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/MA1, V850E/MA2, V850E/MA3, V850E/ME2

32ビット・シングルチップ・マイクロコンピュータ

PCIホスト・ブリッジ・マクロ編

V850E/MA1：
μPD703103A
μPD703105A
μPD703106A
μPD703106A(A)
μPD703107A
μPD703107A(A)
μPD70F3107A
μPD70F3107A(A)

V850E/MA2：
μPD703108

V850E/MA3：
μPD703131A
μPD703131AY
μPD703132A
μPD703132AY
μPD703133A
μPD703133AY
μPD703134A
μPD703134AY
μPD70F3134A
μPD70F3134AY

V850E/ME2：
μPD703111A

資料番号　U17121JJ1V1AN00（第1版）
発行年月　May 2004 NS　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。

注意：μPD703131AY, 703132AY, 703133AY, 703134AY, 70F3134AYはI^2Cバス・インタフェース回路を内蔵しています。

I^2Cバス・インタフェースを使用される場合には，カスタム・コードをご発注いただく時に，事前にその旨ご申告下さい。申告に基づき，以下の特典が受けられます。

当社のI^2Cバス対応部品をご購入いただくことにより，これらの部品をI^2Cシステムに使用する実施権がフィリップス社I^2C特許に基づき許諾されることになります。ただし，これらのI^2Cシステムはフィリップス社によって設定されたI^2C標準規格に合致しているものとします。

Purchase of NEC Electronics I^2C components conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

本製品が外国為替及び外国貿易法の規定により規制貨物等（または役務）に該当するか否かは，ユーザ（仕様を決定した者）が判定してください。該当する場合，日本国外に輸出する際には日本国政府の輸出許可が必要です。

非該当品 ：μPD703103A, 70F3107A, 70F3107A(A),
μPD703108,
μPD70F3134A,
μPD703111A

ユーザ判定品 ：μPD703105A, 703106A, 703106A(A), 703107A, 703107A(A),
μPD703131A, 703131AY, 703132A, 703132AY, 703133A, 703133AY, 703134A, 703134AY

- **本資料に記載されている内容は2004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/MA1, V850E/MA2, V850E/MA3, V850E/ME2, **およびPCIバスの機能を理解し，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。**

目　　的　**このアプリケーション・ノートでは，**V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2**と，PCIホスト・ブリッジ・マクロを用いたシステム例を取り上げ，**PCI**ホスト・ブリッジ・マクロと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。**

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。**

- **各製品の概要**
- PCI**ホスト・ブリッジ・マクロの概要**
- PCI**ホスト・ブリッジ・マクロの仕様**
- FPGA**への組み込み構成例**
- **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/MA1, V850E/MA2, V850E/MA3, V850E/ME2**のハードウエア機能，および電気的特性を知りたいとき**
→個別のユーザーズ・マニュアル　ハードウエア編**を参照してください。**

V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2**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　アーキテクチャ編を参照してください。**

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	**：**$\overline{\text{xxx}}$**（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/MA1, V850E/MA2, V850E/MA3, V850E/ME2**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/MA1　**ユーザーズ・マニュアル　ハードウエア編**	U14359J
V850E/MA1　**アプリケーション・ノート　ハードウエア編**	U15179J
V850E/MA2　**ユーザーズ・マニュアル　ハードウエア編**	U14980J
V850E/MA3　**ユーザーズ・マニュアル　ハードウエア編**	U16397J
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**	U16794J
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
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**	**このマニュアル**

開発ツールに関する資料（ユーザーズ・マニュアル）

資　料　名		資料番号
IE-V850E-MC, IE-V850E-MC-A　**インサーキット・エミュレータ**		U14487J
IE-703107-MC-EM1　**インサーキット・エミュレータ・オプション・ボード**		U14481J
IE-V850E1-CD-NW　PCMCIA**カード型オンチップ・ディバグ・エミュレータ**		U16647J
CA850　Ver.2.50　C**コンパイラ・パッケージ**	**操作編**	U16053J
	C**言語編**	U16054J
	アセンブリ言語編	U16042J
PM plus　Ver.5.20		U16934J
ID850　Ver.2.50　**統合ディバッガ**	**操作編**	U16217J
ID850NW　Ver.2.51　**統合ディバッガ**	**操作編**	U16454J
SM850　Ver.2.40　**システム・シミュレータ**	**操作編**	U15182J
SM850　Ver.2.00**以上　システム・シミュレータ**	**外部部品ユーザ・オープン・インタフェース仕様編**	U14873J
RX850　Ver.3.13**以上　リアルタイム**OS	**基礎編**	U13430J
	インストレーション編	U13410J
	テクニカル編	U13431J
RX850 Pro　Ver.3.15　**リアルタイム**OS	**基礎編**	U13773J
	インストレーション編	U13774J
	テクニカル編	U13772J
RX-NET　TCP/IP**ライブラリ**		U15083J
RD850　Ver.3.01　**タスク・ディバッガ**		U13737J
RD850 Pro　Ver.3.01　**タスク・ディバッガ**		U13916J
AZ850　Ver.3.0　**システム・パフォーマンス・アナライザ**		U14410J
PG-FP4　**フラッシュ・メモリ・プログラマ**		U15260J

目　　次

第1章　各製品の概要

V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2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。この章では，各製品の概要を簡単に説明します。

1.1　概　　説

V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2は，システム・オン・チップ時代のシステムLSIの核となるASIC用32ビットRISC命令型CPUコア「V850E1 CPU」を搭載した32ビット・シングルチップ・マイクロコンピュータです。内蔵メモリや，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，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ます。

1.2 特　　徴

愛　　称			V850E/MA1				V850E/MA2	V850E/MA3				V850E/ME2
最大動作周波数			50 MHz				40 MHz	80 MHz				150 MHz
内部メモリ（Kバイト）	マスクROM		−	128	256	−	−	256	512	−		−
	フラッシュ・メモリ		−			256	−	−		512		−
	RAM		4	10			4	16	32	16	32	命令RAM：128 データRAM：16
キャッシュ（Kバイト）							−					命令キャッシュ：8
外部バス	バス・タイプ		セパレート				セパレート	セパレート／ マルチプレクス				セパレート
	アドレス・バス		26ビット				25ビット	26ビット				26ビット
	データ・バス		8/16ビット				8/16ビット	8/16ビット				16/32ビット
	チップ・セレクト信号		8本				4本	8本				8本
メモリ・コントローラ			SDRAM, EDO DRAM, SRAMなど				SDRAM, SRAMなど					
割り込み		外部[注1]	17（17）				4（4）	26（26）				40（31）
		内部	41				27	49				59
DSP機能	32×32→64		20-40 ns（50 MHz）				25-50 ns（40 MHz）	12.5-25 ns（80 MHz）				6.7-13.3 ns（150 MHz）
	32×32＋32→32		60 ns（50 MHz）				75 ns（40 MHz）	37.5 ns（80 MHz）				20 ns（150 MHz）
16ビット・タイマ	TMC		4 ch				2 ch	−				6 ch
	TMP		−				−	3 ch				−
	TMQ		−				−	1 ch				−
	インターバル・タイマ		4 ch				4 ch	4 ch				4 ch
	アップ・ダウン・カウンタ		−				−	1 ch				2 ch
ウォッチドッグ・タイマ			−				−	1 ch				−
シリアル・インタフェース	CSI		1 ch				−	−				1 ch
	UART		1 ch				−	−				1 ch
	CSI/UART		2 ch				2 ch	3 ch				1 ch
	UART/I^2C		−				−	1 ch[注2]				−
10ビットA/Dコンバータ			8 ch				4 ch	8 ch				8 ch
8ビットD/Aコンバータ			−				−	2 ch				−
DMAコントローラ			4 ch				4 ch	4 ch				4 ch
ポート	CMOS入力		9				5	11				7
	CMOS入出力		106				74	101				77
ディバグ機能			−				−	あり（RUN，ブレーク）				あり（RUN，ブレーク，トレース）
その他周辺機能			PWM×2 ch				−	ROMコレクション機能				USBファンクション， SSCG， PWM×2 ch
電源電圧			3.0〜3.6 V					2.3〜2.7 V（内部） 3.0〜3.6 V（外部）				1.5 V（内部） 3.3 V（外部）
消費電力（マスク版TYP.）			528 mW				416 mW	575 mW				200 mW
パッケージ			144ピンLQFP（20×20） 161ピンFBGA（13×13）				100ピンLQFP （14×14）	144ピンLQFP（20×20） 161ピンFBGA（13×13）				176ピンLQFP（20×20） 240ピンFBGA（16×16）
動作周囲温度			T_A = −40〜+85 ℃									T_A = −40〜+85 ℃（133 MHz時） T_A = −40〜+70 ℃（150 MHz時）

注1.　（　）内は，STOPモード解除可能な外部割り込み本数です。

2.　I^2C内蔵品（Y品）だけ有効です。

1.3　オーダ情報

（1）V850E/MA1

品　　名	パッケージ	内蔵ROM
μPD703103AGJ-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	ROMレス
μPD703105AGJ-xxx-UEN	〃	マスクROM（128 Kバイト）
μPD703106AGJ-xxx-UEN	〃	〃
μPD703106AGJ(A)-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD703106AF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703107AGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	マスクROM（256 Kバイト）
μPD703107AGJ(A)-xxx-UEN	〃	〃
μPD703107AF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD70F3107AGJ-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	フラッシュ・メモリ（512Kバイト）
μPD70F3107AGJ(A)-UEN	〃	〃
μPD70F3107AF1-EN4	161ピン・プラスチックFBGA（13×13）	〃

（2）V850E/MA2

品　　名	パッケージ	内蔵ROM
μPD703108GC-8EU	100ピン・プラスチックLQFP（ファインピッチ）（14×14）	ROMレス

（3）V850E/MA3

品　名	パッケージ	内蔵ROM
μPD703131AGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	マスクROM（256 Kバイト）
μPD703131AF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703131AYGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD703131AYF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703132AGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD703132AF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703132AYGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD703132AYF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703133AGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	マスクROM（512 Kバイト）
μPD703133AF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703133AYGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD703133AYF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703134AGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD703134AF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD703134AYGJ-xxx-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD703134AYF1-xxx-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD70F3134AGJ-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	フラッシュ・メモリ（512Kバイト）
μPD70F3134AF1-EN4	161ピン・プラスチックFBGA（13×13）	〃
μPD70F3134AYGJ-UEN	144ピン・プラスチックLQFP（ファインピッチ）（20×20）	〃
μPD70F3134AYF1-EN4	161ピン・プラスチックFBGA（13×13）	〃

（4）V850E/ME2

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-13-UEU	〃	133 MHz
μPD703111AGM-15-UEU	〃	150 MHz
μPD703111AF1-10-GA3	240ピン・プラスチックFBGA（16×16）	100 MHz
μPD703111AF1-13-GA3	〃	133 MHz
μPD703111AF1-15-GA3	〃	150 MHz

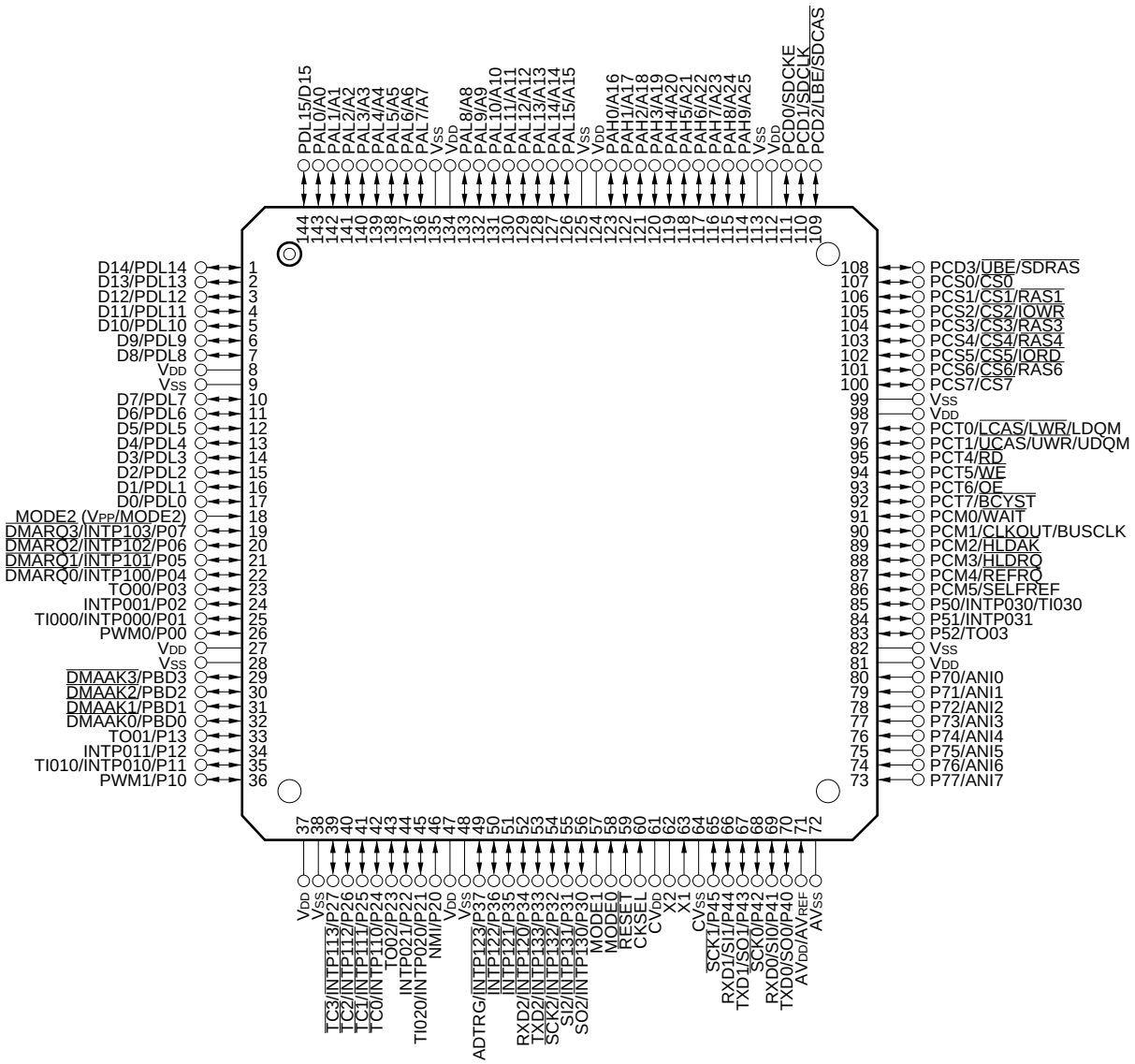
1. 4　端子接続図

（1）V850E/MA1

・144ピン・プラスチックLQFP（ファインピッチ）（20×20）

μPD703103AGJ-UEN	μPD703106AGJ(A)-xxx-UEN	μPD70F3107AGJ-UEN
μPD703105AGJ-xxx-UEN	μPD703107AGJ-xxx-UEN	μPD70F3107AGJ(A)-UEN
μPD703106AGJ-xxx-UEN	μPD703107AGJ(A)-xxx-UEN	

Top View



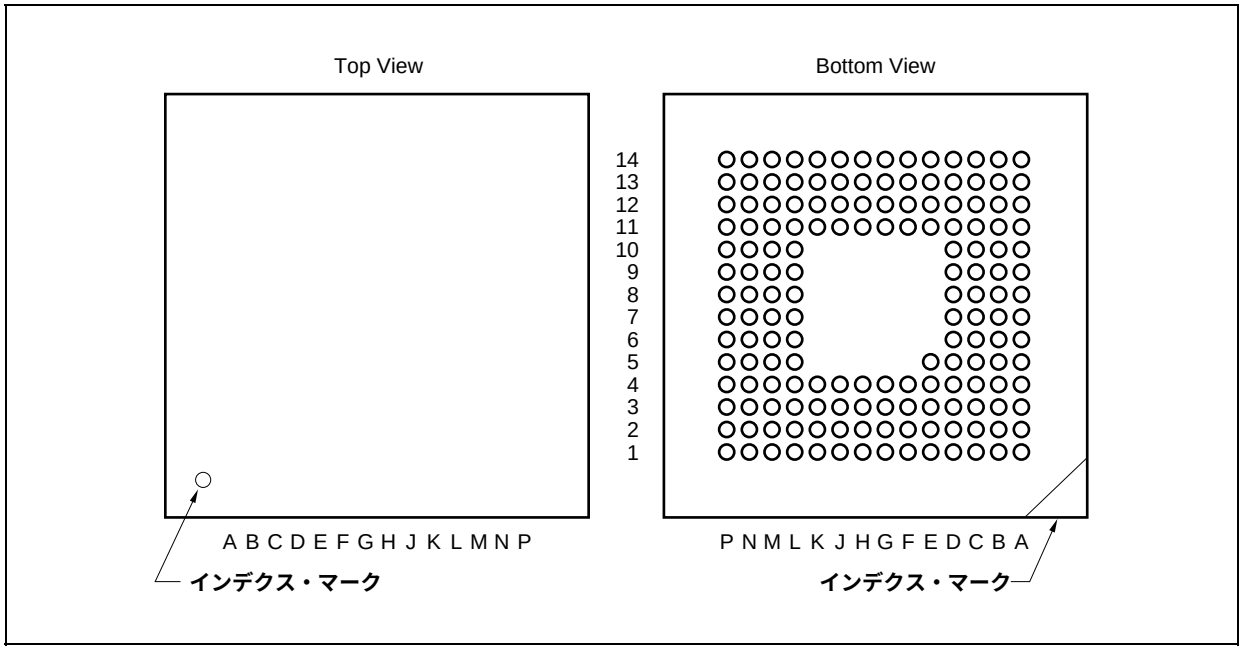
備考　（　）内はμPD70F3107Aでの端子名です。

・161ピン・プラスチックFBGA（13×13）

μPD703106AF1-xxx-EN4

μPD703107AF1-xxx-EN4

μPD70F3107AF1-EN4



（1/2）

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
A1	−	B9	A18/PAH2	D3	D14/PDL14
A2	D15/PDL15	B10	A21/PAH5	D4	A3/PAL3
A3	A2/PAL2	B11	A25/PAH9	D5	A6/PAL6
A4	A5/PAL5	B12	SDCLK/PCD1	D6	A10/PAL10
A5	−	B13	$\overline{CS1}$/$\overline{RAS1}$/PCS1	D7	A14/PAL14
A6	A9/PAL9	B14	−	D8	A16/PAH0
A7	A12/PAL12	C1	−	D9	A20/PAH4
A8	A15/PAL15	C2	D9/PDL9	D10	A23/PAH7
A9	A17/PAH1	C3	D13/PDL13	D11	SDCKE/PCD0
A10	−	C4	A1/PAL1	D12	$\overline{CS0}$/PCS0
A11	A24/PAH8	C5	A7/PAL7	D13	$\overline{CS5}$/$\overline{IORD}$/PCS5
A12	V_{DD}	C6	V_{DD}	D14	−
A13	$\overline{LBE}$/$\overline{SDCAS}$/PCD2	C7	A11/PAL11	E1	D5/PDL5
A14	$\overline{UBE}$/$\overline{SDRAS}$/PCD3	C8	V_{DD}	E2	D7/PDL7
B1	−	C9	A19/PAH3	E3	D8/PDL8
B2	D12/PDL12	C10	A22/PAH6	E4	D11/PDL11
B3	A0/PAL0	C11	V_{SS}	E5	−
B4	A4/PAL4	C12	$\overline{CS3}$/$\overline{RAS3}$/PCS3	E11	$\overline{CS6}$/$\overline{RAS6}$/PCS6
B5	V_{SS}	C13	$\overline{CS2}$/$\overline{IOWR}$/PCS2	E12	$\overline{CS4}$/$\overline{RAS4}$/PCS4
B6	A8/PAL8	C14	−	E13	$\overline{CS7}$/PCS7
B7	A13/PAL13	D1	V_{SS}	E14	V_{SS}
B8	V_{SS}	D2	D10/PDL10	F1	D2/PDL2

