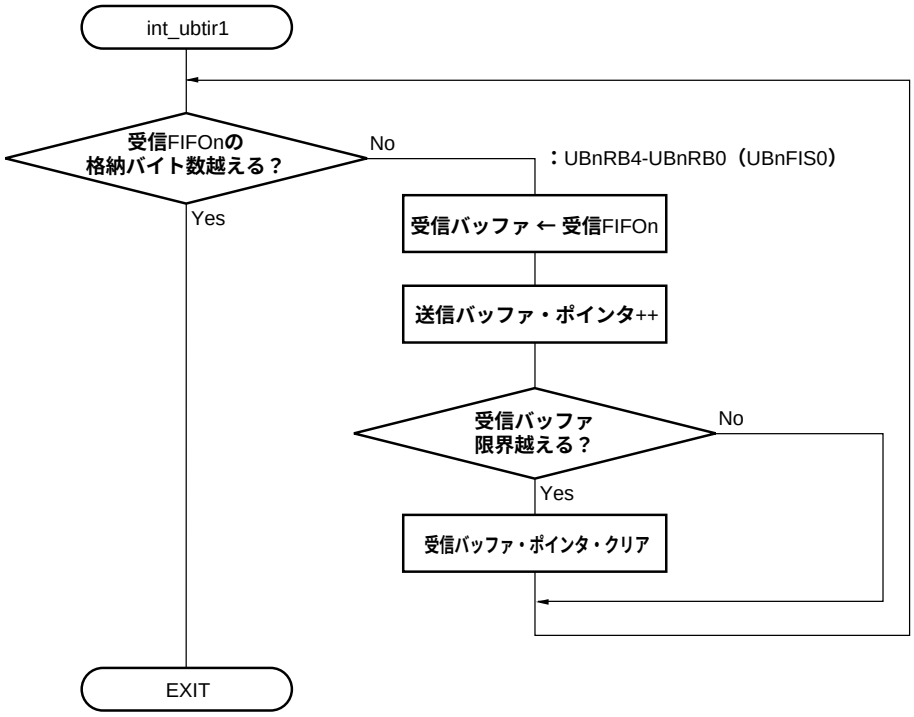
図4－47　受信エラー割り込み（UBTIREn）処理



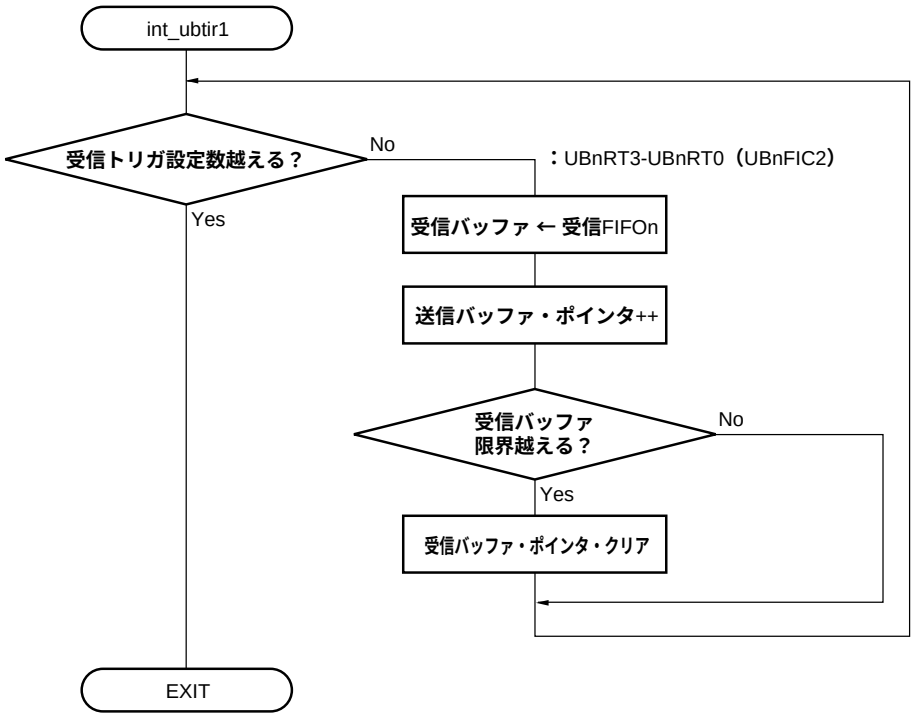
備考　n = 0, 1

図4－48　受信完了割り込み（UBTIRn）処理

（a）ポインタ・モード時



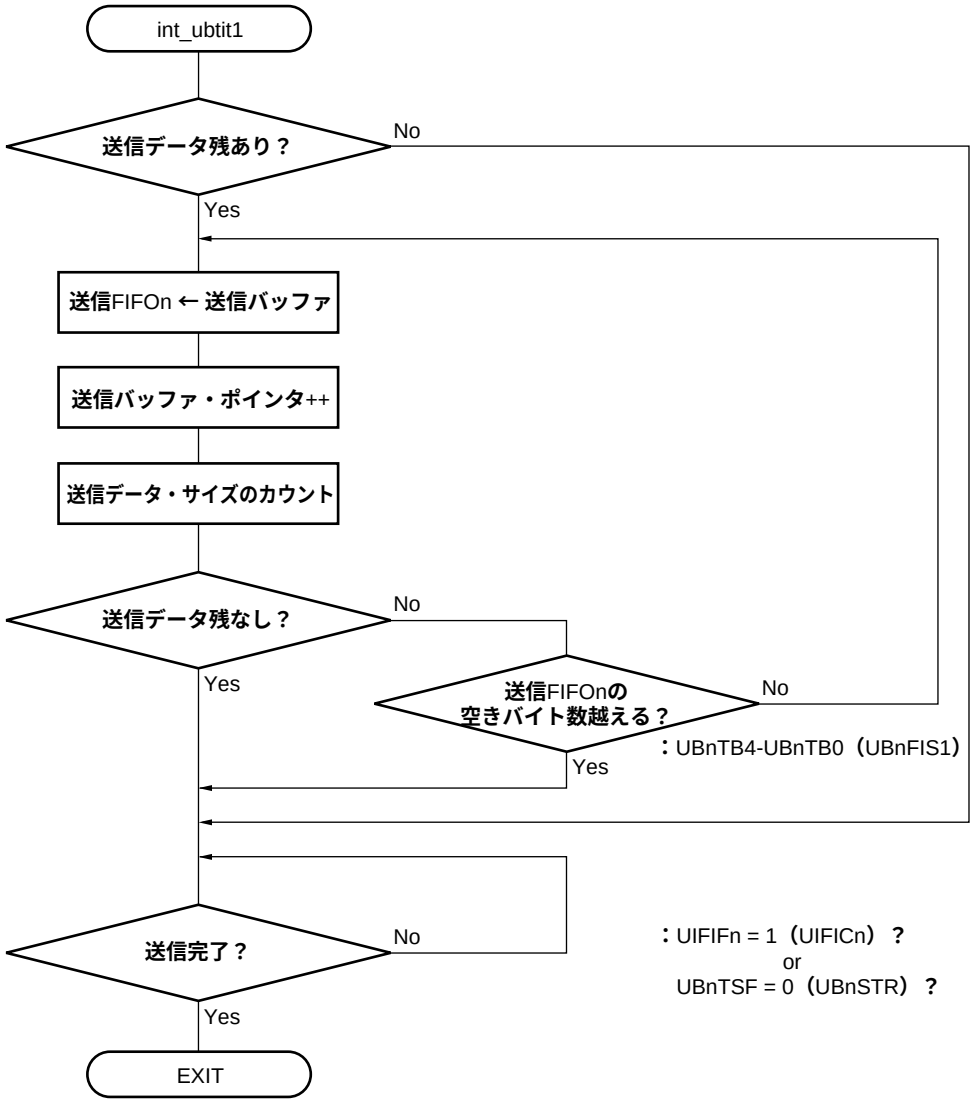
（b）保留モード時



備考　n = 0, 1

図4－49　送信完了割り込み（UBTITn）処理（1/2）

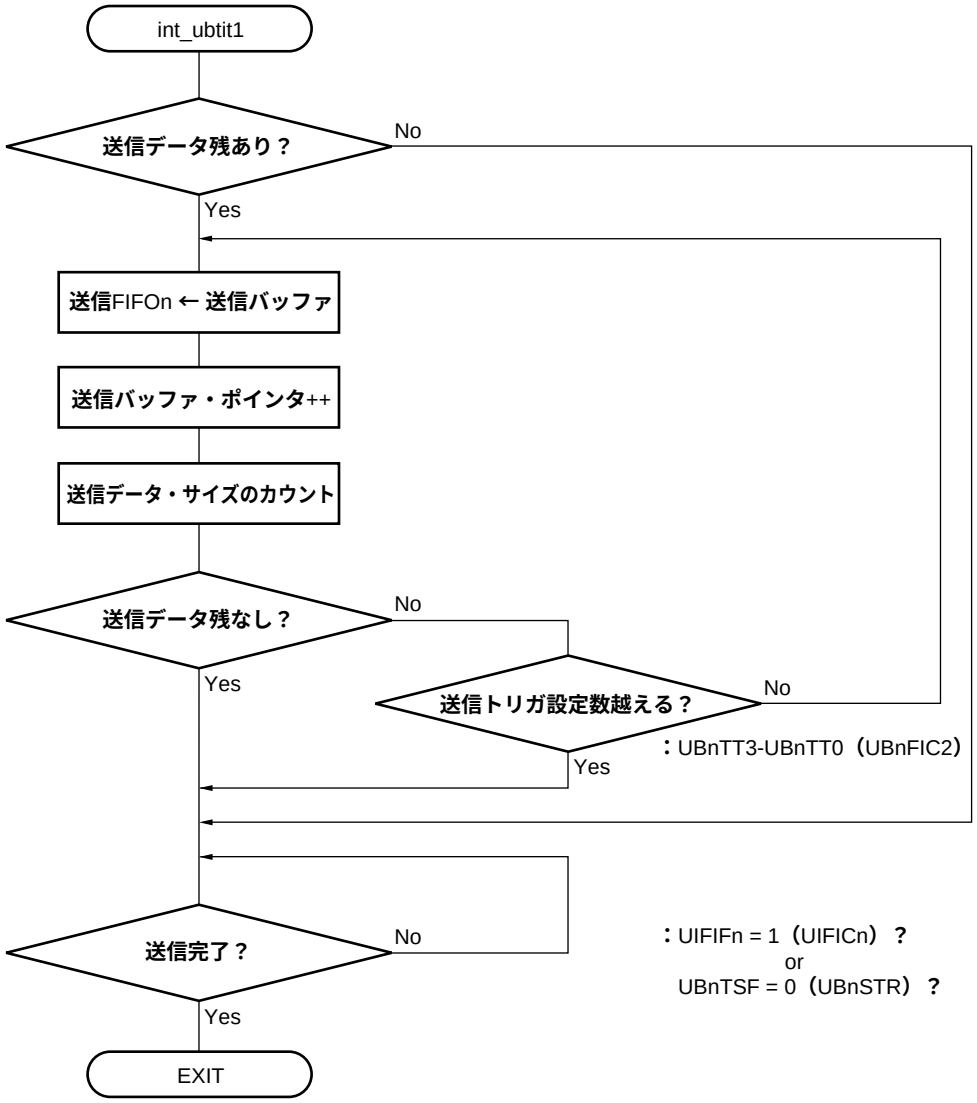
(a) ポインタ・モード時



備考　n = 0, 1

図4－49　送信完了割り込み（UBTITn）処理（2/2）

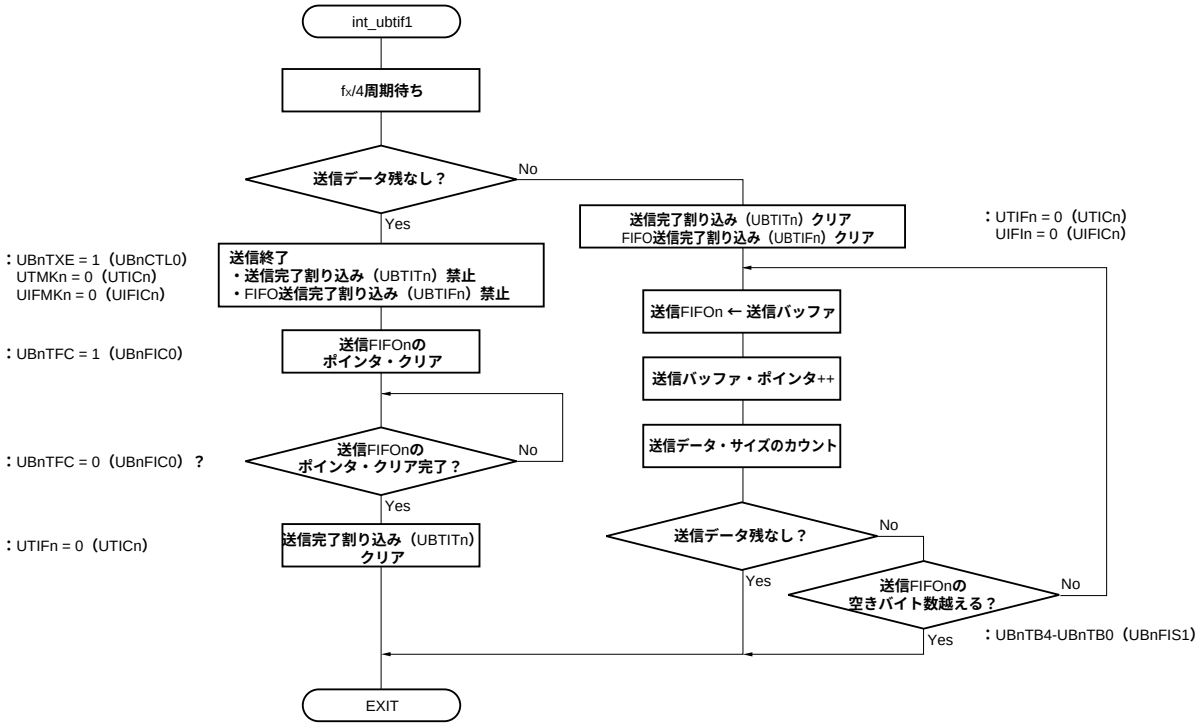
（b）保留モード時



備考　n = 0, 1

図4－50　FIFO送信完了割り込み（UBTIFn）処理（1/2）

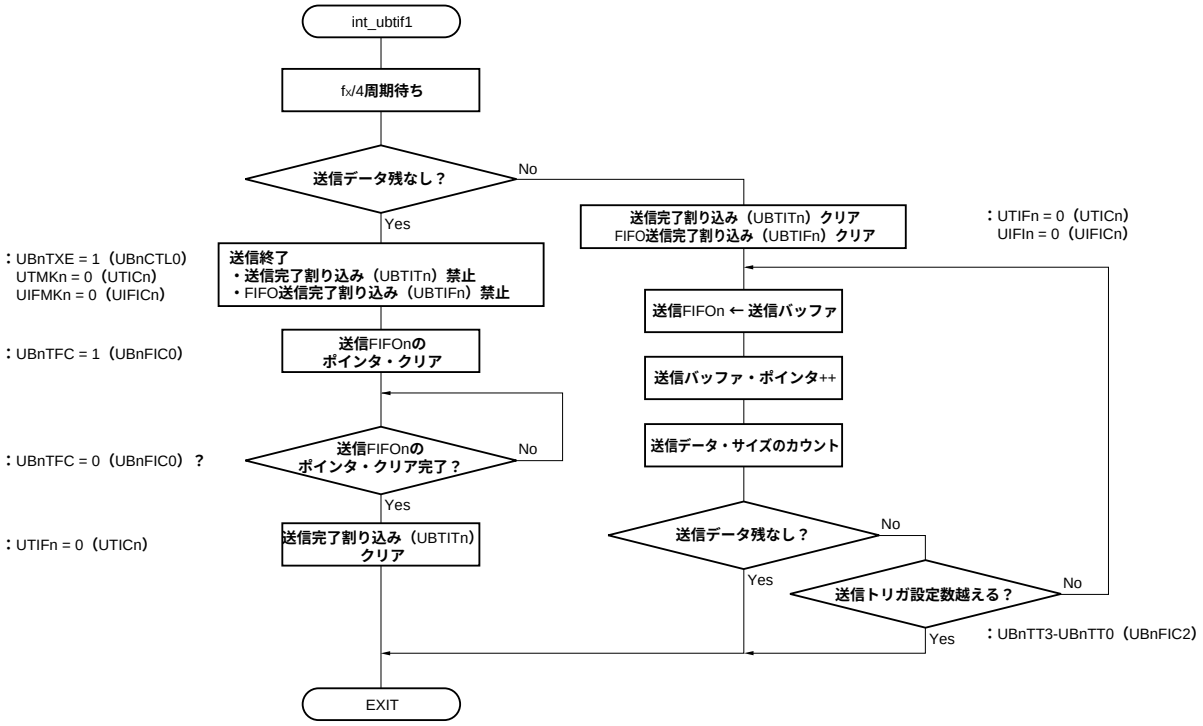
(a) ポインタ・モード時



備考　n = 0, 1

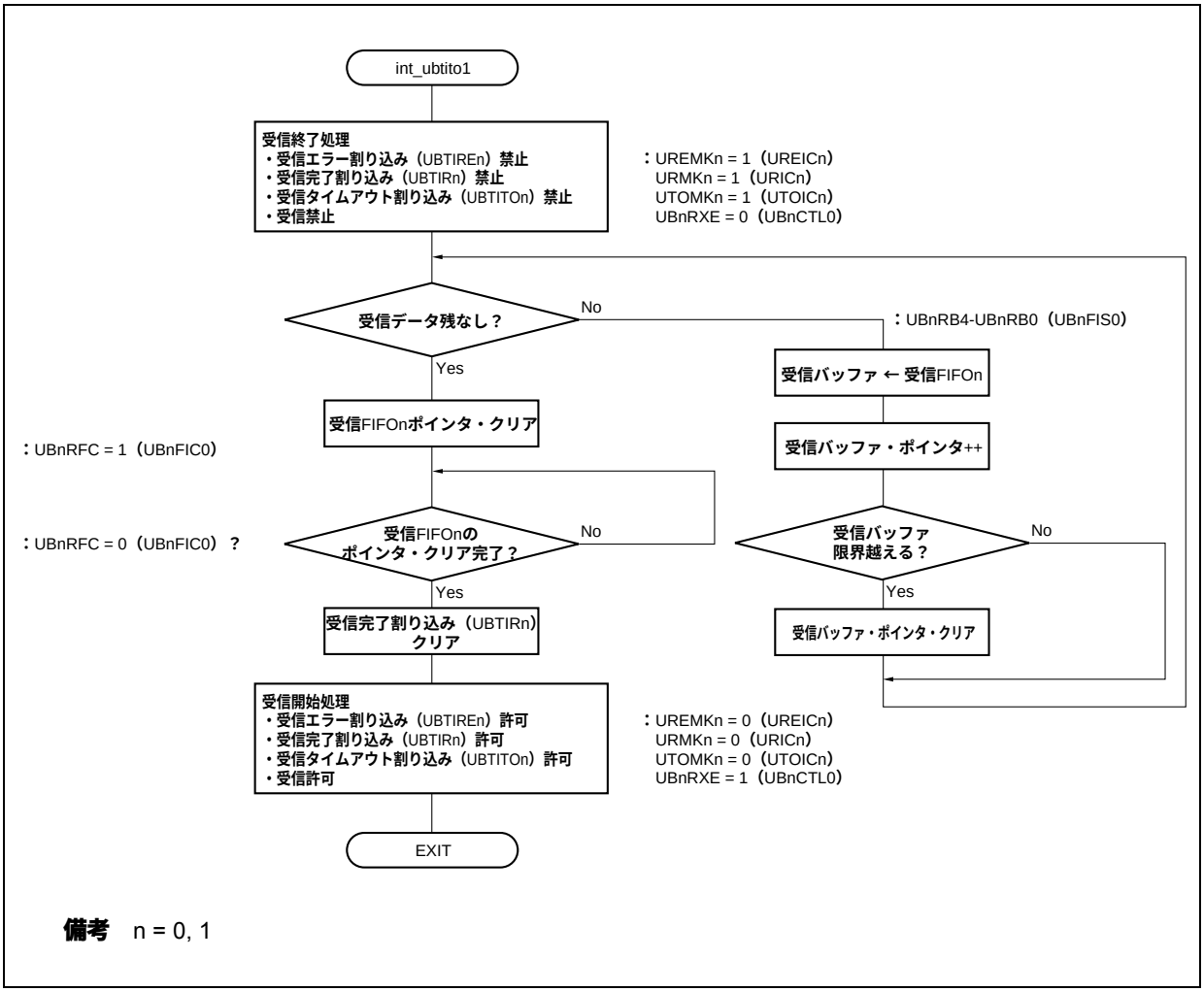
図4－50　FIFO送信完了割り込み（UBTIFn）処理（2/2）

（b）保留モード時



備考　n = 0, 1

図4－51　受信タイムアウト割り込み（UBTITOn）処理



備考　n = 0, 1

(2) プログラム・リスト

次にプログラム・リストのソース・ファイル一覧とプログラム・リストを示します。

表4－13　UARTBnのFIFO送受信プログラム・リストのソース・ファイル一覧

ファイル名	概　　要
uartfifo.c	wait関数，port_inti関数，main関数，FIFO送受信割り込み関数
printf.c	UARTB1 FIFO入出力処理関数群
txdata.c	連続送信文字列定義
txfiro.c	連続送信開始関数
uart.h	UARTB関連ヘッダ・ファイル

```
[uartrifo.c]
/* V850E/ME2 FIFO送受信プログラム */
/*                                */
/* 1.  文字列を繰り返し送信       */
/* 2.  文字列を受信バッファに格納。 */
/* 3.  受信エラーは Dot LEDに表示。 */
/*                                */

#include "uart.h"

#define WAIT 2200000                    /* アイドル時間 */

/* グローバル変数外部参照宣言 */
extern int tx_remain;                   /* 送信データ残りサイズ */
extern char txdata[];                   /* 送信文字列への配列外部宣言 */
extern char *txp;                       /* 送信文字列操作ポインタ外部宣言 */

/************************************************************************
 *
 *  関数名: port_init
 *  機　能: ポートの初期化
 *  引　数: なし
 *  戻り値: なし
 *
 ************************************************************************/
void port_init (void)
{

   P2    = 0x03;                        /* LED1-2(P25-P24)を消灯 */
                                        /* P23(USB制御),TxD1端子にインアクティブ・レベル・ロウ */
   PM2  |= 0xc7;                        /* P25-P24=Dot LED,P23(USB制御)を出力モード */
   PMC2 |= 0x07;                        /* P25-P23を入出力ポート */
   PFC2 |= 0x06;                        /* P22=TxD1,P21=RxD1,P20=NMI */

   P5    = 0x00;                        /* LED3-8(P55-P50)を消灯 */
   PM5  |= 0xc0;                        /* P55-P50=Dot LEDを出力モード */
   PMC5 |= 0x00;                        /* P55-P50を入出力ポート */

   PM6  |= 0xff;                        /* P67-P66=TOGLE SW,P65(USB給電監視)を入力モード */
   PMC6 |= 0x20;                        /* P67-P66を入出力ポート,P65=INTP65 */

   PM7  |= 0xff;                        /* P77-P72=TOGLE SWを入力モード */
   PMC7 |= 0x00;                        /* P77-P72を入出力ポート */

}
```

```
/************************************************************************
 *
 *  関数名: wait
 *  機　能: アイドル・ループ(WAIT定義数分アイドル)
 *  引　数: なし
 *  戻り値: なし
 *
 ************************************************************************/
void wait(void)
{
   int a=0;
   while(a!=WAIT)
   {
      a=a+1;
   }
}

/* グローバル変数定義 */
char RxBuf1[UART_RX_BUF_SIZE];          /* 受信バッファ */
int RxIdx1_wt;                          /* 受信バッファ書き込みインデクス */