(2/2)

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
F2	D3/PDL3	K2	V_{SS}	M12	ANI6/P76
F3	D4/PDL4	K3	$\overline{DMAAK1}$/PBD1	M13	ANI5/P75
F4	V_{DD}	K4	$\overline{DMAAK3}$/PBD3	M14	−
F11	$\overline{RD}$/PCT4	K11	ANI1/P71	N1	−
F12	V_{DD}	K12	ANI0/P70	N2	PWM1/P10
F13	$\overline{LCAS}$/$\overline{LWR}$/LDQM/PCT0	K13	V_{SS}	N3	$\overline{TC3}$/$\overline{INTP113}$/P27
F14	$\overline{UCAS}$/$\overline{UWR}$/UDQM/PCT1	K14	V_{DD}	N4	$\overline{TC0}$/$\overline{INTP110}$/P24
G1	MODE2（MODE2/V_{PP}）	L1	−	N5	NMI/P20
G2	$\overline{DMARQ3}$/$\overline{INTP103}$/P07	L2	$\overline{DMAAK2}$/PBD2	N6	ADTRG/$\overline{INTP123}$/P37
G3	D0/PDL0	L3	TI010/INTP010/P11	N7	TXD2/$\overline{INTP133}$/P33
G4	D6/PDL6	L4	$\overline{DMAAK0}$/PBD0	N8	SO2/$\overline{INTP130}$/P30
G11	$\overline{WAIT}$/PCM0	L5	TO02/P23	N9	X2
G12	$\overline{WE}$/PCT5	L6	V_{DD}	N10	CV_{SS}
G13	$\overline{BCYST}$/PCT7	L7	$\overline{INTP122}$/P36	N11	$\overline{SCK0}$/P42
G14	$\overline{OE}$/PCT6	L8	SI2/$\overline{INTP131}$/P31	N12	AV_{DD}/AV_{REF}
H1	$\overline{DMARQ2}$/$\overline{INTP102}$/P06	L9	$\overline{RESET}$	N13	AV_{SS}
H2	$\overline{DMARQ1}$/$\overline{INTP101}$/P05	L10	TXD1/SO1/P43	N14	−
H3	$\overline{DMARQ0}$/$\overline{INTP100}$/P04	L11	ANI7/P77	P1	V_{DD}
H4	D1/PDL1	L12	ANI4/P74	P2	V_{SS}
H11	$\overline{REFRQ}$/PCM4	L13	ANI3/P73	P3	$\overline{TC1}$/$\overline{INTP111}$/P25
H12	$\overline{HLDRQ}$/PCM3	L14	ANI2/P72	P4	INTP021/P22
H13	$\overline{HLDAK}$/PCM2	M1	−	P5	−
H14	CLKOUT/BUSCLK/PCM1	M2	INTP011/P12	P6	$\overline{INTP121}$/P35
J1	TO00/P03	M3	TO01/P13	P7	$\overline{SCK2}$/$\overline{INTP132}$/P32
J2	TI000/INTP000/P01	M4	$\overline{TC2}$/$\overline{INTP112}$/P26	P8	MODE1
J3	V_{DD}	M5	TI020/INTP020/P21	P9	CV_{DD}
J4	INTP001/P02	M6	V_{SS}	P10	X1
J11	TO03/P52	M7	RXD2/$\overline{INTP120}$/P34	P11	−
J12	TI030/INTP030/P50	M8	MODE0	P12	RXD1/SI1/P44
J13	SELFREF/PCM5	M9	CKSEL	P13	RXD0/SI0/P41
J14	INTP031/P51	M10	$\overline{SCK1}$/P45	P14	−
K1	PWM0/P00	M11	TXD0/SO0/P40		

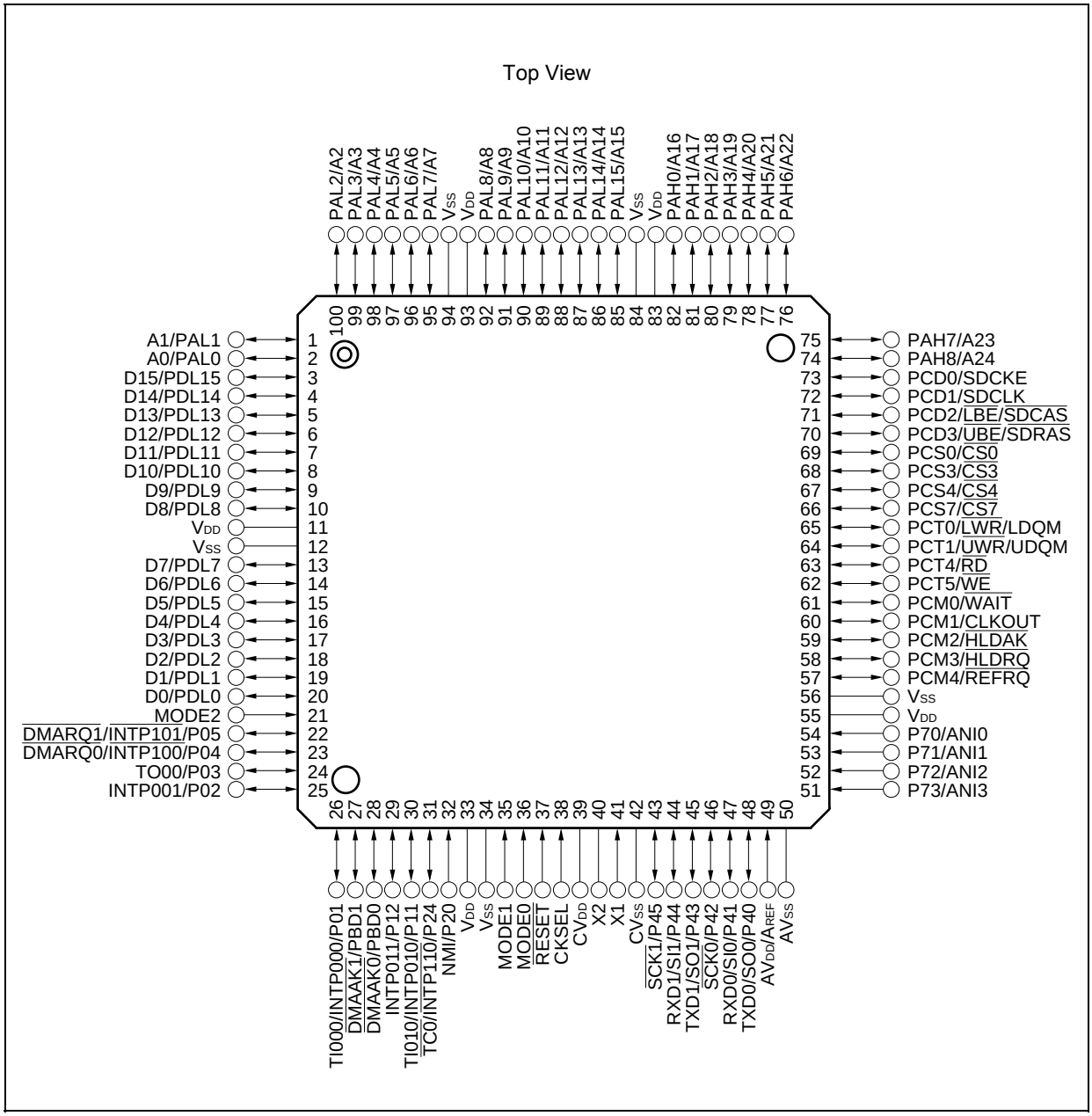
備考1. A1, A5, A10, B1, B14, C1, C14, D14, E5, L1, M1, M14, N1, N14, P5, P11, P14**の端子は，オープンにしてください。**

2. **（　）は**μPD70F3107A**での端子名です。**

（2）V850E/MA2

・100ピン・プラスチックLQFP（ファインピッチ）（14×14）

μPD703108GC-8EU

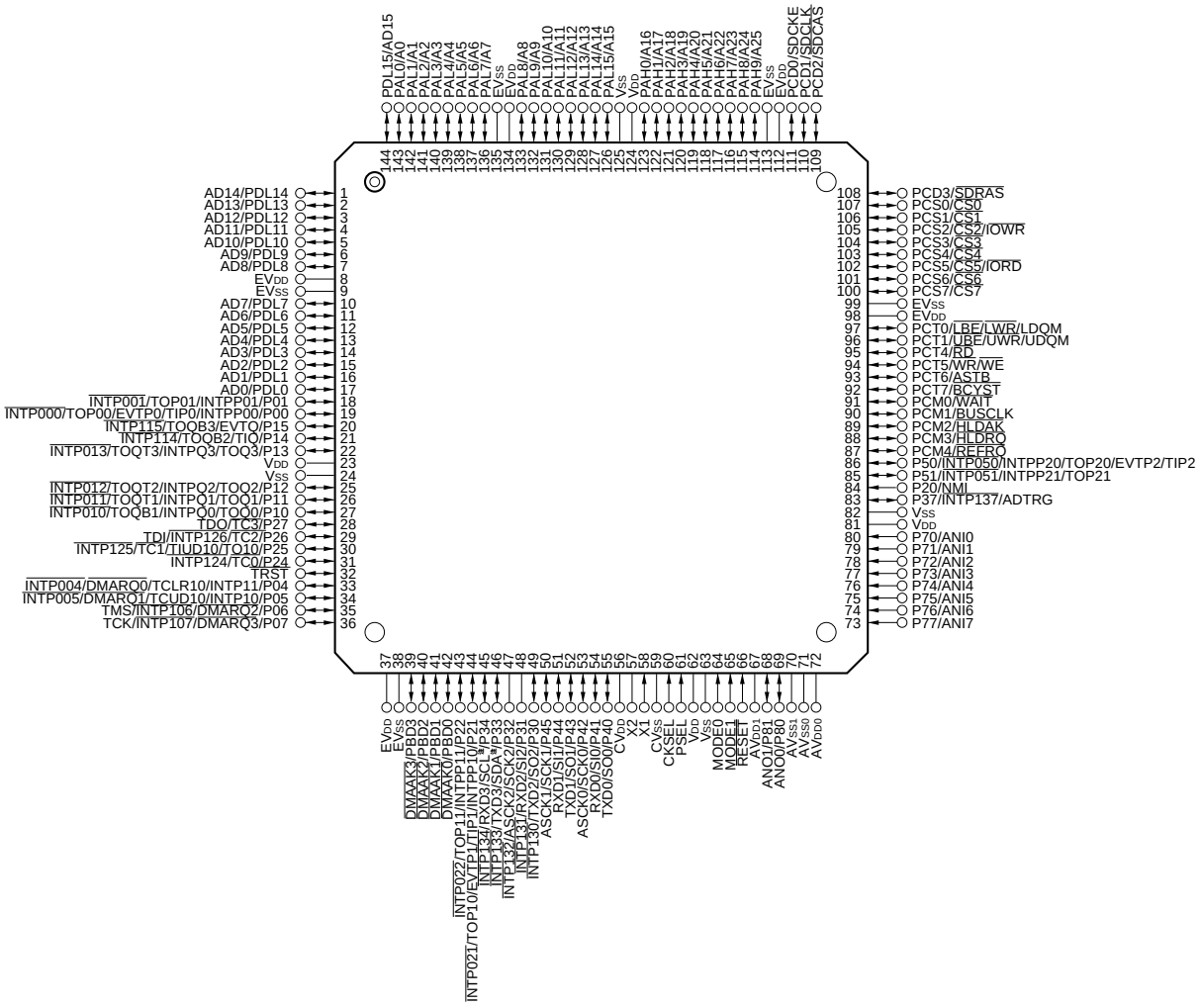


（3）V850E/MA3

・144ピン・プラスチックLQFP（ファインピッチ）（20×20）

μPD703131AGJ-xxx-UEN	μPD703133AGJ-xxx-UEN	μPD70F3134AGJ-UEN
μPD703131AYGJ-xxx-UEN	μPD703133AYGJ-xxx-UEN	μPD70F3134AYGJ-UEN
μPD703132AGJ-xxx-UEN	μPD703134AGJ-xxx-UEN	
μPD703132AYGJ-xxx-UEN	μPD703134AYGJ-xxx-UEN	

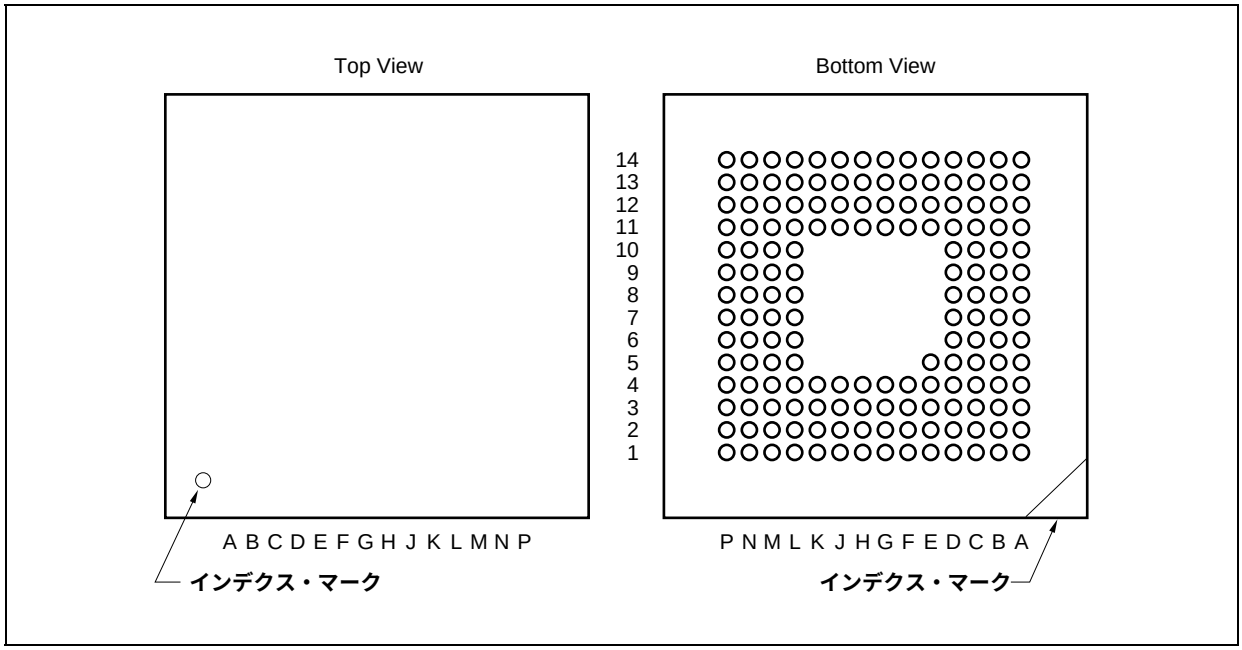
Top View



注　SCL, SDA**は，**μPD703131Y, 703132Y, 703133Y, 703134Y, 703137Y, 70F3134Y**だけ有効です。**

・161ピン・プラスチックFBGA（13×13）

μPD703131AF1-EN4　μPD703133AF1-xxx-EN4　μPD70F3134AF1-EN4
μPD703131AYF1-xxx-EN4　μPD703133AYF1-xxx-EN4　μPD70F3134AYF1-EN4
μPD703132AF1-xxx-EN4　μPD703134AF1-xxx-EN4
μPD703132AYF1-xxx-EN4　μPD703134AYF1-xxx-EN4



(1/2)

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
A1	EV_{SS}	B8	V_{SS}	D1	EV_{SS}
A2	AD15/PDL15	B9	A18/PAH2	D2	AD10/PDL10
A3	A2/PAL2	B10	A21/PAH5	D3	AD14/PDL14
A4	A5/PAL5	B11	A25/PAH9	D4	A3/PAL3
A5	EV_{SS}	B12	SDCLK/PCD1	D5	A6/PAL6
A6	A9/PAL9	B13	$\overline{CS1}$/PCS1	D6	A10/PAL10
A7	A12/PAL12	B14	EV_{SS}	D7	A14/PAL14
A8	A15/PAL15	C1	EV_{SS}	D8	A16/PAH0
A9	A17/PAH1	C2	AD9/PDL9	D9	A20/PAH4
A10	−	C3	AD13/PDL13	D10	A23/PAH7
A11	A24/PAH8	C4	A1/PAL1	D11	SDCKE/PCD0
A12	EV_{DD}	C5	A7/PAL7	D12	$\overline{CS0}$/PCS0
A13	$\overline{SDCAS}$/PCD2	C6	EV_{DD}	D13	$\overline{CS5}$/$\overline{IORD}$/PCS5
A14	$\overline{SDRAS}$/PCD3	C7	A11/PAL11	D14	EV_{SS}
B1	EV_{SS}	C8	V_{DD}	E1	AD5/PDL5
B2	AD12/PDL12	C9	A19/PAH3	E2	AD7/PDL7
B3	A0/PAL0	C10	A22/PAH6	E3	AD8/PDL8
B4	A4/PAL4	C11	EV_{SS}	E4	AD11/PDL11
B5	EV_{SS}	C12	$\overline{CS3}$/PCS3	E5	−
B6	A8/PAL8	C13	$\overline{CS2}$/$\overline{IOWR}$/PCS2	E11	$\overline{CS6}$/PCS6
B7	A13/PAL13	C14	EV_{SS}	E12	$\overline{CS4}$/PCS4

(2/2)