int RxIdx1_rd;                          /* 送信バッファ読み出しインデクス */

/************************************************************************
 *
 *  関数名: main
 *  機　能: メイン処理
 *  引　数: なし
 *  戻り値: なし
 *
 ************************************************************************/
void main(void)
{

   int cnt;

   port_init();                         /* ポート初期化 */
   init_uartb1_fifo(115200);            /* UARTB1 チャネル初期化(FIFO Mode) */

   RxIdx1_rd = RxIdx1_wt = 0;           /* 受信バッファ書き込み／読み出しポインタ先頭へ */


   __set_il(PRIORITY(0), "UBTIRE1");    /* UARTB1 受信エラー割り込み       優先順位0 */
   __set_il(PRIORITY(1), "UBTIR1");     /* UARTB1 受信完了割り込み         優先順位1 */
   __set_il(PRIORITY(2), "UBTIT1");     /* UARTB1 送信完了割り込み         優先順位2 */
   __set_il(PRIORITY(3), "UBTIF1");     /* UARTB1 FIFO送信完了割り込み     優先順位3 */
   __set_il(PRIORITY(4), "UBTITO1");    /* UARTB1 受信タイムアウト割り込み 優先順位4 */
   __set_il(MASK_OFF, "UBTIRE1");       /* UARTB1 受信エラー割り込み許可 */
   __set_il(MASK_OFF, "UBTIR1");        /* UARTB1 受信完了割り込み許可 */
   __set_il(MASK_OFF, "UBTIT1");        /* UARTB1 送信完了割り込み許可 */
   __set_il(MASK_OFF, "UBTIF1");        /* UARTB1 FIFO送信完了割り込み許可 */
```

```
   __set_il(MASK_OFF, "UBTITO1");       /* UARTB1 受信タイムアウト割り込み許可 */


   /* 受信バッファをNULLで埋める初期化 */
   for(cnt = 0 ; cnt < 256 ; cnt++)RxBuf1[cnt] = 0x00;

   __EI();                              /* 全割り込み許可 */

   UART1_CONT_REG |=  UART_RXE_ON;      /* 受信許可 */

   while(1)
   {
      txp = txdata;
      tx_fifo();                        /* 送信開始 */
      wait();                           /* ウエイト */
   }
}


/************************************************************************
 *
 *  割り込み要求名: UBTIRE1
 *  機　　　能: UARTB1 受信エラー
 *
 ************************************************************************/
#pragma interrupt UBTIRE1 int_ubtire1
__interrupt
void int_ubtire1(void)
{
   int i;
   unsigned char  err_status;
   unsigned char rx_fifo_remain;
   unsigned short rx_trigger_cnt;
   unsigned short err_data;

   rx_fifo_remain = UB1FIS0;                    /* 受信FIFO状態読み出し */
   rx_trigger_cnt = (UB1FIC2 & 0x000f) + 1 ;    /* 受信FIFOトリガ数読み出し */

   /* 受信エラー状態と受信データ取得 */
   err_status = UART1_STAT_REG;

#if 0 /* ポインタ・モード */
   for(i = 0; i < rx_fifo_remain; i++){         /* 受信FIFOの空きバイト数分データ読み出し */
      err_data =  UB1RXAP;                      /* FIFO格納数分読み捨て */
   }
#else /* 保留モード */
   for(i = 0; i < rx_trigger_cnt; i++){         /* 受信FIFOのトリガ設定数分データ読み出し */
      err_data =  UB1RXAP;                      /* FIFO設定定数分読み捨て */
   }
```

```
#endif
   UART1_FIFO_CONT_REG |= UART_RX_FIFO_CLEAR;   /* FIFOクリアでエラー状態をリセット */

   /* 受信エラーの種類を判別 */
   if ((err_status &  UART_ERR_OVERFLOW) ||
      (err_data & (UART_ERR_PARITY_FIFO | UART_ERR_FRAME_FIFO))){
        if (err_status & UART_ERR_OVERFLOW){
           P2 = 0x10;                              /* オーバフロー・エラーをDot LEDへ表示 */
        }else if(err_data & UART_ERR_FRAME_FIFO){
           P2 = 0x20;                              /* フレーミング・エラーをDot LEDへ表示 */
        }else{
           P2 = 0x30;                              /* パリティ・エラーをDot LEDへ表示 */
        }
        err_status = UART1_STAT_REG &= 0x80;    /* 受信エラー状態フラグ・クリア */
   }
}
/*****************************************************************************
 *
 *  割り込み要求名: UBTIR1
 *  機          能: UARTB1 受信完了割り込み
 *
 *****************************************************************************/
#pragma interrupt UBTIR1 int_ubtir1
_ _interrupt
void int_ubtir1(void)
{
   unsigned char ch;
   unsigned char rx_fifo_remain;
   unsigned short rx_trigger_cnt;
   int i;
   rx_fifo_remain = UB1FIS0;                    /* 受信FIFO状態読み出し */
   rx_trigger_cnt = (UB1FIC2 & 0x000f) + 1 ;    /* 受信FIFOトリガ数読み出し */

#if 0 /* ポインタ・モード */
   for(i = 0; i < rx_fifo_remain; i++){         /* 受信FIFOの空きバイト数分データ読み出し */
      /* 受信FIFOから受信リング・バッファへ格納 */
      /* 受信リング・バッファ・ポインタ++ */
      RxBuf1[RxIdx1_wt++] =  UART1_RX_DATA_REG;
      /* 受信リング・バッファ・サイズ越ならば受信リング・バッファ・ポインタ初期化 */
      if(RxIdx1_wt > UART_RX_BUF_SIZE-1) RxIdx1_wt=0;
   }
```

```
#else /* 保留モード */
   for(i = 0; i < rx_trigger_cnt; i++){          /* 受信FIFOのトリガ設定数分データ読み出し */
      /* 受信FIFOから受信リング・バッファへ格納 */
      /* 受信リング・バッファ・ポインタ++ */
      RxBuf1[RxIdx1_wt++] =  UART1_RX_DATA_REG;
      /* 受信リング・バッファ・サイズ越ならば受信リング・バッファ・ポインタ初期化 */
      if(RxIdx1_wt > UART_RX_BUF_SIZE-1) RxIdx1_wt=0;
   }
#endif
}

/***********************************************************************
 *
 *  割り込み要求名: UBTIT1
 *  機　　　能: UARTB1 送信完了割り込み
 *
 ***********************************************************************/
#pragma interrupt UBTIT1 int_ubtit1
_ _interrupt
void int_ubtit1(void)
{
   int i;
   unsigned char fifo_empty_cnt;
   fifo_empty_cnt= ((UART1_FIFO_TRG_CONT_REG & 0x0f00) >> 8)+1;

   /* 送信データ残あり? */
   if(tx_remain > 0){
      do{
           /* 送信バッファから送信FIFOへ */
           /* 送信バッファ・ポインタ++ */
           UART1_TX_DATA_REG = *txp++;
           tx_remain--;                          /*送信データ・サイズ・カウント */
           if(tx_remain == 0)break;              /* 送信データ残りなし? */
      }while(fifo_empty_cnt--);                  /* 送信トリガ設定数越? */
   }
   /* 送信シフト・レジスタおよび送信FIFOに転送すべきデータが存在しない */
   while(UB1TSF != 0);
   /* ここに至れば完全に連続送信終了。FIFO送信完了割り込みで後処理 */
}
```

```
/************************************************************************
 *
 *  割り込み要求名: UBTIF1
 *  機　　能: UARTB1 FIFO送信完了割り込み
 *
 ************************************************************************/
#pragma interrupt UBTIF1 int_ubtif1
_ _interrupt
void int_ubtif1(void)
{
   unsigned char fifo_empty_cnt;
   fifo_empty_cnt= ((UART1_FIFO_TRG_CONT_REG & 0x0f00) >> 8)+1;

   _asm("nop"); _asm("nop"); _asm("nop"); _asm("nop");  /* fx/4 1周期待ち */
   _asm("nop"); _asm("nop"); _asm("nop"); _asm("nop");  /* fx/4 1周期待ち */

   if(tx_remain > 0){                    /* 送信データ残りなし? */