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
E13	$\overline{CS7}$/PCS7	J14	NMI/P20	M11	AV_{SS0}
E14	EV_{SS}	K1	TOQT1/$\overline{INTP011}$/INTPQ1/TOQ1/P11	M12	ANI6/P76
F1	AD2/PDL2	K2	$\overline{TC3}$/TDO/P27	M13	ANI5/P75
F2	AD3/PDL3	K3	$\overline{TC0}$/$\overline{INTP124}$/P24	M14	−
F3	AD4/PDL4	K4	$\overline{TC2}$/TDI/$\overline{INTP126}$/P26	N1	EV_{SS}
F4	EV_{DD}	K11	ANI1/P71	N2	$\overline{DMARQ3}$/TCK/$\overline{INTP107}$/P07
F11	$\overline{RD}$/PCT4	K12	ANI0/P70	N3	$\overline{DMAAK3}$/PBD3
F12	EV_{DD}	K13	V_{SS}	N4	$\overline{DMAAK0}$/PBD0
F13	$\overline{LBE}$/$\overline{LWR}$/LDQM/PCT0	K14	V_{DD}	N5	TXD3/SDA[注]/$\overline{INTP133}$/P33
F14	$\overline{UBE}$/$\overline{UWR}$/UDQM/PCT1	L1	EV_{SS}	N6	TXD2/SO2/$\overline{INTP130}$/P30
G1	TOP01/$\overline{INTP001}$/INTPP01/P01	L2	TC1/TIUD10/TO10/$\overline{INTP125}$/P25	N7	ASCK0/SCK0/P42
G2	TOP00/$\overline{INTP000}$/EVTP0/TIP0/INTPP00/P00	L3	$\overline{DMARQ2}$/TMS/$\overline{INTP106}$/P06	N8	V_{SS}
G3	AD0/PDL0	L4	$\overline{TRST}$	N9	X2
G4	AD6/PDL6	L5	TOP11/INTPP11/$\overline{INTP022}$/P22	N10	CV_{SS}
G11	$\overline{WAIT}$/PCM0	L6	ASCK2/SCK2/$\overline{INTP132}$/P32	N11	ANO1/P81
G12	$\overline{WR}$/$\overline{WE}$/PCT5	L7	ASCK1/SCK1/P45	N12	AV_{SS1}
G13	$\overline{BCYST}$/PCT7	L8	TXD0/SO0/P40	N13	AV_{DD1}
G14	ASTB/PCT6	L9	MODE0	N14	−
H1	TOQB3/$\overline{INTP115}$/EVTQ/P15	L10	AV_{DD0}	P1	EV_{DD}
H2	TOQB2/$\overline{INTP114}$/TIQ/P14	L11	ANI7/P77	P2	EV_{SS}
H3	TOQT3/$\overline{INTP013}$/INTPQ3/TOQ3/P13	L12	ANI4/P74	P3	$\overline{DMAAK1}$/PBD1
H4	AD1/PDL1	L13	ANI3/P73	P4	TOP10/INTPP10/EVTP1/TIP1/$\overline{INTP021}$/P21
H11	$\overline{REFRQ}$/PCM4	L14	ANI2/P72	P5	EV_{SS}
H12	$\overline{HLDRQ}$/PCM3	M1	EV_{SS}	P6	RXD1/SI1/P44
H13	$\overline{HLDAK}$/PCM2	M2	$\overline{DMARQ1}$/TCUD10/INTP10/$\overline{INTP005}$/P05	P7	RXD0/SI0/P41
H14	BUSCLK/PCM1	M3	$\overline{DMARQ0}$/INTP11/TCLR10/$\overline{INTP004}$/P04	P8	PSEL
J1	V_{DD}	M4	$\overline{DMAAK2}$/PBD2	P9	CV_{DD}
J2	TOQT2/$\overline{INTP012}$/INTPQ2/TOQ2/P12	M5	RXD3/SCL[注]/$\overline{INTP134}$/P34	P10	X1
J3	TOQB1/$\overline{INTP010}$/INTPQ0/TOQ0/P10	M6	RXD2/SI2/$\overline{INTP131}$/P31	P11	−
J4	V_{SS}	M7	TXD1/SO1/P43	P12	$\overline{RESET}$
J11	ADTRG/$\overline{INTP137}$/P37	M8	V_{DD}	P13	ANO0/P80
J12	TOP21/INTPP21/$\overline{INTP051}$/P51	M9	CKSEL	P14	−
J13	TOP20/INTPP20/EVTP2/TIP2/$\overline{INTP050}$/P50	M10	MODE1		

注　SCL, SDAは，μPD703131AY, 703132AY, 703133AY, 703134AY, 70F3134AYだけ有効です。

備考　A10, E5, M14, N14, P11, P14の端子は，オープンにしてください。

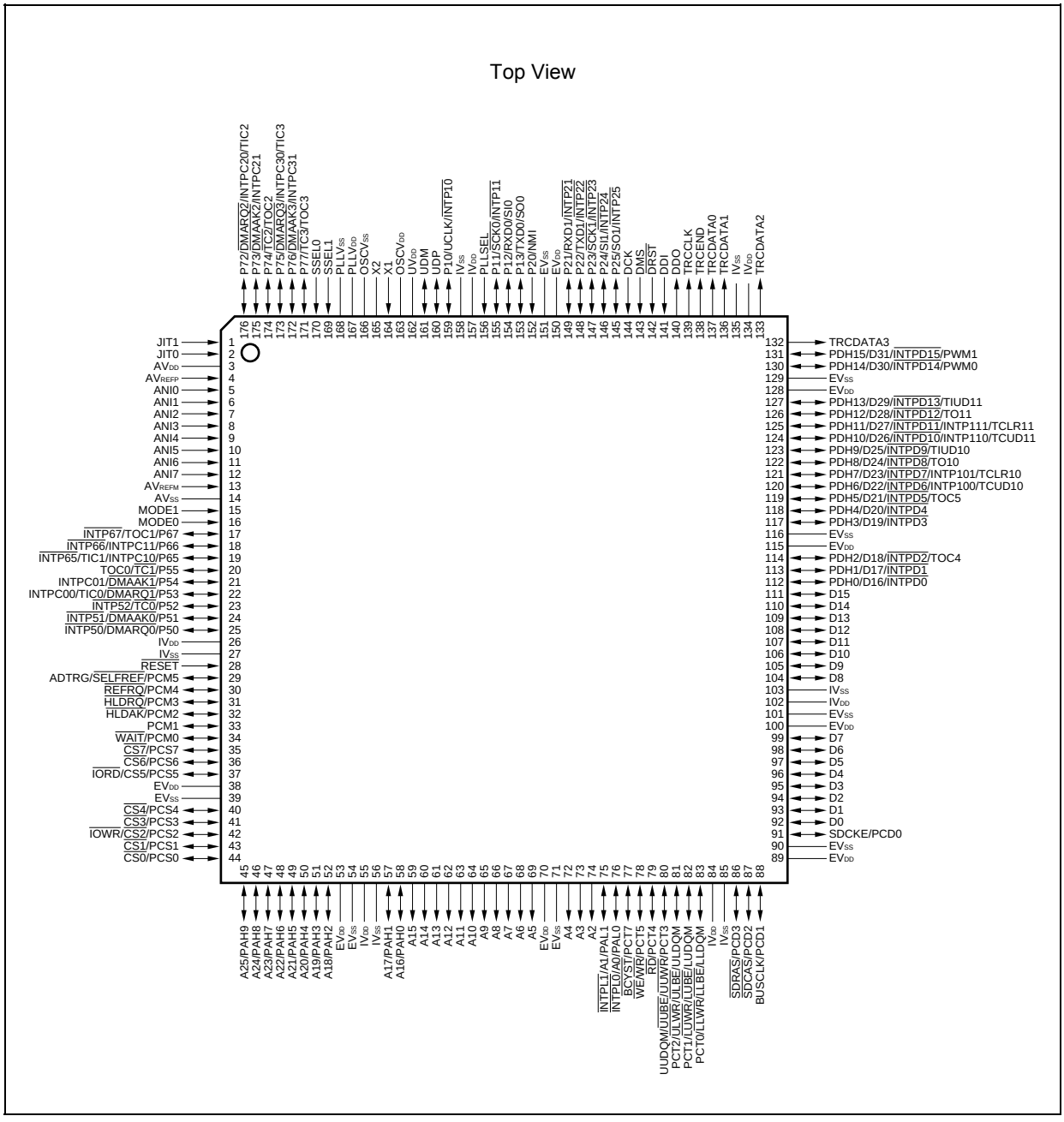
（4）V850E/ME2

・176ピン・プラスチックLQFP（ファインピッチ）（24×24）

μPD703111AGM-10-UEU

μPD703111AGM-13-UEU

μPD703111AGM-15-UEU

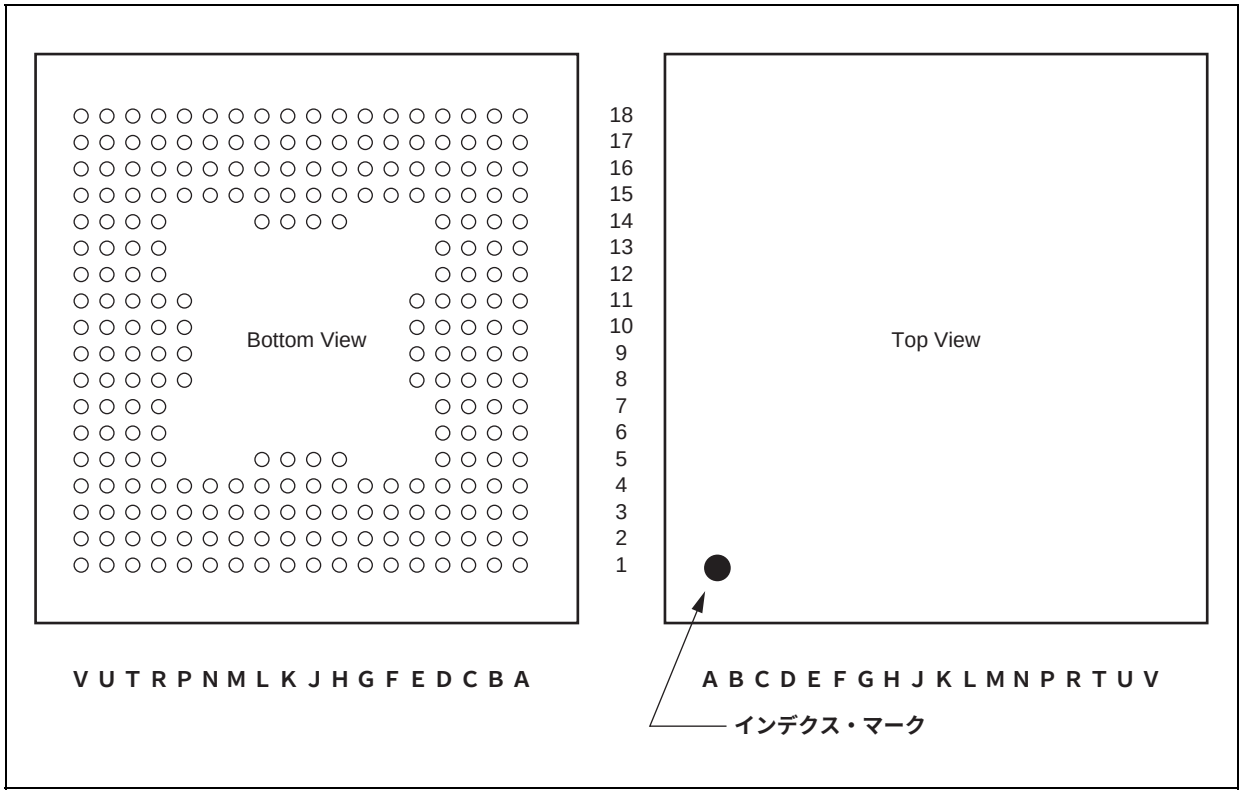


・240ピン・プラスチックFBGA（16×16）

μPD703111AF1-10-GA3

μPD703111AF1-13-GA3

μPD703111AF1-15-GA3



(1/2)

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
A1	−	C12	IV_{DD}	G3	EV_{SS}
A2	IV_{SS}	C13	PAH2/A18	G4	D7
A3	PCT0/$\overline{\text{LLWR}}$/$\overline{\text{LLBE}}$/LLDQM	C14	PAH4/A20	G15	PCM1
A4	−	C15	PAH6/A22	G16	PCM3/$\overline{\text{HLDRQ}}$
A5	PCT4/$\overline{\text{RD}}$	C16	−	G17	PCM4/$\overline{\text{REFRQ}}$
A6	−	C17	PCS0/$\overline{\text{CS0}}$	G18	PCM5/ADTRG/$\overline{\text{SELFREF}}$
A7	−	C18	−	H1	−
A8	EV_{DD}	D1	D0	H2	D8
A9	A9	D2	EV_{SS}	H3	D9
A10	−	D3	PCD0/SDCKE	H4	D10
A11	A14	D4	EV_{DD}	H5	IV_{SS}
A12	IV_{SS}	D5	PCT1/$\overline{\text{LUWR}}$/$\overline{\text{LUBE}}$/LUDQM	H14	−
A13	EV_{DD}	D6	−	H15	$\overline{\text{RESET}}$
A14	−	D7	PAL0/$\overline{\text{INTPL0}}$/A0	H16	IV_{SS}
A15	PAH5/A21	D8	A4	H17	−
A16	PAH7/A23	D9	A6	H18	IV_{DD}
A17	PAH9/A25	D10	−	J1	−
A18	−	D11	A13	J2	D11
B1	−	D12	EV_{SS}	J3	D12
B2	PCD1/BUSCLK	D13	PAH3/A19	J4	−
B3	PCD2/$\overline{\text{SDCAS}}$	D14	−	J5	D13
B4	−	D15	−	J14	−
B5	PCT3/$\overline{\text{UUWR}}$/$\overline{\text{UUBE}}$/UUDQM	D16	PCS2/$\overline{\text{CS2}}$/$\overline{\text{IOWR}}$	J15	P50/$\overline{\text{INTP50}}$/$\overline{\text{DMARQ0}}$
B6	PCT7/$\overline{\text{BCYST}}$	D17	PCS3/$\overline{\text{CS3}}$	J16	P51/$\overline{\text{INTP51}}$/$\overline{\text{DMAAK0}}$
B7	A2	D18	EV_{DD}	J17	P52/$\overline{\text{INTP52}}$/$\overline{\text{TC0}}$
B8	−	E1	D3	J18	P53/INTPC00/TIC0/$\overline{\text{DMARQ1}}$
B9	A8	E2	D2	K1	D14
B10	A12	E3	D1	K2	D15
B11	PAH0/A16	E4	−	K3	PDH0/D16/$\overline{\text{INTPD0}}$
B12	−	E8	A3	K4	PDH1/D17/$\overline{\text{INTPD1}}$
B13	−	E9	A5	K5	PDH2/D18/$\overline{\text{INTPD2}}$/TOC4
B14	−	E10	A10	K14	P55/TOC0/$\overline{\text{TC1}}$
B15	−	E11	PAH1/A17	K15	P54/INTPC01/$\overline{\text{DMAAK1}}$
B16	PAH8/A24	E15	PCS4/$\overline{\text{CS4}}$	K16	P65/$\overline{\text{INTP65}}$/INTPC10/TIC1
B17	−	E16	EV_{SS}	K17	P66/$\overline{\text{INTP66}}$/INTPC11
B18	PCS1/$\overline{\text{CS1}}$	E17	PCS5/$\overline{\text{CS5}}$/$\overline{\text{IORD}}$	K18	−
C1	−	E18	PCS6/$\overline{\text{CS6}}$	L1	EV_{DD}
C2	−	F1	D6	L2	−
C3	PCD3/$\overline{\text{SDRAS}}$	F2	D5	L3	EV_{SS}
C4	IV_{DD}	F3	D4	L4	PDH3/D19/$\overline{\text{INTPD3}}$
C5	PCT2/$\overline{\text{ULWR}}$/$\overline{\text{ULBE}}$/ULDQM	F4	−	L5	PDH4/D20/$\overline{\text{INTPD4}}$
C6	PCT5/$\overline{\text{WE}}$/$\overline{\text{WR}}$	F15	−	L14	MODE1
C7	PAL1/$\overline{\text{INTPL1}}$/A1	F16	PCS7/$\overline{\text{CS7}}$	L15	−
C8	EV_{SS}	F17	PCM0/$\overline{\text{WAIT}}$	L16	MODE0
C9	A7	F18	PCM2/$\overline{\text{HLDAK}}$	L17	−
C10	A11	G1	IV_{DD}	L18	P67/$\overline{\text{INTP67}}$/TOC1
C11	A15	G2	EV_{DD}	M1	−

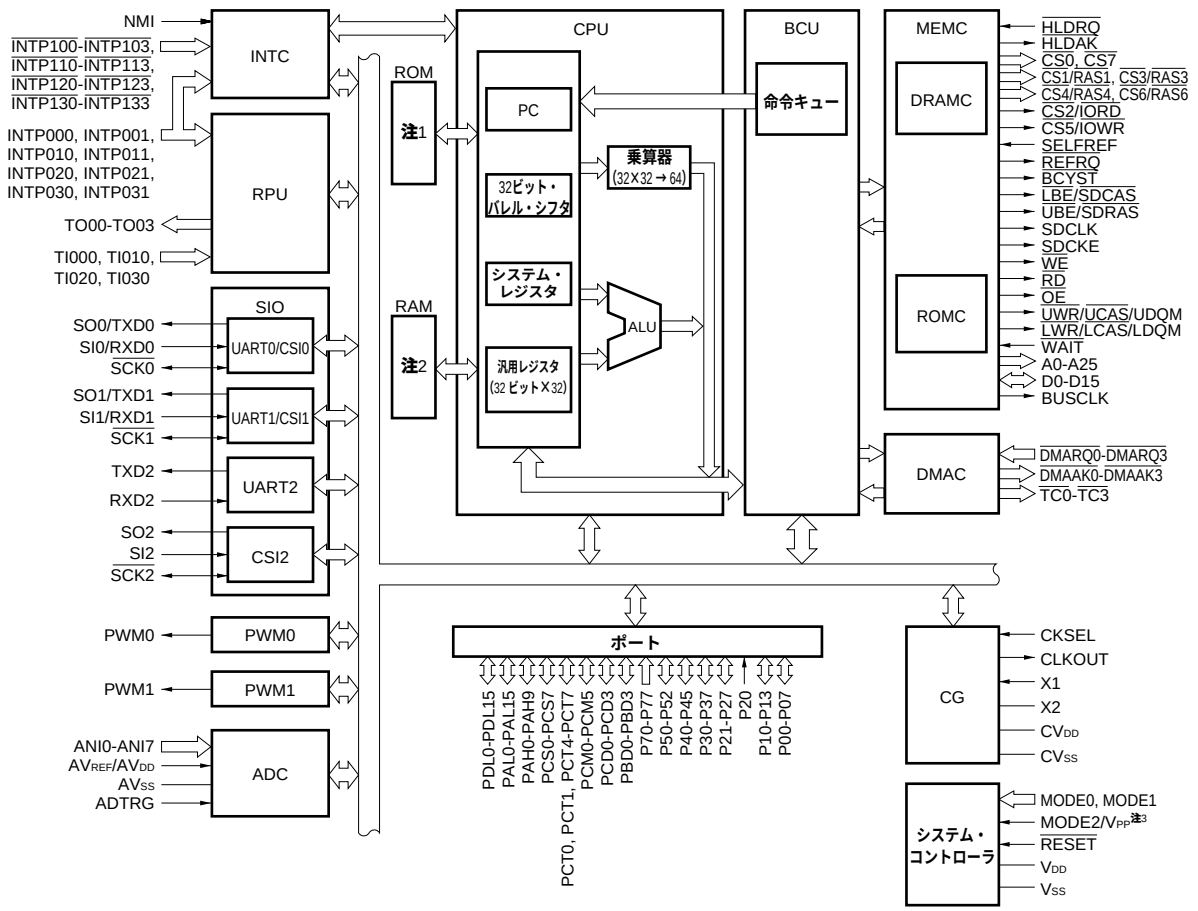
(2/2)