          UTIC1  &= ~IREQ_FLAG;          /* UTIF1 =0:送信完了割り込み要求クリア */
          UIFIC1 &= ~IREQ_FLAG;          /* UIFIF1=0 :FIFO送信完了割り込み要求クリア */
          do{
                /* 送信バッファから送信FIFOへ */
                /* 送信バッファ・ポインタ++ */
                UART1_TX_DATA_REG = *txp++;
                tx_remain--;                    /* 送信データ・サイズ・カウント */
                if(tx_remain == 0)break;        /* 送信データ残りなし? */
          }while(fifo_empty_cnt--);             /* 送信トリガ設定数越? */
   }else{
          /* 送信シフト・レジスタにも送信FIFOにも転送すべきデータが */
          /* 存在しない(連続送信終了した)ので送信処理初期化 */
          UART1_TX = DISABLE;                   /* UB1TXE=0:送信禁止 */
          _ _set_il(MASK_ON, "UBTIT1");         /* UARTB1 送信完了割り込みマスク */
          _ _set_il(MASK_ON, "UBTIF1");         /* UARTB1 FIFO送信完了割り込みマスク */
#if 1
          UART1_FIFO_POINTER = CLEAR;           /* UB1TFC=1:送信FIFOのポインタ・クリア(0) */
          while(UART1_FIFO_POINTER != NORMAL);  /* UB1TFC=0の確認 */
#endif
          UTIC1  &= ~IREQ_FLAG;                 /* UTIF1=0:送信完了割り込み要求クリア */
          UIFIC1 &= ~IREQ_FLAG;                 /* UIFIF1=0:FIFO送信完了割り込み要求クリア */
   }
}
```

```
/************************************************************************
 *
 *  割り込み要求名：UBTITO1
 *  機　　　能：UARTB1 受信タイムアウト割り込み
 *
 ************************************************************************/
#pragma interrupt UBTITO1 int_ubtito1
_ _interrupt
void int_ubtito1(void)
{
   unsigned char rx_fifo_remain;
   unsigned short rx_trigger_cnt;
   int i;

   rx_trigger_cnt = (UB1FIC2 & 0x000f) +1 ;      /* 受信FIFOトリガ数読み出し */
   rx_fifo_remain = UB1FIS0;                     /* 受信FIFO状態読み出し */

   /* 受信終了処理 */
   _ _set_il(MASK_ON, "UBTIR1");                 /* UARTB1 受信完了割り込みマスク */
   _ _set_il(MASK_ON, "UBTIRE1");                /* UARTB1 受信エラー割り込みマスク */
   _ _set_il(MASK_ON, "UBTITO1");                /* UARTB1 受信タイムアウト割り込みマスク */
   UART1_CONT_REG &=  ~UART_RXE_OFF;             /* 受信禁止 */

   /* トリガ設定数2byte以上の保留モード時にトリガ設定数未満のデータを取り出す */
   for(i = 0; i < rx_fifo_remain; i++){
      /* 受信FIFOから受信リング・バッファへ格納 */
      /* 受信リング・バッファ・ポインタ++ */
      RxBuf1[RxIdx1_wt++] =  UART1_RX_DATA_REG;
      /* 受信リング・バッファ・サイズ越ならば受信リング・バッファ・ポインタ初期化 */
      if(RxIdx1_wt > UART_RX_BUF_SIZE-1) RxIdx1_wt=0;
   }

   UB1FIC0.2 = CLEAR;                    /* UB1RFC=1:受信FIFOのポインタ・クリア(0) */
   while(UB1FIC0.2 != NORMAL);           /* UB1RFC=0の確認 */
   URIC1 &= ~IREQ_FLAG;                  /* URIF1=0:受信完了割り込み要求クリア */
   /* 受信開始処理 */
   _ _set_il(MASK_OFF, "UBTIR1");        /* UARTB1受信完了割り込みマスク解除 */
   _ _set_il(MASK_OFF, "UBTIRE1");       /* UARTB1受信エラー割り込みマスク解除 */
   _ _set_il(MASK_OFF, "UBTITO1");       /* UARTB1受信タイムアウト割り込みマスク解除 */
   UART1_CONT_REG |= UART_RXE_OFF;       /* 受信許可 */
}
```

```
 [printf.c]
#include <stdio.h>
#include <stdarg.h>

#include "uart.h"
#define TRUE  1
#define FALSE 0

/*****************************************************************************
 *
 *  関数名: init_uartb1_fifo