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
M2	PDH5/D21/INTPD5/TOC5	R7	DCK	U4	−
M3	PDH6/D22/INTPD6/INTP100/TCUD10	R8	EV_{DD}	U5	TRCCLK
M4	−	R9	P11/INTP11/SCK0	U6	DRST
M15	ANI6	R10	IV_{SS}	U7	P25/INTP25/SO1
M16	AV_{REFM}	R11	UDM	U8	P22/INTP22/TXD1
M17	ANI7	R12	X2	U9	EV_{SS}
M18	AV_{SS}	R13	$PLLV_{DD}$	U10	IV_{DD}
N1	PDH7/D23/INTPD7/INTP101/TCLR10	R14	SSEL0	U11	−
N2	PDH8/D24/INTPD8/TO10	R15	−	U12	$OSCV_{DD}$
N3	PDH9/D25/INTPD9/TIUD10	R16	AV_{REFP}	U13	−
N4	PDH10/D26/INTPD10/INTP110/TCUD11	R17	AV_{DD}	U14	−
N15	ANI2	R18	−	U15	P76/INTPC31/DMAAK3
N16	ANI3	T1	EV_{DD}	U16	P73/INTPC21/DMAAK2
N17	ANI4	T2	TRCDATA3	U17	P72/INTPC20/TIC2/DMARQ2
N18	ANI5	T3	−	U18	−
P1	−	T4	TRCDATA1	V1	−
P2	PDH11/D27/INTPD11/INTP111/TCLR11	T5	TRCEND	V2	TRCDATA2
P3	PDH13/D29/INTPD13/TIUD11	T6	DDI	V3	IV_{SS}
P4	−	T7	−	V4	TRCDATA0
P8	P23/INTP23/SCK1	T8	P21/INTP21/RXD1	V5	−
P9	P12/SI0/RXD0	T9	P20/NMI	V6	DMS
P10	−	T10	−	V7	P24/INTP24/SI1
P11	UV_{DD}	T11	UDP	V8	−
P15	−	T12	X1	V9	P13/SO0/TXD0
P16	ANI0	T13	$OSCV_{SS}$	V10	PLLSEL
P17	ANI1	T14	SSEL1	V11	P10/INTP10/UCLK
P18	−	T15	P75/INTPC30/TIC3/DMARQ3	V12	−
R1	PDH12/D28/INTPD12/TO11	T16	−	V13	−
R2	EV_{SS}	T17	JIT1	V14	−
R3	PDH14/D30/INTPD14/PWM0	T18	JIT0	V15	$PLLV_{SS}$
R4	IV_{DD}	U1	PDH15/D31/INTPD15/PWM1	V16	P77/TOC3/TC3
R5	−	U2	−	V17	P74/TOC2/TC2
R6	DDO	U3	−	V18	−

備考　A1, A4, A6, A7, A10, A14, A18, B1, B4, B8, B12-B15, B17, C1, C2, C16, C18, D6, D10, D14, D15, E4, F4, F15, H1, H14, H17, J1, J4, J14, K18, L2, L15, L17, M1, M4, P1, P4, P10, P15, P18, R5, R15, R18, T3, T7, T10, T16, U2-U4, U11, U13, U14, U18, V1, V5, V8, V12-V14, V18**の端子は，オープンにしてください。**

1. 5　内部ブロック図

（1）V850E/MA1



注1.　μPD703103A　　：ROM**レス**

μPD703105A, 703106A　　：128 K**バイト（マスク**ROM）

μPD703107A　　：256 K**バイト（マスク**ROM）

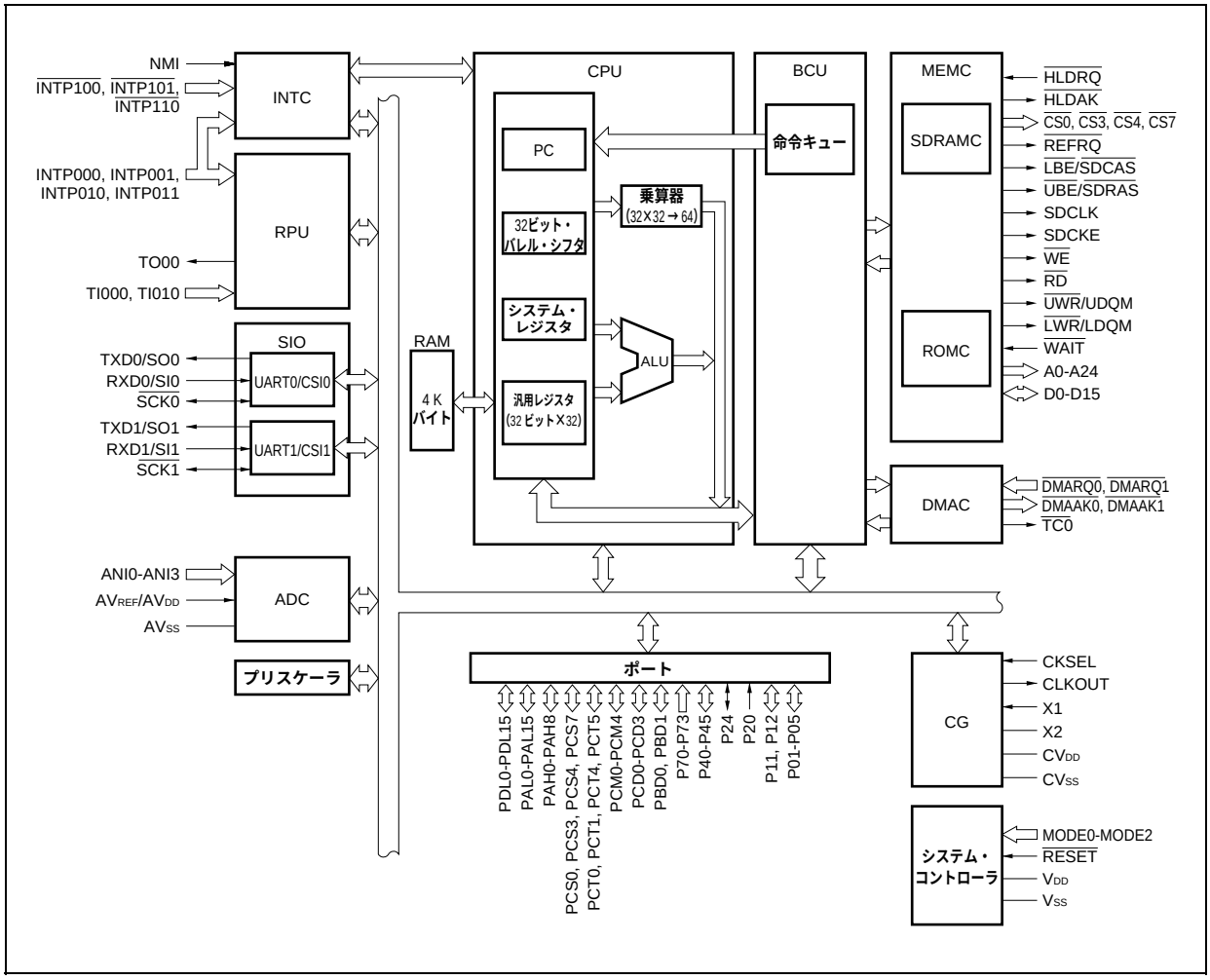
μPD70F3107A　　：256 K**バイト（フラッシュ・メモリ）**

2.　μPD703103A, 703105A　　：4 K**バイト**

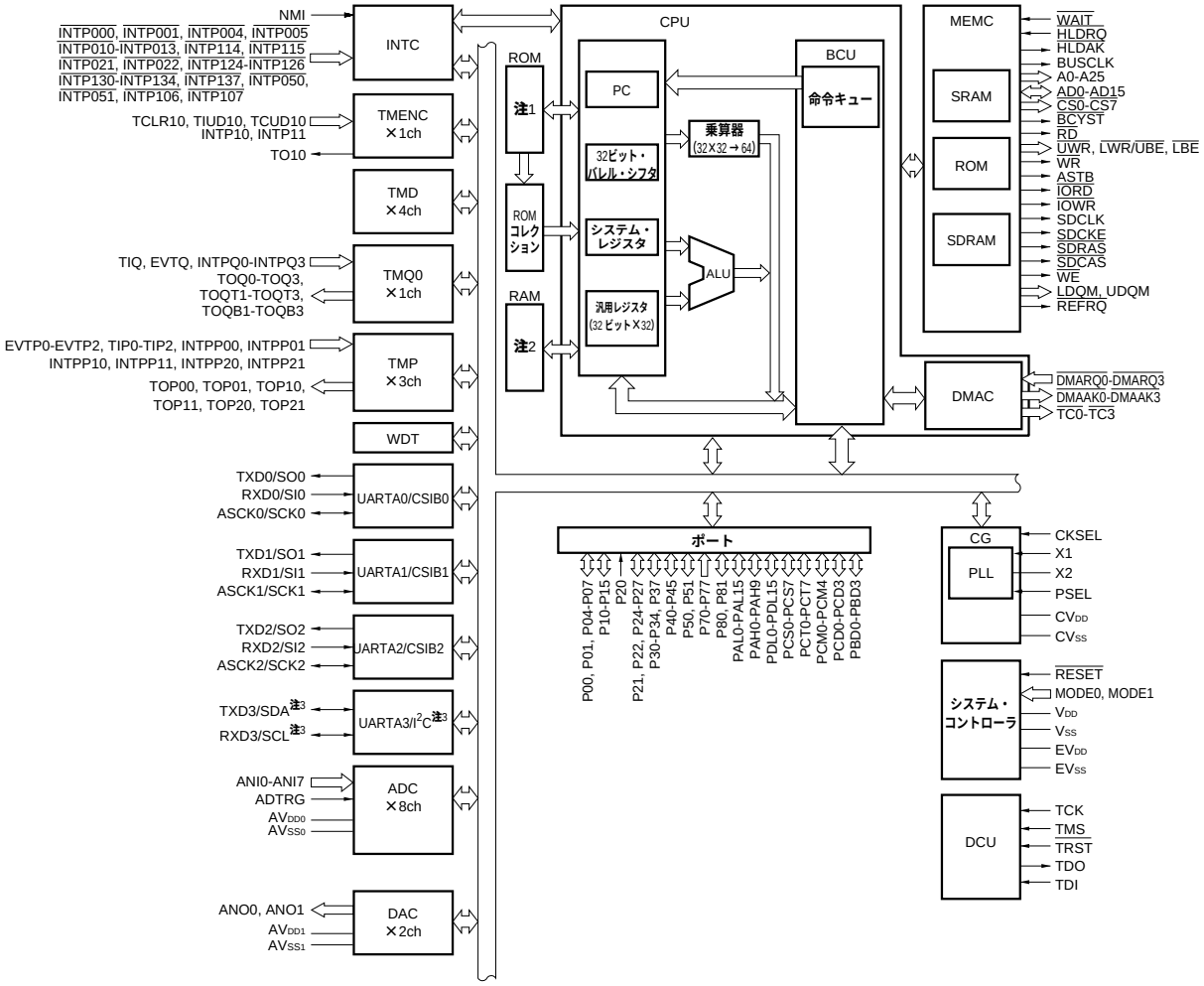
μPD703106A, 703107A, 70F3107A　：10 K**バイト**

3.　μPD70F3107A**のみ有効**

（2）V850E/MA2


NMI
INTP100, INTP101, INTP110
INTC
INTP000, INTP001, INTP010, INTP011
RPU
TO00
TI000, TI010
SIO
TXD0/SO0
RXD0/SI0
SCK0
UART0/CSI0
TXD1/SO1
RXD1/SI1
SCK1
UART1/CSI1
ANI0-ANI3
AVREF/AVDD
AVSS
ADC
プリスケーラ
RAM
4 K バイト
CPU
PC
乗算器 (32×32→64)
32ビット・バレル・シフタ
システム・レジスタ
ALU
汎用レジスタ (32 ビット×32)
BCU
命令キュー
MEMC
SDRAMC
ROMC
DMAC
HLDRQ
HLDAK
CS0, CS3, CS4, CS7
REFRQ
LBE/SDCAS
UBE/SDRAS
SDCLK
SDCKE
WE
RD
UWR/UDQM
LWR/LDQM
WAIT
A0-A24
D0-D15
DMARQ0, DMARQ1
DMAAK0, DMAAK1
TC0
ポート
PDL0-PDL15
PAL0-PAL15
PAH0-PAH8
PCS0, PCS3, PCS4, PCS7
PCT0, PCT1, PCT4, PCT5
PCM0-PCM4
PCD0-PCD3
PBD0, PBD1
P70-P73
P40-P45
P24
P20
P11, P12
P01-P05
CG
CKSEL
CLKOUT
X1
X2
CVDD
CVSS
システム・コントローラ
MODE0-MODE2
RESET
VDD
VSS

（3）V850E/MA3



注1.　μPD703131A, 703131AY, 703132A, 703132AY　：256 K**バイト（マスク**ROM）

μPD703133A, 703133AY, 703134A, 703134AY　：512 K**バイト（マスク**ROM）

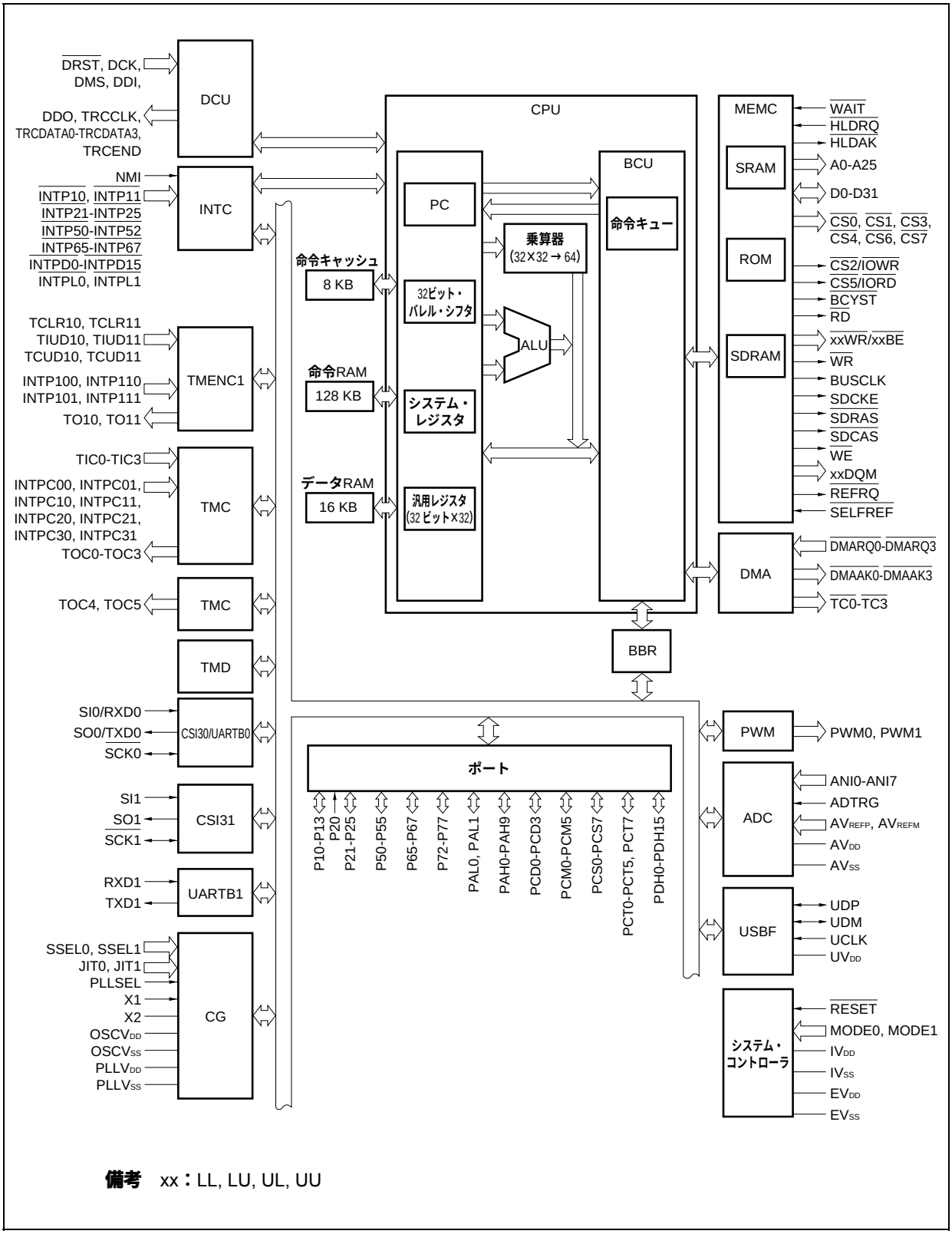
μPD70F3134A, 70F3134AY　：512 K**バイト（フラッシュ・メモリ）**

2.　μPD703131A, 703131AY, 703133A, 703133AY　：16 K**バイト**

μPD703132A, 703132AY, 703134A, 703134AY,
70F3134A, 70F3134AY　：32 K**バイト**

3.　μPD703131AY, 703132AY, 703133AY, 703134AY, 703137AY, 70F3134AY**のみ**

（4）V850E/ME2



備考　xx：LL, LU, UL, UU

第2章　PCIホスト・ブリッジ・マクロの概要

PCIホスト・ブリッジ・マクロは，V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2の外部バス・インタフェースとPCIバス・インタフェースの接続を可能にするマクロです。この章では，PCIホスト・ブリッジ・マクロの概要を説明します。

2. 1　概　　説

PCIホスト・ブリッジ・マクロは，V850E/MA1, V850E/MA2, V850E/MA3, V850E/ME2の外部バス・インタフェース（メモリ・コントローラ（MEMC））とPCIバス・インタフェースを接続するブリッジ制御マクロです。

PCIデバイスからメイン・メモリ（SDRAM）へのアクセス時に，直接SDRAMを制御できます。

2.2 特　　徴

PCIホスト・ブリッジ・マクロの特徴を次に示します。

○PCIバス・マスタ・サイクル制御

PCI Configuration Register Read/Write Single Cycle

PCI I/O Register Read/Write Single Cycle

PCI Memory Read/Write Single Cycle

○PCIバス・スレーブ・サイクル制御

PCI Memory Read/Write Cycle（最大8ダブル・ワードのバースト転送（32ビット×8バースト））

○PCIバス・アービタ制御

最大8マスタまで制御可能（そのうちの1つはPCIホスト・ブリッジ・マクロが占有）

バス・パーキング・マスタ：PCIホスト・ブリッジ・マクロ限定／最後にアクセスしたマスタから選択可能

○PCIバス・エラー処理

マスタ・アボート／ターゲット・アボート／PERR#受信／SERR#受信に対してエラー割り込みを発生

エラー発生直前のアドレスを保持

○PCIバス・アドレス変換制御

PCIバスに対してCPUからの物理アドレスを変換するためのPCI I/Oアドレス・レジスタおよびPCIメモリ・アドレス・レジスタを備えて対応

○CPUインタフェース制御

外部バス・インタフェース（MEMC）

データ・バス幅：32ビット／16ビット

ハードウエア・ウエイト制御によるサイクル制御

○SDRAM制御

PCIデバイスからのメイン・メモリ（SDRAM）アクセスに応答してSDRAMを制御

データ・バス幅：16ビット／32ビットに対応

○PCIクロック

33 MHzに対応

SDRAM制御とPCI制御のクロックは非同期設計

第3章　PCIホスト・ブリッジ・マクロの仕様

この章では，PCIホスト・ブリッジ・マクロのブロック図，信号，レジスタ仕様，動作仕様について説明します。

3. 1　PCIホスト・ブリッジ・マクロの内部ブロック

PCIホスト・ブリッジ・マクロは，図3－1　PCIホスト・ブリッジ・マクロ概略ブロック図に示すように，4つのブロックから構成されています。各ブロックの機能について，次に説明します。

(1) LM_BRIDGE：外部バス・インタフェース・マスタ制御回路

外部バス・インタフェースに接続され，CPUからのアクセスに対して応答し，PCIバス・コントロール回路のPH_FLIP_BRIDGEブロックに対してアクセス要求を出します。CPUからは，16ビット／32ビットのバス幅のアクセスが可能です。

(2) LS_BRIDGE：外部バス・インタフェース・スレーブ制御回路

PCIデバイスからのメモリ・データ転送要求に対して，PCIバス・コントロール回路のPH_FLIP_BRIDGEブロックからのアクセスに対して応答し，SDRAMCへアクセス要求を出します。

(3) SDRAMC：外部バス・インタフェースSDRAM制御回路

SDRAMバスに接続され，LS_BRIDGEブロックを通したPCIデバイスからのメモリ要求を，SDRAMバスを起動してデータ転送を行います。

SDRAMのバス幅が16ビットの場合，最大で8バーストのメモリ・サイクルを起動します。また，32ビットの場合，最大で4バーストのメモリ・サイクルを起動します。

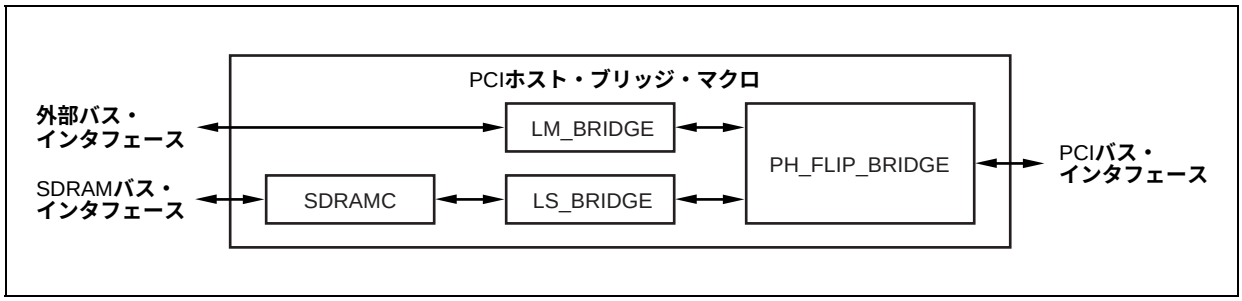
(4) PH_FLIP_BRIDGE：外部バス・インタフェース・ホスト制御回路

PCIバスに接続され，PCIホスト・デバイスとして動作します。

LM_BRIDGEブロックからの要求に対して，PCI Configuration Register Read/Write Cycle, PCI IO Register Read/Write Cycle, およびPCI Memory Read/Write Cycleを起動します。

また，PCIバスに接続されているPCIデバイスからのメモリ・データ転送要求に対して，LS_BRIDGEブロックへ要求を出します。

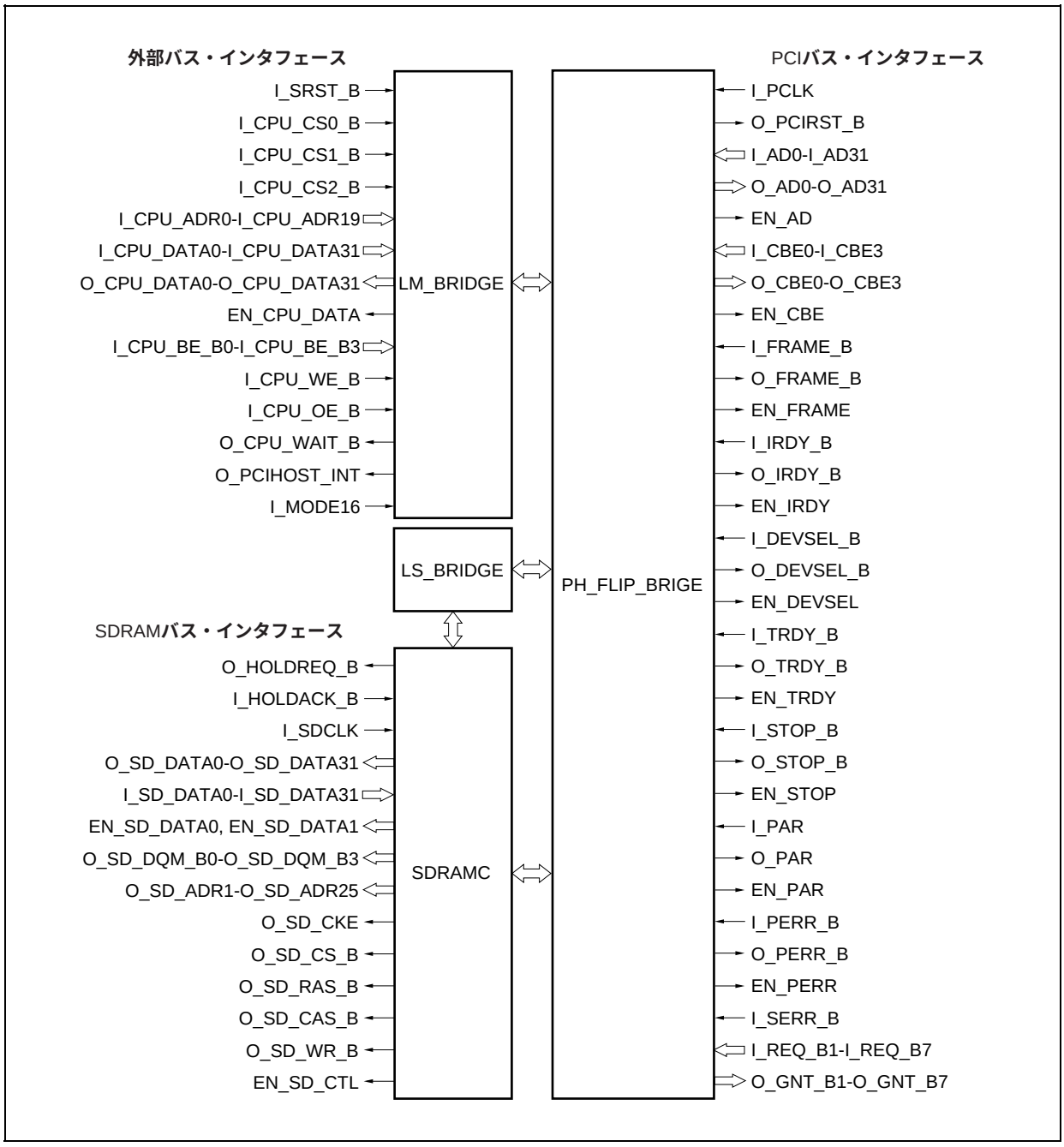
図3－1　PCIホスト・ブリッジ・マクロ概略ブロック図



3. 2　内部ブロックと信号の関係

PCIホスト・ブリッジ・マクロとしての各ブロックに対する入出力信号を，次に示します。

図3－2　PCIホスト・ブリッジ・マクロのブロックと端子信号



3. 3　端子機能

各インタフェースにおける端子機能について，次に説明します。

3. 3. 1　外部バス・スレーブ・インタフェース端子

端子名称	入出力	機　　能	アクティブ
I_SRST_B	入力	システム・リセット入力	ロウ
I_CPU_CS0_B	入力	PCIホスト・ブリッジ・レジスタ・チップ・セレクト入力	ロウ
I_CPU_CS1_B	入力	PCI I/O領域チップ・セレクト入力	ロウ
I_CPU_CS2_B	入力	PCIメモリ領域チップ・セレクト入力	ロウ
I_CPU_ADR0-I_CPU_ADR19	入力	CPUアドレス入力	－
I_CPU_DATA0-I_CPU_DATA31	入力	CPUデータ入力	－
O_CPU_DATA0-O_CPU_DATA31	出力	CPUデータ出力	－
EN_CPU_DATA	出力	CPUデータ・アウトプット・イネーブル出力	ハイ
I_CPU_BE_B0-I_CPU_BE_B3	入力	CPUデータ・バイト・イネーブル入力	－
I_CPU_WE_B	入力	CPUライト・データ・イネーブル入力	ロウ
I_CPU_OE_B	入力	CPUリード・データ・アウトプット・イネーブル入力	ロウ
O_CPU_WAIT_B	出力	CPUデータ・ウエイト出力	ロウ
O_PCIHOST_INT	出力	PCIホスト・ブリッジ割り込み出力	ロウ
I_MODE16	入力	CPUデータ・バス幅セレクト入力	ロウ：32ビット幅 ハイ：16ビット幅

3. 3. 2　SDRAMバス・インタフェース端子

端子名称	入出力	機　　能	アクティブ
O_HOLDREQ_B	出力	SDRAMバス・ホールド・リクエスト出力	ロウ
I_HOLDACK_B	入力	SDRAMバス・ホールド・アクノリッジ入力	ロウ
I_SDCLK	入力	SDRAMクロック入力	－
O_SD_DATA0-O_SD_DATA31	出力	SDRAMデータ出力	－
I_SD_DATA0-I_SD_DATA31	入力	SDRAMデータ入力	－
EN_SD_DATA0, EN_SD_DATA1	出力	SDRAMデータ・イネーブル出力 ロウ：下位16ビット（O_SD_DATA0-O_SD_DATA15） ハイ：上位16ビット（O_SD_DATA16-O_SD_DATA31）	←
O_SD_DQM_B0-O_SD_DQM_B3	出力	SDRAMデータ・マスク出力	ロウ
O_SD_ADR1-O_SD_ADR25	出力	SDRAMアドレス出力	－
O_SD_CKE	出力	SDRAMクロック・イネーブル出力	ハイ
O_SD_CS_B	出力	SDRAMチップ・セレクト出力	ロウ
O_SD_RAS_B	出力	SDRAMロウ・アドレス・ストローブ出力	ロウ
O_SD_CAS_B	出力	SDRAMカラム・アドレス・ストローブ出力	ロウ
O_SD_WR_B	出力	SDRAMリード／ライト出力	ロウ
EN_SD_CTL	出力	SDRAM制御信号アウトプット・イネーブル出力 （O_SD_ADR1-O_SD_ADR25, O_SD_CKE, O_SD_CS_B, O_SD_RAS_B, O_SD_CAS_B, O_SD_WR_Bの各端子出力バッファ・イネーブル）	ハイ

3. 3. 3　PCIバス・インタフェース端子

端子名称	入出力	機　　能	アクティブ
I_PCLK	入力	PCIクロック入力	−
O_PCIRST_B	出力	PCIリセット出力	ロウ
I_AD0-I_AD31	入力	PCIアドレス／データ入力	−
O_AD0-O_AD31	出力	PCIアドレス／データ出力	−
EN_AD	出力	PCIアドレス／データ・アウトプット・イネーブル出力 （O_AD0-O_AD31の出力バッファ・イネーブル）	ハイ
I_CBE0-I_CBE3	入力	PCIコマンド／バイト・イネーブル入力	ロウ
O_CBE0-O_CBE3	出力	PCIコマンド／バイト・イネーブル出力	ロウ
EN_CBE	出力	PCIコマンド／バイト・イネーブル・アウトプット・イネーブル出力 （O_CBE0-O_CBE3の出力バッファ・イネーブル）	ハイ
I_FRAME_B	入力	PCIフレーム入力	ロウ
O_FRAME_B	出力	PCIフレーム出力	ロウ
EN_FRAME	出力	PCIフレーム・アウトプット・イネーブル出力 （O_FRAME_Bの出力バッファ・イネーブル）	ハイ
I_IRDY_B	入力	PCIイニシエータ・レディ入力	ロウ
O_IRDY_B	出力	PCIイニシエータ・レディ出力	ロウ
EN_IRDY	出力	PCIイニシエータ・レディ・アウトプット・イネーブル出力 （O_IRDY_Bの出力バッファ・イネーブル）	ハイ
I_DEVSEL_B	入力	PCIデバイス・セレクト入力	ロウ
O_DEVSEL_B	出力	PCIデバイス・セレクト出力	ロウ
EN_DEVSEL	出力	PCIデバイス・セレクト・アウトプット・イネーブル出力 （O_DEVSEL_Bの出力バッファ・イネーブル）	ハイ
I_TRDY_B	入力	PCIターゲット・レディ入力	ロウ
O_TRDY_B	出力	PCIターゲット・レディ出力	ロウ
EN_TRDY	出力	PCIターゲット・レディ・アウトプット・イネーブル出力 （O_TRDY_Bの出力バッファ・イネーブル）	ハイ
I_STOP_B	入力	PCIストップ入力	ロウ
O_STOP_B	出力	PCIストップ出力	ロウ
EN_STOP	出力	PCIストップ・アウトプット・イネーブル出力 （O_STOP_Bの出力バッファ・イネーブル）	ハイ
I_PAR	入力	PCIパリティ入力	−
O_PAR	出力	PCIパリティ出力	−
EN_PAR	出力	PCIパリティ・アウトプット・イネーブル出力 （O_PARの出力バッファ・イネーブル）	ハイ
I_PERR_B	入力	PCIパリティ・エラー入力	ロウ
O_PERR_B	出力	PCIパリティ・エラー出力	ロウ
EN_PERR	出力	PCIパリティ・エラー・アウトプット・イネーブル出力 （O_PERR_Bの出力バッファ・イネーブル）	ハイ
I_SERR_B	入力	PCIシステム・エラー入力	ロウ
I_REQ_B1-I_REQ_B7	入力	PCIリクエスト入力	ロウ
O_GNT_B1-O_GNT_B7	出力	PCIグラント出力	ロウ

3. 4　レジスタ

PCIホスト・ブリッジ・マクロのレジスター覧を次に示します。レジスタのビット幅はすべて32ビットです。

各レジスタのオフセット・アドレスは，I_CPU_CS0_B端子がアクティブとなる領域のベース・アドレスからのオフセット値です。

オフセット・アドレス	レジスタ名	R/W	機　能
00H	PCI_CONFIG_DATA	R/W	PCI Configuration Registerアクセス・データ設定
04H	PCI_CONFIG_ADD	R/W	PCI Configuration Registerアクセス・アドレス設定
08H	PCI_CONTROL	R/W	PCIバス制御
0CH	予約		
10H	PCI_IO_BASE	R/W	CPUメモリ・マップ上のPCI I/O領域からアクセスするPCIバスI/O空間のベース・アドレスを設定
14H	PCI_MEM_BASE	R/W	CPUメモリ・マップ上のPCIメモリ領域からアクセスするPCIバス・メモリ空間のベース・アドレスを設定
18H	PCI_INT_CTL	R/W	PCIエラー割り込み制御
1CH	PCI_ERR_ADD	R	PCIエラー発生アドレス保持
20H-3FH	予約		
40H	SYSTEM_MEM_BASE	R/W	PCIバス・メモリ空間にマッピングするシステム・メモリ領域のベース・アドレスを設定
44H	SYSTEM_MEM_RANGE	R/W	PCIバス・メモリ空間にマッピングするシステム・メモリ領域の範囲を設定
48H	SDRAM_CTL	R/W	SDRAMアクセス制御
4CH-FFH	予約		

3. 4. 1　PCI_CONFIG_DATAレジスタ

リセット時：不定　R/W　オフセット・アドレス：00H

31	0
CDATA	

ビット名	R/W	機　能
CDATA	R/W	このフィールドにデータをライトすることにより，PCI Configuration Register Writeアクセスを実行し，このフィールドにライトしたデータをアクセス対象レジスタにライトします。 このフィールドをリードすることにより，PCI Configuration Register Readアクセスを実行し，アクセス対象レジスタのデータをリードします。

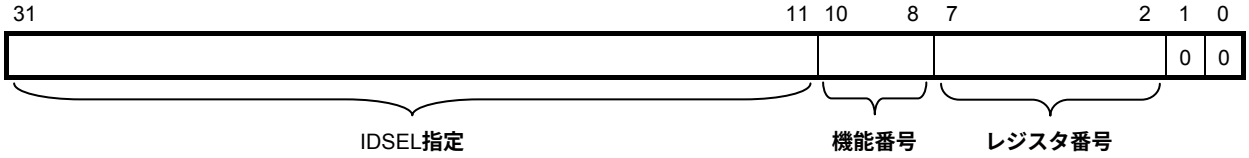
3. 4. 2　PCI_CONFIG_ADDレジスタ

リセット時：00000000H　R/W　**オフセット・アドレス：**04H

ビット名	R/W	機　能
CADD	R/W	アクセス対象のPCI Configuration Registerアドレスを設定します。

(1) PCI_CONFIG_ADDレジスタの設定方法

(a) タイプ0 (PCIデバイス)

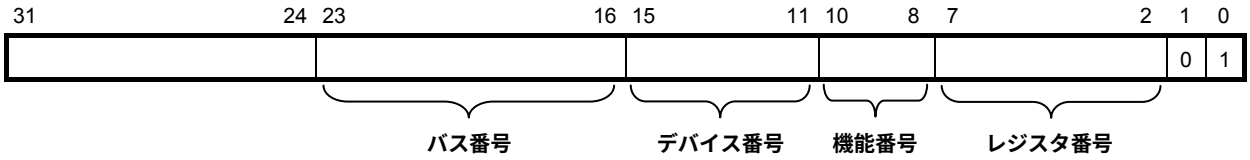


IDSEL指定　：アクセス対象PCIデバイスに対応するIDSEL信号を選択します。
このPCIホスト・ブリッジ・マクロは，各PCIデバイスに対するIDSEL信号をAD31-AD11信号で代用するため，各PCIデバイスのIDSEL端子に接続しているAD信号をこのフィールドで指定します。たとえば，あるPCIデバイスのIDSEL端子にAD31信号が接続されている場合，CADDのビット31に1を設定することにより，アクセスが可能となります。

機能番号　：マルチファンクション・デバイスに対する機能番号を指定します。

レジスタ番号　：アクセス対象PCI Configuration Registerの番号を指定します。

(b) タイプ1 (PCI-PCIブリッジ)