 *  機　能: UARTB1のFIFOモード初期化
 *  引　数: bps: 転送速度(uart.h参照)
 *  戻り値: なし
 *
 *****************************************************************************/
void init_uartb1_fifo(int bps)
{
   int i;

   /* UARTB1制御関連のPort2初期化 */
   P2     = 0x03;
   PM2   |= 0xc7;
   PMC2  |= 0x07;
   PFC2  |= 0x06;

   /* 転送速度よりボー・レート・ファクタ設定 */
   switch(bps){
      case UART_115200BPS  :UB1CTL2 = UART_BAUD_115200;break;
      case UART_57600BPS   :UB1CTL2 = UART_BAUD_57600;break;
      case UART_38400BPS   :UB1CTL2 = UART_BAUD_38400;break;
      case UART_19200BPS   :UB1CTL2 = UART_BAUD_19200;break;
      case UART_9600BPS    :UB1CTL2 = UART_BAUD_9600;break;
      case UART_4800BPS    :UB1CTL2 = UART_BAUD_4800;break;
      case UART_2400BPS    :UB1CTL2 = UART_BAUD_2400;break;
      default              :UB1CTL2 = UART_BAUD_9600;break; /* ディフォールト 9600bps */
   }


   UART1_CONT_REG = UART_PWR_OFF;        /* UARTを非同期にリセット */
   for(i = 0 ; i < 10 ; i++);            /* fx/4の2周期分以上待ち */
   UART1_CONT_REG |= UART_PWR_ON;        /* UARTをへクロック供給 */
   /* LSB先頭,パリティなし,8bit長,ストップ・ビット 1 */
   UART1_CONT_REG |= (UART_DIR_LSB |UART_CL_8 | UART_SL_1);
      /* FIFOを使用 */
```

```
#if 0   /* ポインタ・モード */
   UART1_FIFO_CONT_REG =
      (UART_FIFO_MODE            |
       UART_TX_FIFO_CLEAR        |
       UART_RX_FIFO_CLEAR        |
       UART_TX_FIFO_INT_POINTER  |
       UART_RX_FIFO_INT_POINTER
      );
#else
   UART1_FIFO_CONT_REG =
      (UART_FIFO_MODE            |
       UART_TX_FIFO_CLEAR        |
       UART_RX_FIFO_CLEAR        |
       UART_TX_FIFO_INT_PENDING  |
       UART_RX_FIFO_INT_PENDING
      );
#endif
   while(UART1_FIFO_POINTER != NORMAL);          /* UB1TFC=0の確認 */
   while(UB1FIC0.2 != NORMAL);                   /* UB1RFC=0の確認 */

   /* FIFOのポインタ・モード時は送受信トリガ数1バイト固定 */
   UART1_FIFO_TRG_CONT_REG |=(UART_TX_FIFO_TRIGGER(12)|UART_RX_FIFO_TRIGGER(12));
   UART1_RX_FIFO_TMOUT_CONT_REG = ~(12)&0x1f;
   UART1_RX_FIFO_TMOUT_CONT_REG |= (UART_FIFO_TIMEOUT_OFF);
}

/*****************************************************************************
 *
 *  関数名: strlen
 *  機　能: 文字を数える
 *  引　数: str 文字列へのポインタ(NULLで終っている必要がある)
 *  戻り値: 文字数 NULLを含まない
 *
 *****************************************************************************/
int strlen(const char *str)
{
   int len = 0;

   for(; str[len] != (const char )NULL; len++)
         ;

   return len;
}
```

```
/*****************************************************************************
 *
 *  関数名: printf
 *  機　能: UARTB1出力printf処理
 *  引き数: 通常のprintfと同じ
 *  戻り値: 通常のprintfと同じ
 *
 *****************************************************************************/
int printf(const char *fmt, ...)
{
  char buf[256];
  va_list ap;
  int ret = 0;

  /* 引数リスト走査用変数の初期化 */
  va_start(ap, fmt);

  /* 書式付データをバッファへ出力 */
  ret = vsprintf(buf, fmt, ap);

  /* 引数リスト走査の終了 */
  va_end(ap);

  /* UARTB1でバッファの中身を出力 */
  tx_fifo(buf);
  return ret;
}
```