バス番号　：アクセス対象PCIデバイスが接続されているPCIバスの番号を指定します。

デバイス番号　：アクセス対象PCIデバイスのデバイス番号を指定します。

機能番号　：マルチファンクション・デバイスに対する機能番号を指定します。

レジスタ番号　：アクセス対象PCI Configuration Registerの番号を指定します。

（2）PCI Configuration Registerへのアクセス方法

・ライト・アクセス

PCI_CONFIG_ADDレジスタにアクセス対象レジスタのアドレスを設定

↓

PCI_CONFIG_DATAレジスタにアクセス対象レジスタへの設定値をライト

・リード・アクセス

PCI_CONFIG_ADDレジスタにアクセス対象レジスタのアドレスを設定

↓

PCI_CONFIG_DATAレジスタをリード

3. 4. 3　PCI_CONTROLレジスタ

リセット時：07000100H　R/W　オフセット・アドレス：08H

31 24	23						17	16	15 8	7		5	4	3	2	1	0
PCI_PARKCNT	0	0	0	0	0	0	0	PCI_BPMODE	PCI_REQ	0	0	0	PCI_RESET	0	TARGET_EN	MEM_EN	IO_EN

ビット名	R/W	機　能
PCI_PARKCNT	R/W	バス・パーキングに移行する時間を設定します。 ディフォールト値では，バスがIDLE状態になってから7クロック後にバス・パーキングを行います。 カウンタ開始はFRAME# = High & IRDY# = Highです。
PCI_BPMODE	R/W	バス・パーキング・マスタを設定します。 0：このマクロ限定 1：最後にアクセスしたマスタ
PCI_REQ	R/W	バス・マスタからのREQ#信号（I_REQ_B1-I_REQ_B7端子）の有効／無効を設定します。 このフィールドのビット0（PCI_CONTROLレジスタのビット8）はこのPCIホスト・ブリッジ・マクロに割り当てられており，常に1となります。 0：無効 1：有効
PCI_RESET	R/W	PCIバスのリセット状態を設定します。 0：リセット状態 1：リセット解除
TARGET_EN	R/W	このPCIホスト・ブリッジ・マクロのPCIバス・ターゲットとしての動作を設定します。 0：PCIデバイスからのメイン・メモリ（SDRAM）アクセスに応答しない 1：PCIデバイスからのメイン・メモリ（SDRAM）アクセスに応答する
MEM_EN	R/W	CPUからPCIメモリ領域へのアクセスの許可／禁止を設定します。 0：アクセス禁止 1：アクセス許可
IO_EN	R/W	CPUからPCI I/O領域へのアクセスの許可／禁止を設定します。 0：アクセス禁止 1：アクセス許可

3. 4. 4　PCI_IO_BASEレジスタ

PCI I/O領域（I_CPU_CS1_B端子がアクティブとなる領域：64 Kバイト）を介してPCIバスI/O空間へのI/Oアクセスを行うとき，このレジスタへの設定によって，4 GバイトのPCIバスI/O空間のうち任意の領域にアクセスできます。

リセット時：00000000H　R/W　**オフセット・アドレス：**10H

31	16	15														0
IO_BASE		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ビット名	R/W	機　能
IO_BASE	R/W	CPUからPCI I/O領域（I_CPU_CS1_B端子がアクティブとなる領域）にアクセスするときの，PCIバスI/O空間ベース・アドレスの上位16ビット（ビット16-31）を設定します。

3. 4. 5　PCI_MEM_BASEレジスタ

PCIメモリ領域（I_CPU_CS2_B端子がアクティブとなる領域：1 Mバイト）を介してPCIバス・メモリ空間へのメモリ・アクセスを行うとき，このレジスタへの設定によって，4 GバイトのPCIバス・メモリ空間のうち任意の領域にアクセスできます。

ただし，PCIバス・メモリ空間上にメイン・メモリ（SDRAM）がマッピングされているので，後述のSYSTEM_MEM_BASEレジスタおよびSYSTEM_MEM_RANGEレジスタで設定される領域と重複しないように注意してください。

リセット時：80000000H　R/W　**オフセット・アドレス：**14H

31	20	19																		0
M_BASE		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ビット名	R/W	機　能
M_BASE	R/W	CPUからPCIメモリ領域（I_CPU_CS2_B端子がアクティブとなる領域）にアクセスするときの，PCIバス・メモリ空間ベース・アドレスの上位12ビット（ビット20-31）を設定します。

3. 4. 6　PCI_INT_CTLレジスタ

PCI_INT_CTLレジスタは，PCIバス・エラー割り込み（O_PCIHOST_INT）の割り込み要因を示し，それらに対するマスク制御およびクリア制御を行います。

この機能は，ディバグ時だけ使用されます。通常時には使用しません。

リセット時：000x0F00H　R/W　オフセット・アドレス：18H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	CLR_SERR	CLR_PERR	CLR_MAB	CLR_TAB	0	0	0	0	MSK_SERR	MSK_PERR	MSK_MAB	MSK_TAB	0	0	0	0	SERR	PERR	MABORT	TABORT

ビット名	R/W	機　能
CLR_SERR	W	PCIバス・システム・エラー（SERR#受信）割り込みをクリアします。 1：クリア
CLR_PERR	W	PCIバス・パリティ・エラー（PERR#受信）割り込みをクリアします。 1：クリア
CLR_MAB	W	PCIバス・マスタ・アボート割り込みをクリアします。 1：クリア
CLR_TAB	W	PCIバス・ターゲット・アボート割り込みをクリアします。 1：クリア
MSK_SERR	R/W	PCIバス・システム・エラー（SERR#受信）割り込みのマスク状態を設定します。 0：マスクしない 1：マスクする
MSK_PERR	R/W	PCIバス・パリティ・エラー（PERR#受信）割り込みのマスク状態を設定します。 0：マスクしない 1：マスクする
MSK_MAB	R/W	PCIバス・マスタ・アボート割り込みのマスク状態を設定します。 0：マスクしない 1：マスクする
MSK_TAB	R/W	PCIバス・ターゲット・アボート割り込みのマスク状態を設定します。 0：マスクしない 1：マスクする
SERR	R	PCIバス・システム・エラー（SERR#受信）の発生状態を検出します。 1：システム・エラー発生
PERR	R	PCIバス・パリティ・エラー（PERR#受信）の発生状態を検出します。 1：パリティ・エラー発生
MABORT	R	PCIバス・マスタ・アボートの発生状態を検出します。 1：マスタ・アボート発生
TABORT	R	PCIバス・ターゲット・アボートの発生状態を検出します。 1：ターゲット・アボート発生

3. 4. 7　PCI_ERR_ADDレジスタ

PCI_ERR_ADDレジスタは，次に示したエラー要因が発生したときのPCIバス・アドレスを保持します。

- システム・エラー（SERR#受信）
- パリティ・エラー（PERR#受信）
- マスタ・アボート
- ターゲット・アボート

PCI_ERR_ADDレジスタをリードすると，すべてのビットがクリアされます。一度エラーが発生し，PCI_ERR_ADDレジスタに値が設定されると，リード・アクセスを受けるか，新たなエラーが発生して値が更新されるまでは，最初の値を保持します。

この機能は，ディバグ時だけ使用されます。通常時には使用しません。

リセット時：00000000H　R　**オフセット・アドレス：**1CH

31	0
ERR_ADR	

ビット名	R/W	機　能
ERR_ADR	R	PCIバス・エラー発生時のアドレスを保持します。

3. 4. 8　SYSTEM_MEM_BASEレジスタ

SYSTEM_MEM_BASEレジスタとSYSTEM_MEM_RANGEレジスタの設定によって，PCIデバイスからのメイン・メモリ・アクセスが発生した場合，一致したアドレスへのアクセスに対して応答します。

リセット時：00000000H　R/W　**オフセット・アドレス：**40H

31	16	15															0
S_BASE		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ビット名	R/W	機　能
S_BASE	R/W	メイン・メモリ（SDRAM）をマッピングするPCIバス・メモリ空間上ベース・アドレスの上位16ビット（ビット16-31）を設定します。

3. 4. 9　SYSTEM_MEM_RANGEレジスタ

リセット時：0000FFFFH　R/W　**オフセット・アドレス：**44H

31	16	15															0
S_RANGE		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

ビット名	R/W	機　能
S_RANGE	R/W	メイン・メモリ（SDRAM）をマッピングするPCIバス・メモリ空間の範囲を設定します。 64Kバイト単位で設定できます。 0000H：64 Kバイト 0001H：128 Kバイト ： 000FH：1 Mバイト ： 00FFH：16 Mバイト ： 0FFFH：256 Mバイト ： FFFFH：4 Gバイト

3. 4. 10　SDRAM_CTLレジスタ

リセット時：00070230H　R/W　**オフセット・アドレス：**48H

31	30	29	28	27	26	25	24	23 ～ 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	CYCLE_LATENCY	0	0	0	BUS_SIZE	0	0	CAS_LATENCY		0	0	WAIT_STATE		0	0	COLUMN_SIZE	

ビット名	R/W	機　　能
CYCLE_LATENCY	R/W	PCIデバイスからの連続したメイン・メモリ（SDRAM）アクセスに対するレーテンシを設定します。 最大7,650 nsのレーテンシが設定可能です。 00H：レーテンシなし 01H：1 PCIクロック（30 ns） ： FFH：255 PCIクロック（7,650 ns）
BUS_SIZE	R/W	データ・バスのビット幅を設定します。 0：16ビット幅 1：32ビット幅
CAS_LATENCY	R/W	CASレーテンシを設定します。 00：設定禁止 01：1 10：2 11：3
WAIT_STATE	R/W	ACT→CMD, PRE→ACT, CMD→ACTのウエイト間隔を設定します。 00：設定禁止 01：1クロック 10：2クロック 11：3クロック
COLUMN_SIZE	R/W	カラム・アドレスのビット幅を設定します。 00：8ビット幅 01：9ビット幅 10：10ビット幅 11：11ビット幅

メイン・メモリ（SDRAM）へアクセスするときに出力するアドレス信号とPCIバス・アドレス信号との対応を，次に示します。

表3－1　ロウ・アドレス出力時

COLUMN_SIZE	メイン・メモリ（SDRAM）アドレス端子（O_SD_ADR1-O_SD_ADR25）に対するPCIバス・アドレス信号対応																	
フィールド設定値	25-18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
00（8ビット）	25-18	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
01（9ビット）	25-18	17	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
10（10ビット）	25-18	17	16	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
11（11ビット）	25-18	17	16	15	25	24	23	22	21	20	19	18	17	16	15	14	13	12

表3－2　カラム・アドレス出力時（プリチャージ・コマンド）

BUS_SIZE	メイン・メモリ（SDRAM）アドレス端子（O_SD_ADR1-O_SD_ADR25）に対するPCIバス・アドレス信号対応																	
ビット設定値	25-18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0（16ビット）	25-18	17	16	15	14	12	11	H	10	9	8	7	6	5	4	3	2	1
1（32ビット）	25-18	17	16	15	14	12	H	11	10	9	8	7	6	5	4	3	2	1

備考　H：ハイ・レベル

表3－3　カラム・アドレス出力時（リード／ライト・コマンド）

BUS_SIZE	メイン・メモリ（SDRAM）アドレス端子（O_SD_ADR1-O_SD_ADR25）に対するPCIバス・アドレス信号対応																	
ビット設定値	25-18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0（16ビット）	25-18	17	16	15	14	12	11	L	10	9	8	7	6	5	4	3	2	1
1（32ビット）	25-18	17	16	15	14	12	L	11	10	9	8	7	6	5	4	3	2	1

備考　L：ロウ・レベル

3. 5　アドレス・マップ

CPUメモリ空間およびPCIバスI/O／メモリ空間のアドレス・マップを，次に示します。

図3－3　CPUメモリ空間／PCIバスI/O空間アドレス・マップ

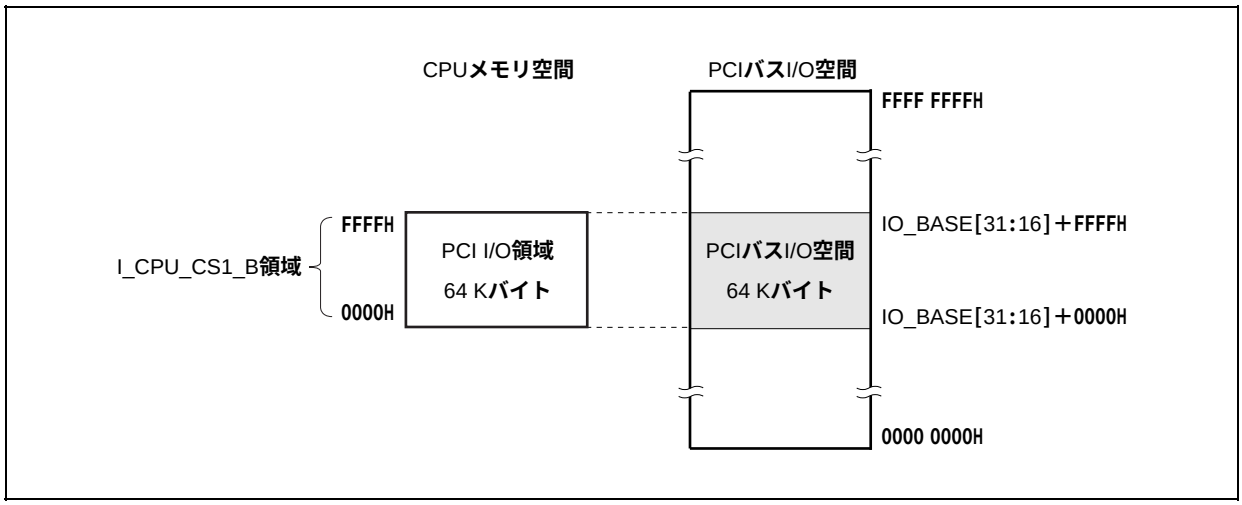
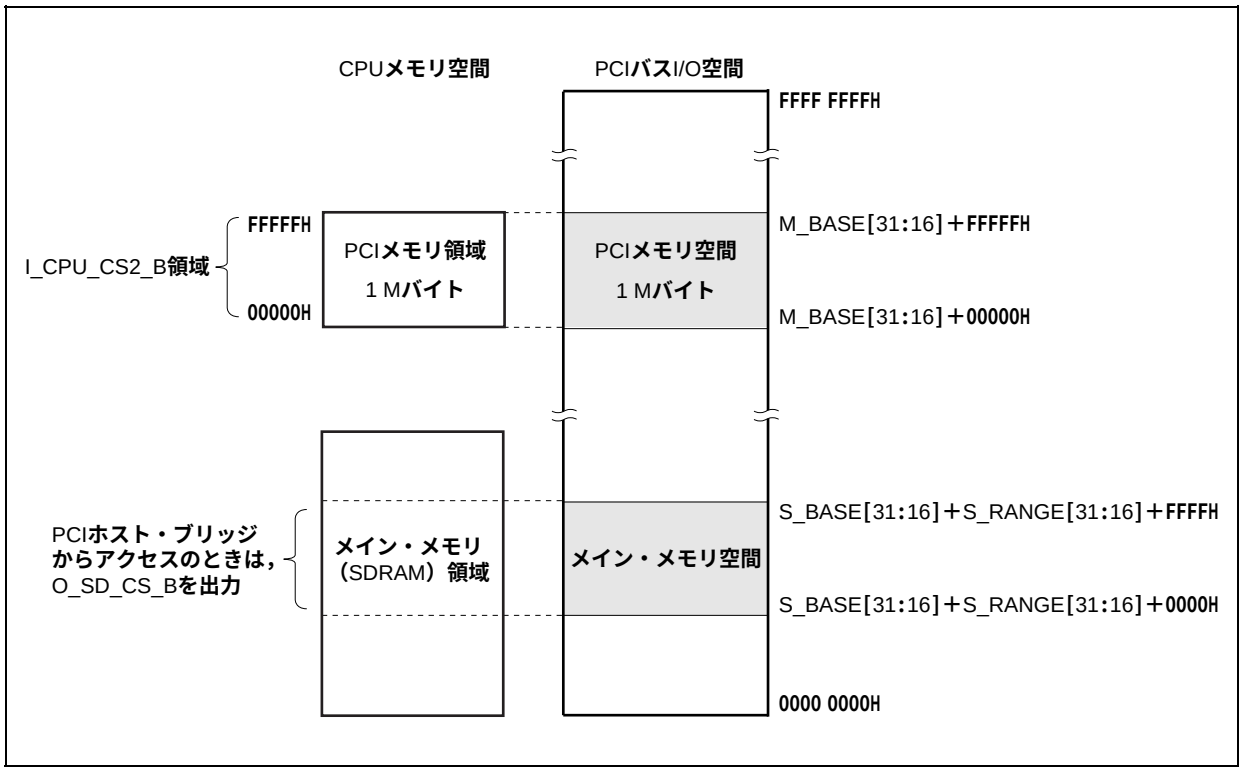


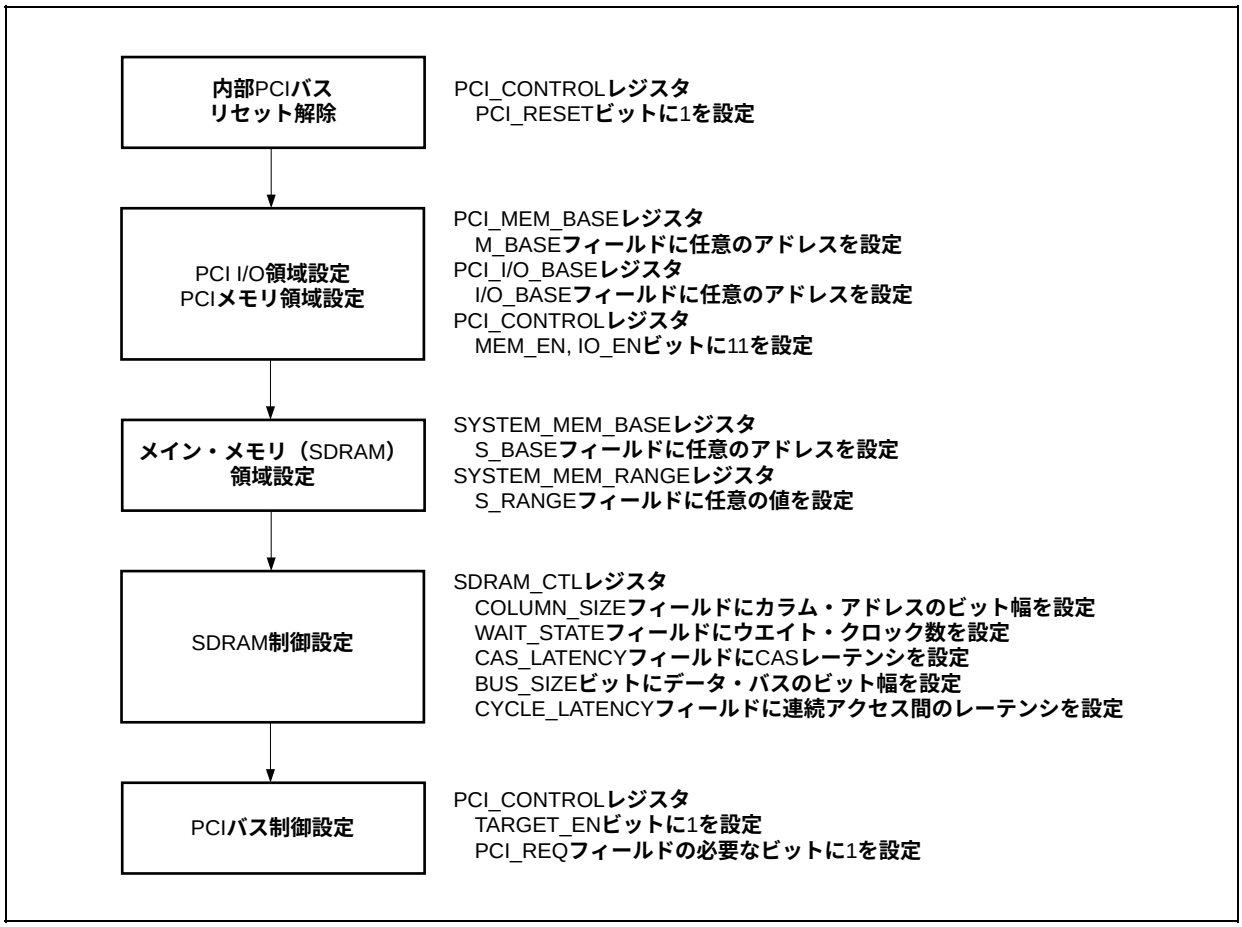
図3－4　CPUメモリ空間／PCIバス・メモリ空間アドレス・マップ



3. 6　PCIホスト・ブリッジ・マクロ初期化方法

PCIバスへのメモリ・アクセスおよびI/Oアクセス，PCIデバイスからのメイン・メモリ（SDRAM）アクセスを受け付けるためには，次に示す手順に従って，このPCIホスト・ブリッジ・マクロを初期化する必要があります。