[txdata.c]

```
/* 連続送信する文字列 */
char txdata[] = {
"¥
* UARTB1 FIFO Tx demo 00time *¥r¥n¥
* UARTB1 FIFO Tx demo 01time *¥r¥n¥
* UARTB1 FIFO Tx demo 02time *¥r¥n¥
* UARTB1 FIFO Tx demo 03time *¥r¥n¥
* UARTB1 FIFO Tx demo 04time *¥r¥n¥
* UARTB1 FIFO Tx demo 05time *¥r¥n¥
* UARTB1 FIFO Tx demo 06time *¥r¥n¥
* UARTB1 FIFO Tx demo 07time *¥r¥n¥
* UARTB1 FIFO Tx demo 08time *¥r¥n¥
* UARTB1 FIFO Tx demo 09time *¥r¥n¥
* UARTB1 FIFO Tx demo 0atime *¥r¥n¥
* UARTB1 FIFO Tx demo 0btime *¥r¥n¥
* UARTB1 FIFO Tx demo 0ctime *¥r¥n¥
* UARTB1 FIFO Tx demo 0dtime *¥r¥n¥
* UARTB1 FIFO Tx demo 0etime *¥r¥n¥
* UARTB1 FIFO Tx demo 0ftime *¥r¥n¥
* UARTB1 FIFO Tx demo 10time *¥r¥n¥
* UARTB1 FIFO Tx demo 11time *¥r¥n¥
* UARTB1 FIFO Tx demo 12time *¥r¥n¥
* UARTB1 FIFO Tx demo 13time *¥r¥n¥
* UARTB1 FIFO Tx demo 14time *¥r¥n¥
* UARTB1 FIFO Tx demo 15time *¥r¥n¥
* UARTB1 FIFO Tx demo 16time *¥r¥n¥
* UARTB1 FIFO Tx demo 17time *¥r¥n¥
* UARTB1 FIFO Tx demo 18time *¥r¥n¥
* UARTB1 FIFO Tx demo 19time *¥r¥n¥
* UARTB1 FIFO Tx demo 1atime *¥r¥n¥
* UARTB1 FIFO Tx demo 1btime *¥r¥n¥
* UARTB1 FIFO Tx demo 1ctime *¥r¥n¥
* UARTB1 FIFO Tx demo 1dtime *¥r¥n¥
* UARTB1 FIFO Tx demo 1etime *¥r¥n¥
* UARTB1 FIFO Tx demo 1ftime *¥r¥n¥
"¥
};
```

```
 [txfifo.c]
#include "uart.h"

/* グローバル変数定義 */
char *txp;                              /* 連続送信用の送信文字列へのポインタ */
int tx_remain;                          /* 送信データの残りサイズ */

/*****************************************************************************
 *
 *  関数名: tx_fifo
 *  機　能: FIFO連続送信の開始
 *  引　数: なし
 *  戻り値: なし
 *
 *****************************************************************************/
void tx_fifo(void)
{
   int i;
   unsigned char tx_trigger_cnt;
   unsigned char fifo_empty_cnt;

   if(tx_remain > 0)return;             /* 送信データ残なし? */

   /* 送信シフト・レジスタおよび送信FIFOに転送すべきデータが存在しない */
   while(UB1TSF != 0);
   /* ここに至れば完全に連続送信終了。FIFO送信完了割り込みで後処理 */

   tx_remain = strlen(txp);             /* 送信データ・サイズ算出 */

   fifo_empty_cnt= UB1FIS1;
```

```
  /*保留モード時のトリガ設定数読み出し */
  tx_trigger_cnt = ((UB1FIC2 & 0x0f00) >> 8) + 1;
    if(UB1FIS1 == 16) {
    /* FIFOクリア後の最初の書き込みでFIFOが16byte空いているならば */
    /* FIFO最大16byte目まで書き込み */
         do{
               /* 送信バッファから送信FIFOへ */
               /* 送信バッファ・ポインタ++ */
               UART1_TX_DATA_REG = *txp++;
               tx_remain--;                    /*送信データ・サイズ・カウント */
               if(tx_remain == 0)break;        /* 送信データ残りなし? */
         }while(fifo_empty_cnt--);             /* 送信トリガ設定数越? */
    }else{
    /* FIFOへトリガ設定数分だけ書き込み */
         for(i = 0 ; i < tx_trigger_cnt; i++){
         /* 送信FIFOへ送信リング・バッファ文字列を書き込み */
         UART1_TX_DATA_REG = *txp++;
         tx_remain--;                          /* 転送残数カウント */
         }
    }
    /* 送信割り込みを許可してFIFO送信を開始する */
    _ _set_il(MASK_OFF, "UBTIT1");             /* UARTB1 送信完了割り込みマスク解除 */
    _ _set_il(MASK_OFF, "UBTIF1");             /* UARTB1 FIFO送信完了割り込みマスク解除 */
    UART1_CONT_REG/*UB1CTL0*/ |=  UART_TXE_ON; /* 送信許可 */

    /* 送信FIFOから少なくとも1byte分のデータUB1TXへ転送されることにより */
    while(!(UB1FIS1 >= 1));                    /* 送信FIFOの空き1byte以上発生? */
    /* ここまで来たならばこの後に送信完了割り込み(UBTIT1)が発生するはず */
}
```