図3－5　PCIホスト・ブリッジ・マクロ初期化方法



3. 7　外部バス・インタフェースのバス幅について

外部バス・インタフェースに対して，I_MODE16端子の状態で，データ・バス幅の動作モードを変更できます。

注意1. **動作中に**I_MODE16**端子の状態を変更しないでください。**

2. I_MODE16**端子でデータ・バス幅の動作モードを変更できるのは，外部バス・スレーブ・インタフェースだけです。**
   SDRAM**バス・インタフェースのデータ・バス幅を変更するには，**3. 4. 10　SDRAM_CTL**レジスタの**BUS_SIZE**ビットで設定してください。**
3. I_MODE16**端子の設定は，**CPU**の外部バス・インタフェース動作モードと一致させてください。**
4. 16**ビット・モードに設定した場合，外部バス・インタフェース上で，**32**ビット・アクセスはアクセス・サイクルが分割されます。したがって，**PCI**バス・インタフェース上でも同様にアクセスが分割されます。このため，**16**ビット・モードに設定した場合，**PCI**バス・インタフェース上では**32**ビット・アクセス・サイクルは発生しませんので，**32**ビット・アクセスだけが有効なレジスタを持つ**PCI**デバイスに対しては，アクセスできません。**

表3－4　I_MODE16**端子の状態とデータ・バス幅の動作モード**

I_MODE16端子	データ・バス動作モード	備　　考
ロウ・レベル	32ビット・モード	32ビット・データ・バス
ハイ・レベル	16ビット・モード	16ビット・データ・バス

3. 8　タイミング

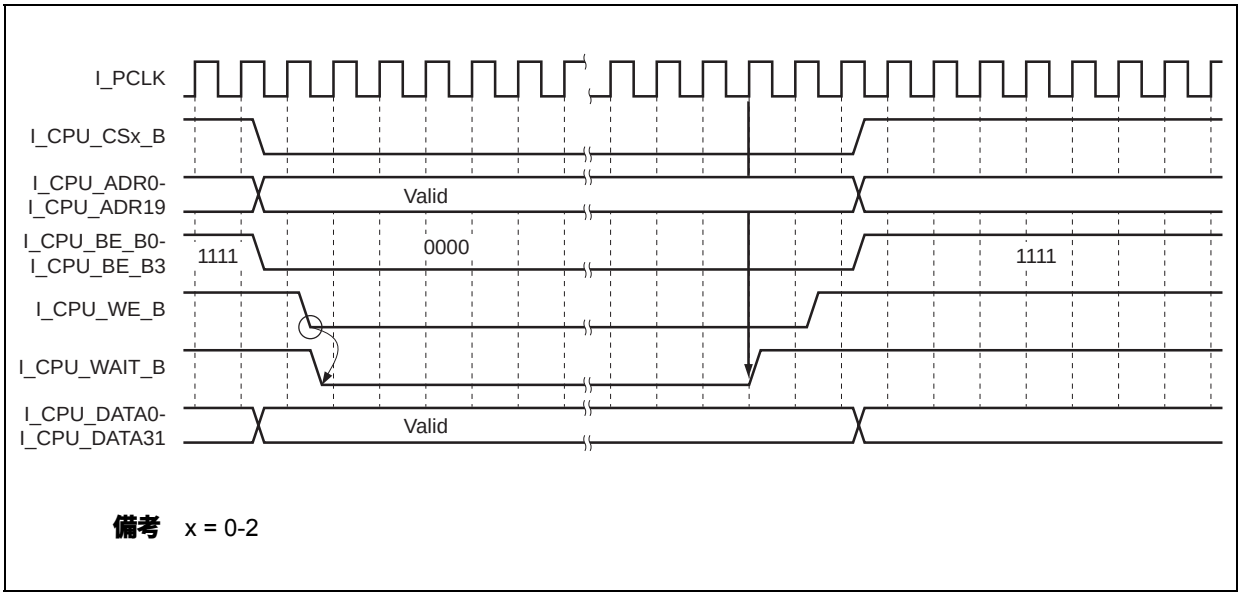
PCIホスト・ブリッジ・マクロの各インタフェースに対するタイミングを，次に示します。

3. 8. 1　外部バス・インタフェース・タイミング

CPUからは，バス・インタフェースにより，CPUライト／CPUリードでアクセスされます（図3－6，図3－7）。

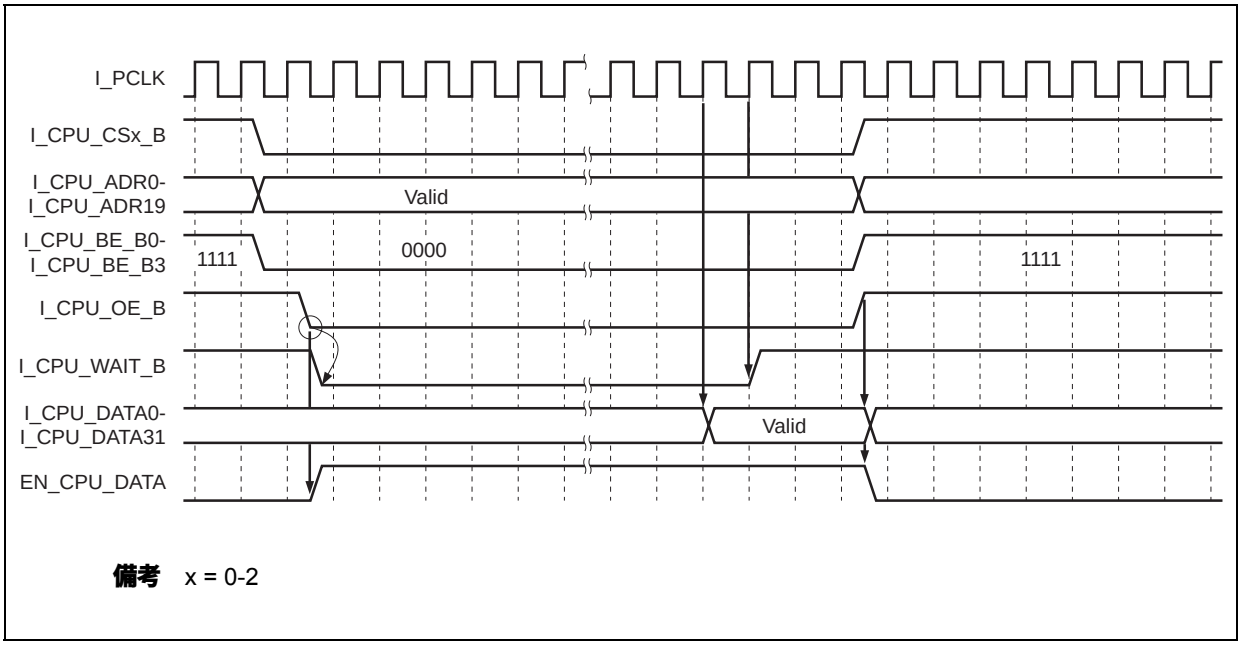
PCIホスト・ブリッジ・マクロからSDRAMにアクセスするときは，バスのホールドを行い，メイン・メモリへのライト／リードでアクセスを行います（図3－8～図3－10）。

図3－6　CPUライト・アクセス



備考　x = 0-2

図3－7　CPUリード・アクセス



備考　x = 0-2

図3－8　ホールド・リクエスト／ホールド・アクノリッジ

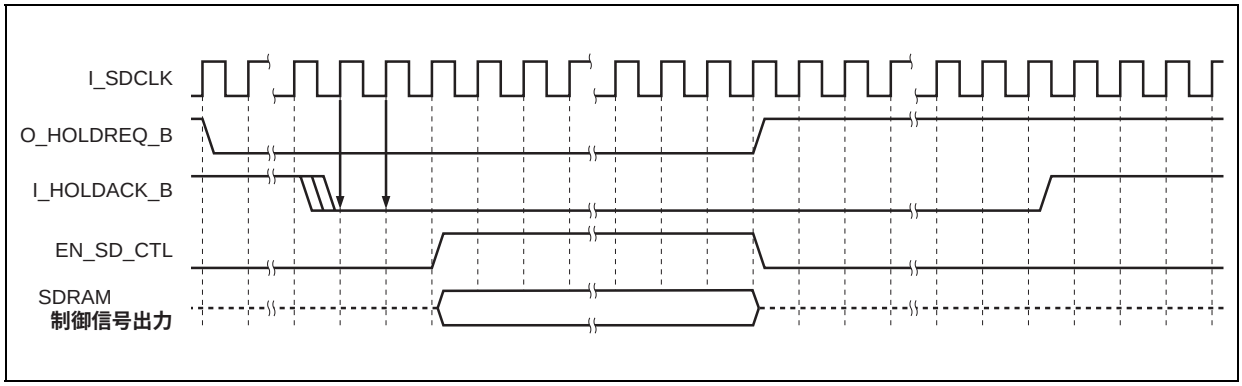
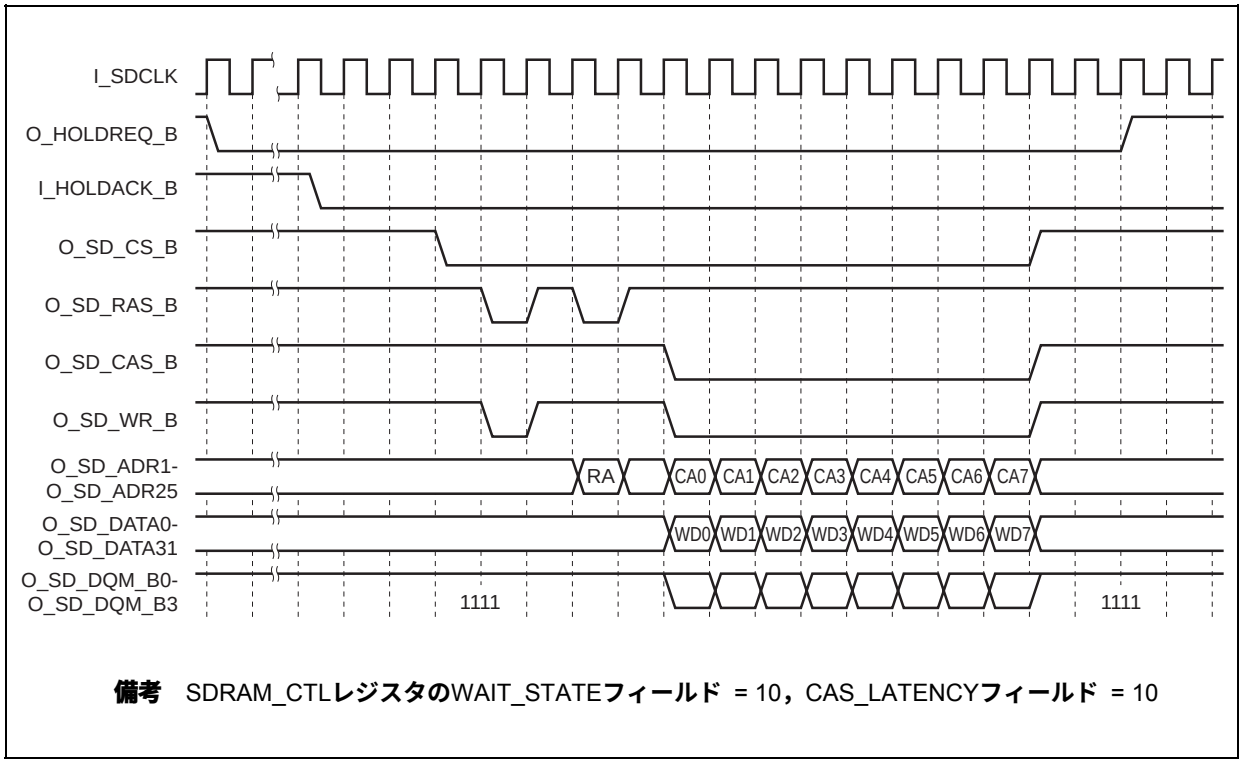
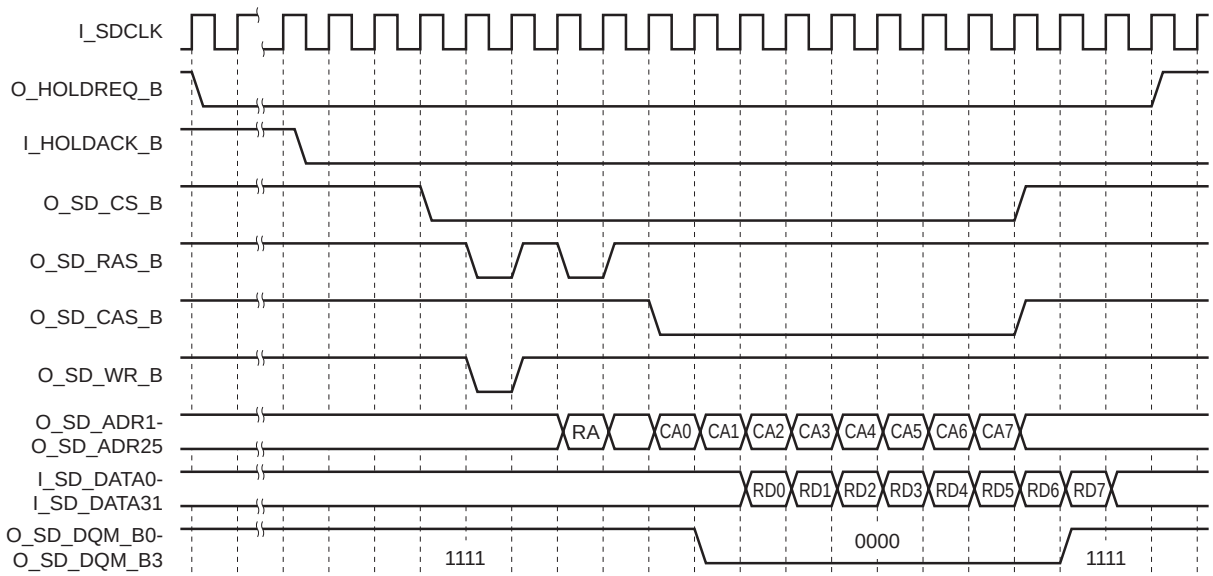


図3－9　メイン・メモリ（SDRAM）ライト・アクセス（8バースト）



備考　SDRAM_CTL**レジスタの**WAIT_STATE**フィールド** = 10，CAS_LATENCY**フィールド** = 10

図3－10　**メイン・メモリ（**SDRAM**）リード・アクセス（**8**バースト）**



備考　SDRAM_CTL**レジスタの**WAIT_STATE**フィールド** = 10，CAS_LATENCY**フィールド** = 10

3. 8. 2　PCIバス・インタフェース・タイミング

PCIホスト・ブリッジ・マクロでは，次に示すPCIバス・インタフェース・タイミングに対応しています。

(1) PCIバス・マスタ・サイクル・タイミング

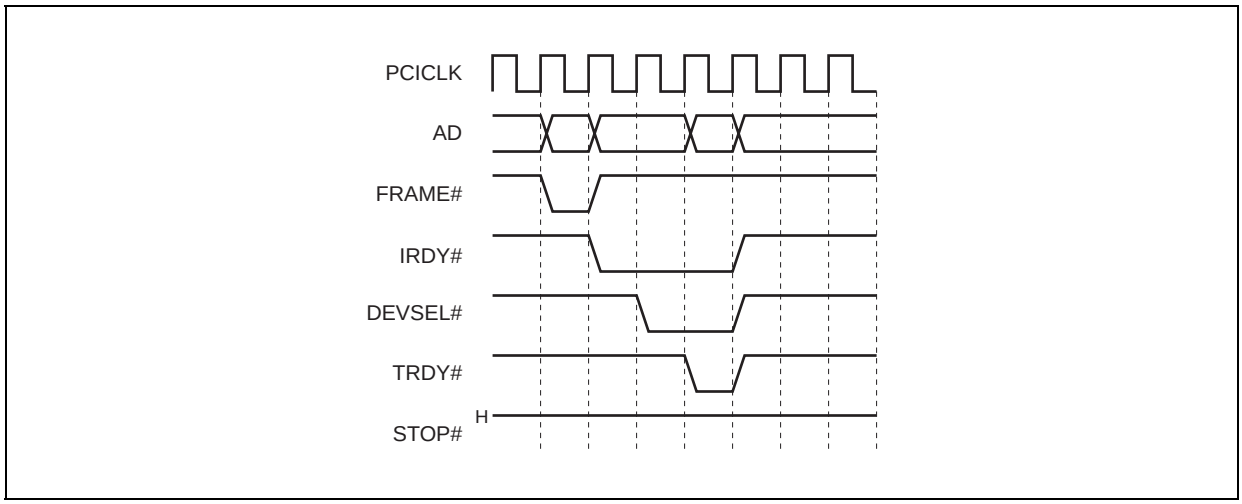
CPUからPCIデバイスへのアクセス・タイミングを示します。

(a) Configuration Read/Write Cycle & I/O Read/Write Cycle & Memory Read/Write Cycle