[uart.h]

```
/*--------------------------------------------------------------------*/
/* 内蔵I/Oレジスタ名定義                                              */
/*--------------------------------------------------------------------*/
#pragma ioreg

#define BRG1_FACTOR                     UB1CTL2         /* BRG factor制御 */
#define UART1_CONT_REG                  UB1CTL0         /* 全体の制御 */
#define UART1_STAT_REG                  UB1STR          /* 状態制御 */
#define UART1_RX_ERR_INT_CONT_REG       UREIC1          /* 受信エラー割り込み制御 */
#define UART1_RX_INT_CONT_REG           URIC1           /* 受信完了割り込み制御 */
#define UART1_TX_INT_CONT_REG           UTIC1           /* 送信完了割り込み制御 */
#define UART1_TX_FIFO_INT_CONT_REG      UIFIC1          /* FIFO送信完了割り込み制御 */
#define UART1_RX_FIFO_TMOUT_INT_CONT_REG UTOIC1         /* FIFO受信タイムアウト割り込み制御 */
#define UART1_TX_DATA_REG               UB1TX           /* 送信データ */
#define UART1_RX_DATA_REG               UB1RX           /* 送信データ */
#define UART1_FIFO_CONT_REG             UB1FIC0         /* FIFO制御 */
#define UART1_RX_FIFO_TMOUT_CONT_REG    UB1FIC1         /* 受信FIFOタイムアウト制御 */
#define UART1_FIFO_TRG_CONT_REG         UB1FIC2         /* 送受信FIFOトリガ制御 */
#define UART1_RX_FIFO_STAT_REG          UB1FIS0         /* 受信FIFO状態 */
#define UART1_TX_FIFO_STAT_REG          UB1FIS1         /* 送信FIFO状態 */

#define UART1_TX                        UART1_CONT_REG.6        /* 送信許可／禁止*/
#define UART1_FIFO_POINTER              UART1_FIFO_CONT_REG.3   /* 送信FIFO ポインタ・クリア
*/

/*--------------------------------------------------------------------*/
/* 転送速度定義                                                       */
/*--------------------------------------------------------------------*/
#define UART_115200BPS    115200
#define UART_57600BPS     57600
#define UART_38400BPS     38400
#define UART_19200BPS     19200
#define UART_9600BPS      9600
#define UART_4800BPS      4800
#define UART_2400BPS      2400

#if 0
/* UB0CTL1,UB0CTL2 設定値 fx=133MHz,Clock=fx/4基準 */
#define UART_BAUD_2400    6927                  /* 誤差 +0.001% */
#define UART_BAUD_4800    3464                  /* 誤差 -0.013% */
#define UART_BAUD_9600    1732                  /* 誤差 -0.013% */
#define UART_BAUD_19200   866                   /* 誤差 -0.013% */
#define UART_BAUD_38400   433                   /* 誤差 -0.013% */
#define UART_BAUD_57600   289                   /* 誤差 -0.128% */
#define UART_BAUD_115200  144                   /* 誤差 +0.218% */
#else
/* UB0CTL1,UB0CTL2 設定値 fx= 96MHz,Clock=fx/4基準 */
#define UART_BAUD_2400    5000                  /* 誤差 -0.000% */
#define UART_BAUD_4800    2500                  /* 誤差 -0.000% */
```

```
#define UART_BAUD_9600    1250                /* 誤差 -0.000% */
#define UART_BAUD_19200   625                 /* 誤差 -0.000% */
#define UART_BAUD_38400   313                 /* 誤差 -0.160% */
#define UART_BAUD_57600   208                 /* 誤差 +0.160% */
#define UART_BAUD_115200  104                 /* 誤差 +0.160% */
#endif

/*-------------------------------------------------------------------*/
/* UARTBn制御レジスタ0(UBnCTL0)ビット定義                             */
/*-------------------------------------------------------------------*/
#define UART_PWR_ON     (1 << 7)              /* UARTへのクロック供給停止 */
#define UART_PWR_OFF    (0 << 7)              /* UARTへのクロック供給開始 */

#define UART_TXE_ON     (1 << 6)              /* 送信許可 */
#define UART_TXE_OFF    (0 << 6)              /* 送信禁止 */

#define UART_RXE_ON     (1 << 5)              /* 受信許可 */
#define UART_RXE_OFF    (0 << 5)              /* 受信禁止 */

#define UART_DIR_LSB    (1 << 4)              /* 転送データ先頭ビットはLSB */
#define UART_DIR_MSB    (0 << 4)              /* 転送データ先頭ビットはMSB */

#define UART_PS_NO      (0 << 3) | (0 << 2)   /* パリティなし */
#define UART_PS_0       (0 << 2) | (1 << 2)   /* パリティ0 */
#define UART_PS_ODD     (1 << 3) | (0 << 2)   /* 奇数パリティ */
#define UART_PS_EVN     (1 << 3) | (1 << 2)   /* 偶数パリティ */

#define UART_CL_8       (1 << 1)              /* 転送データ１フレームは8ビット */
#define UART_CL_7       (0 << 1)              /* 転送データ１フレームは7ビット */

#define UART_SL_2       (1 << 0)              /* 転送データ ストップ・ビットは2ビット */
#define UART_SL_1       (0 << 0)              /* 転送データ ストップ・ビットは1ビット */

/*-------------------------------------------------------------------*/
/* UARTBn受信データ・レジスタAP(UBnRXAP)エラー・ビット定義            */
/*-------------------------------------------------------------------*/
#define UART_ERR_PARITY_FIFO    (1 << 9)      /* FIFOモード パリティ・エラー */
#define UART_ERR_FRAME_FIFO     (1 << 8)      /* FIFOモード フレーミング・エラー */

/*-------------------------------------------------------------------*/
/* UARTBn状態レジスタ(UBnSTR)ビット定義                               */
/*-------------------------------------------------------------------*/
#define UART_ERR_OVERFLOW       (1 << 3)      /* FIFOモード・オーバフロー・エラー */
#define UART_ERR_PARITY         (1 << 2)      /* シングル・モード・パリティ・エラー */
#define UART_ERR_FRAME          (1 << 1)      /* シングル・モード・フレーミング・エラー */
#define UART_ERR_OVERRUN        (1 << 0)      /* シングル・モード・オーバラン・エラー */
```

```
/*--------------------------------------------------------------------*/
/* UARTBn FIFO制御レジスタ0(UBnFIC0)ビット定義                          */
/*--------------------------------------------------------------------*/
#define UART_FIFO_MODE          (1 << 7)        /* FIFOモード */
#define UART_SINGLE_MODE        (0 << 7)        /* シングル・モード */

#define UART_TX_FIFO_CLEAR      (1 << 3)        /* 送信FIFOポインタ・クリア */
#define UART_TX_FIFO_NORMAL     (0 << 3)        /* 送信FIFOポインタ通常 */
#define UART_RX_FIFO_CLEAR      (1 << 2)        /* 受信FIFOポインタ・クリア */
#define UART_RX_FIFO_NORMAL     (0 << 2)        /* 受信FIFOポインタ通常 */

#define UART_TX_FIFO_INT_POINTER  (1 << 1)      /* 送信FIFOポインタ・モード */
#define UART_TX_FIFO_INT_PENDING  (0 << 1)      /* 送信FIFO保留モード */
#define UART_RX_FIFO_INT_POINTER  (1 << 0)      /* 受信FIFOポインタ・モード */
#define UART_RX_FIFO_INT_PENDING  (0 << 0)      /* 受信FIFO保留モード */

/*--------------------------------------------------------------------*/
/* UARTBn FIFO制御レジスタ1(UBnFIC1)ビット定義                          */
/*--------------------------------------------------------------------*/
#define UART_FIFO_TIMEOUT_ON    (1 << 7)        /* 受信タイムアウト許可 */
#define UART_FIFO_TIMEOUT_OFF   (0 << 7)        /* 受信タイムアウト禁止 */

/* UARTBn FIFO制御レジスタ2(UBnFIC2)ビット定義 */
/* トリガ設定数を即値で代入する場合 */
#define UART_TX_FIFO_TRIGGER(byte) ((byte-1) << 8)
#define UART_RX_FIFO_TRIGGER(byte) (byte-1)

/*--------------------------------------------------------------------*/
/* その他の定義                                                       */
/*--------------------------------------------------------------------*/
#define UART_RX_BUF_SIZE  256           /* 受信バッファ・サイズ */
#define EMPTY   0x00                    /* 空っぽ */
#define DISABLE 0                       /* 禁止 */
#define ENABLE  1                       /* 許可 */
#define NORMAL  0                       /* 通常 */
#define CLEAR   1                       /* クリア */
/*--------------------------------------------------------------------*/
/* 割り込み制御レジスタ(xxICn)ビット定義                               */
/*--------------------------------------------------------------------*/
#define IREQ_FLAG   0x80                /* 割り込み要求フラグ */

#define MASK_ON     -1                  /* 割り込みマスク */
#define MASK_OFF    0                   /* 割り込みマスク解除 */
#define PRIORITY(pri) (pri+1)           /* 割り込み優先順位(0-7) */
```

付　録　改版履歴

付. 1　本版で改訂された主な箇所

箇　所	内　　容
全般	・μPD703111AF1-10-GA3, 703111AF1-13-GA3, 703111AF1-15-GA3を削除 ・μPD703111AGM-10-UEU-A, 703111AGM-13-UEU-A, 703111AGM-15-UEU-Aを追加 ・V850E2/ME3（μPD703500）を追加 ・240ピン・プラスチックFBGA（16×16）を削除
p.11, 12	表1－1　V850E/ME2とV850E2/ME3の製品間の違い　追加
p.13, 15	1. 2. 1　特　　徴　○命令数，○パッケージを変更
p.22	1. 3　V850E2/ME3　追加
p.30-98	第2章　バス・インタフェース接続回路例　記述追加
p.99	第3章　内蔵周辺機能の接続回路例　記述追加
p.105	第4章　アプリケーション例　記述追加
p.295	付. 2　前版までの改版履歴　追加

★ 付. 2　前版までの改版履歴

前版までの改版履歴を次に示します。なお，適用箇所は各版での章を示します。

版　数	前版までの改版内容	適用箇所
第2版	μPD703111を削除し，μPD703111Aを追加 FBGAパッケージを追加	全般
	図2－52　①MB29PL320TE80の$\overline{WE}$ハイ・レベル・パルス幅：t_{WWRH}の計算式を修正	第2章　バス・インタフェース接続回路例
	付 録　改版履歴を追加	付録　改版履歴

【発　行】
NECエレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753
電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。
URL（アドレス）　http://www.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）
電　話　：044−435−9494
E-mail　：info@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。

C04.2T