(i) Read Cycle

Timing Type：Configuration Register Read, Internal I/O Register Read, Memory Read

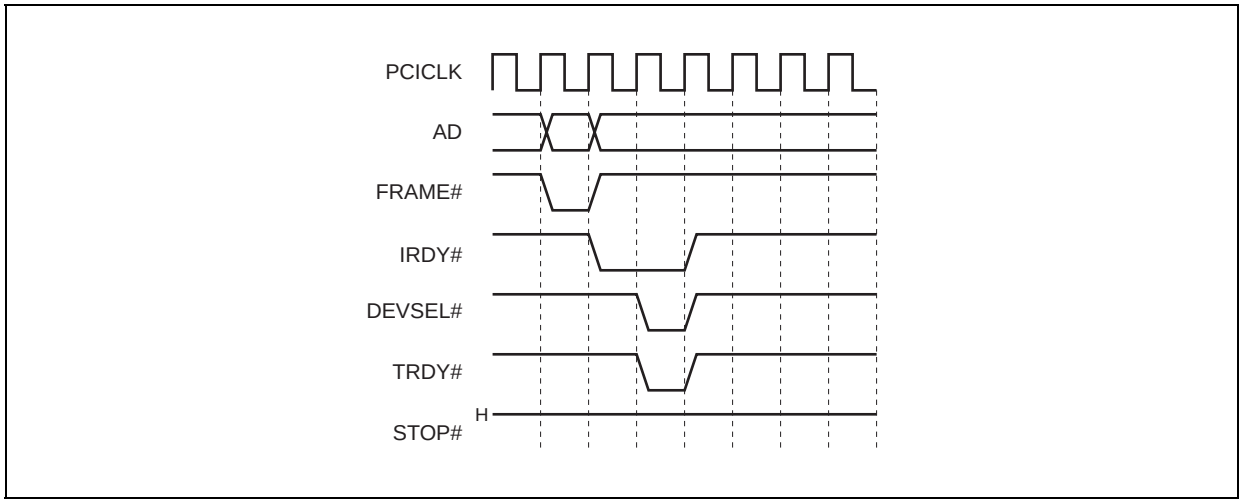
図3－11　Read Cycle



(ii) Write Cycle

Timing Type：Configuration Register Write, Internal I/O Register Write, Memory Write

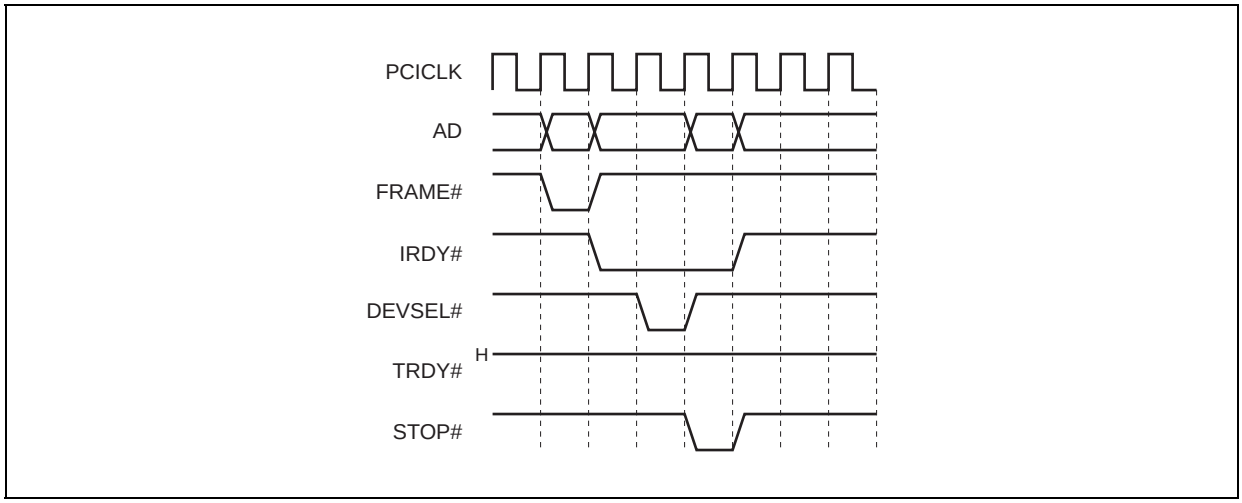
図3－12　Write Cycle



（b）Target Abort Cycle

Timing Type：Target Abort

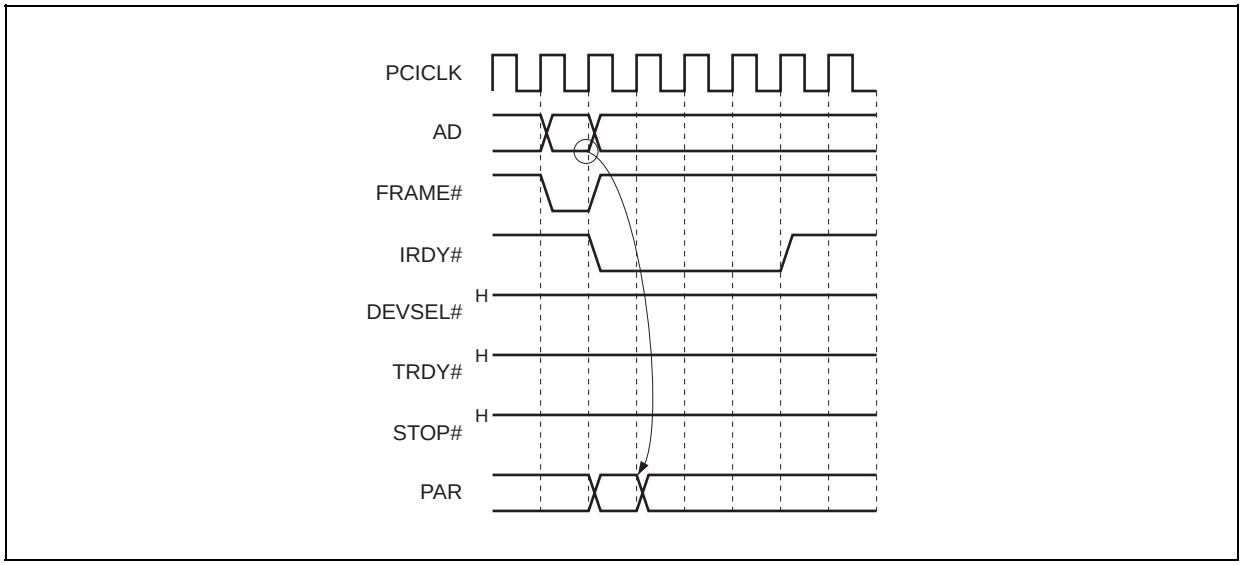
図3－13　Target Abort Cycle



（c）Master Abort Cycle

Timing Type：Master Abort Cycle

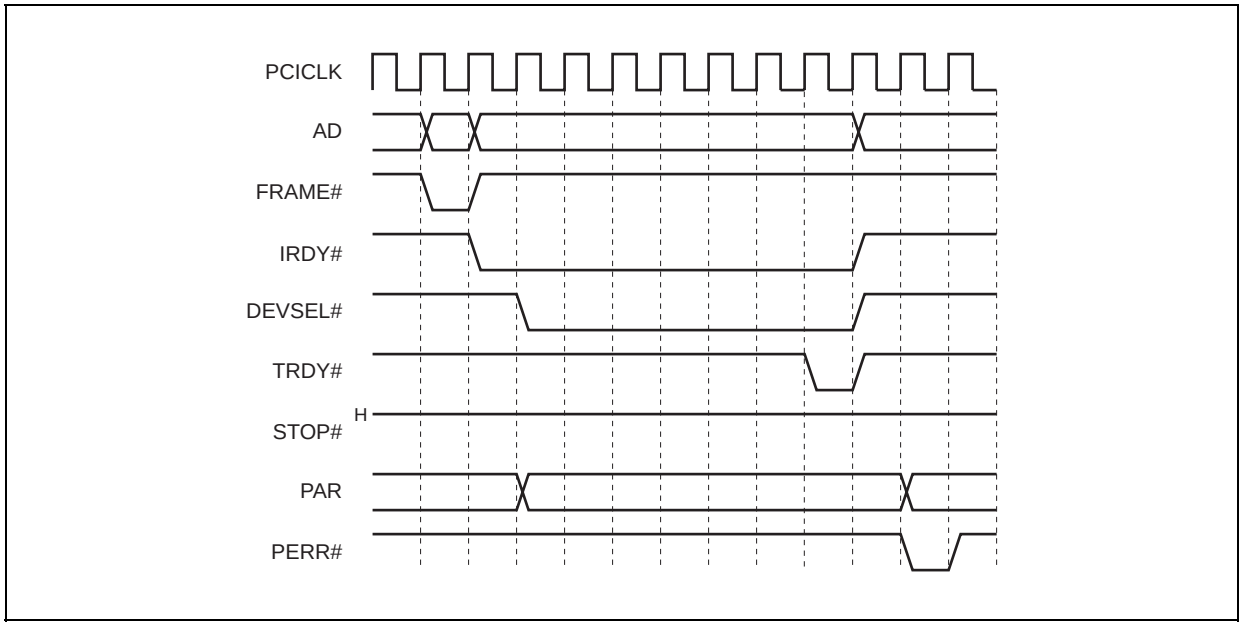
図3－14　Master Abort Cycle



（d）Data Parity Error

Timing Type：Single Read & Write Cycle Data Parity Error

図3－15　Data Parity Error



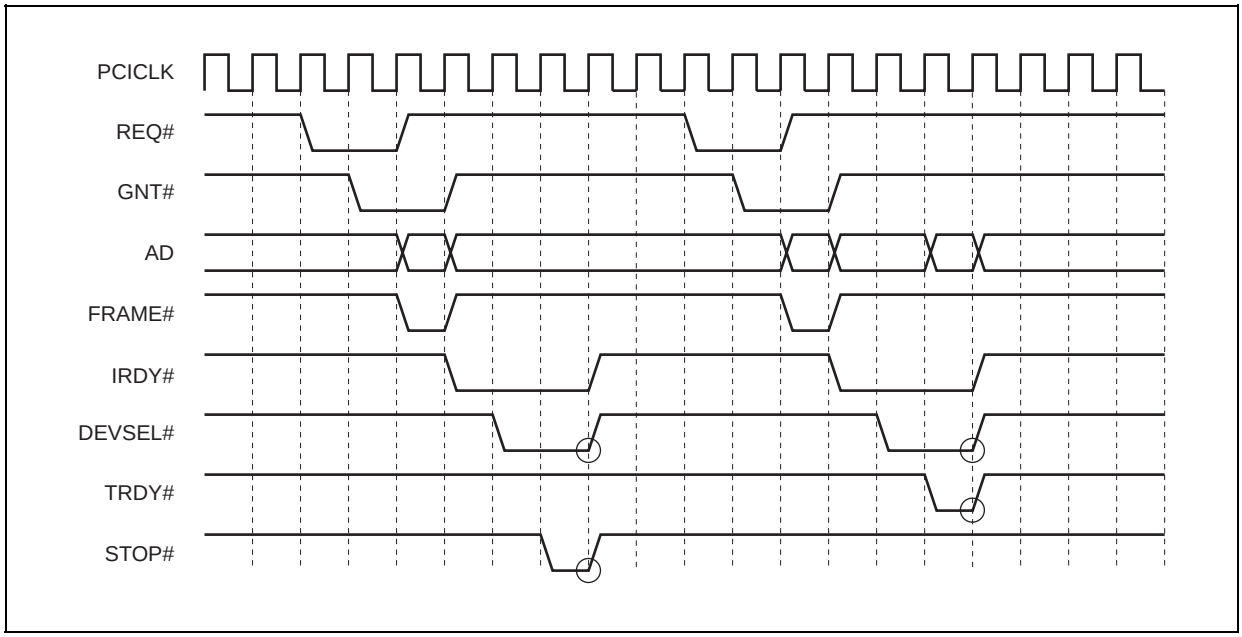
（2）PCIバス・スレーブ・サイクル・タイミング

PCIデバイスからSDRAMへのアクセス・タイミングを示します。

（a）Memory Single Read Cycle

Timing Type：Memory Single Read Cycle

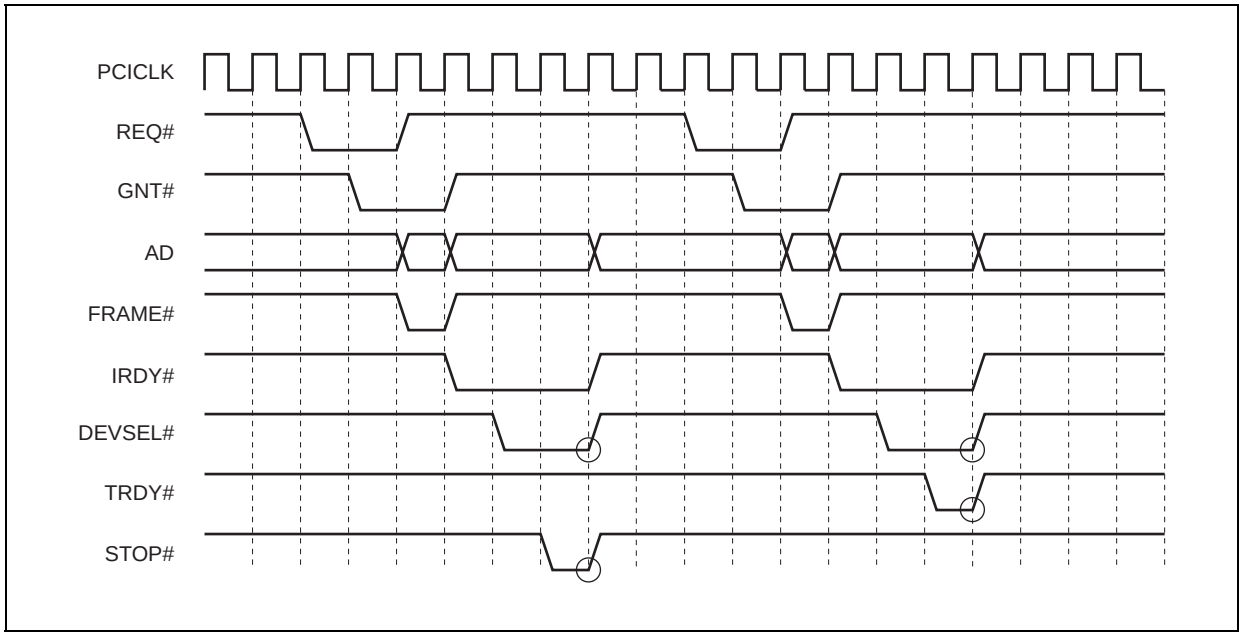
図3－16　Single Read Cycle



（b）Memory Single Write Cycle

Timing Type：Memory Single Write Cycle

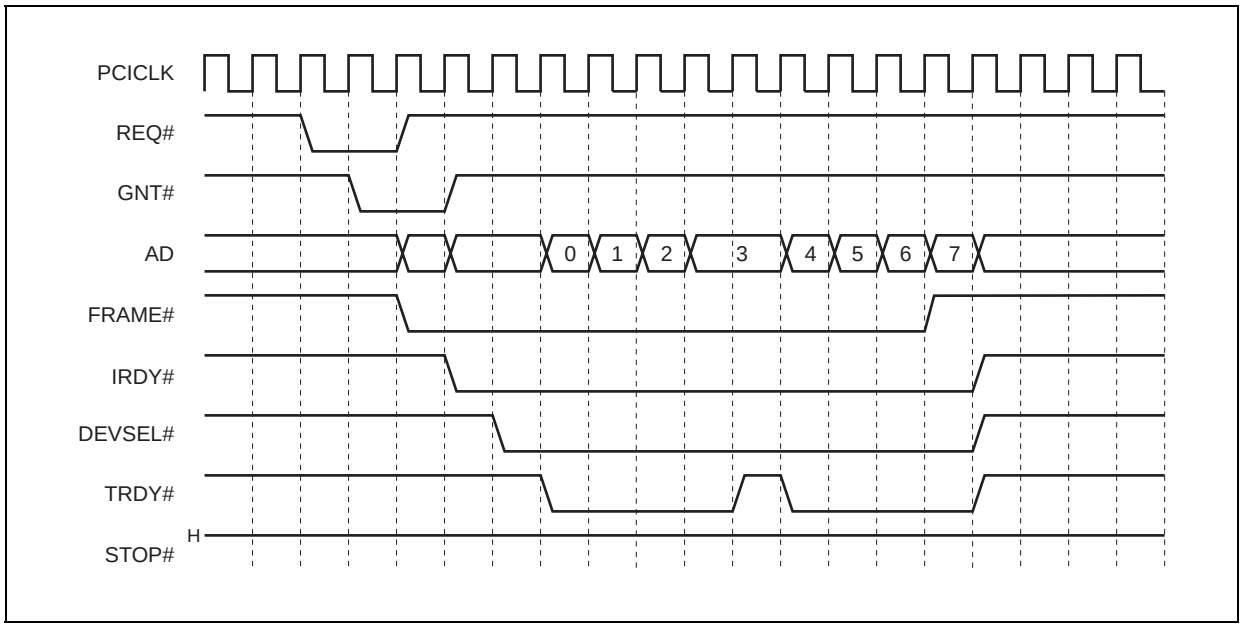
図3－17　Single Write Cycle



（c）Burst Read Cycle

Timing Type：Memory Burst Read Cycle - Not Disconnect

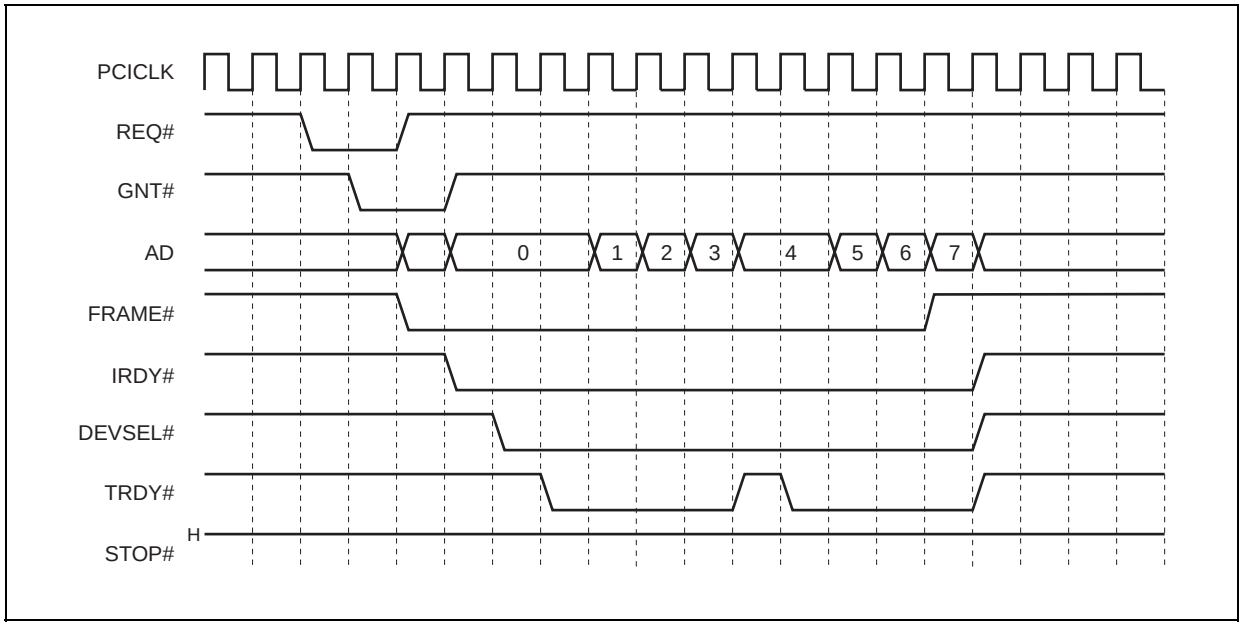
図3－18　Burst Read Cycle



（d）Burst Write Cycle

Timing Type：Memory Burst Write Cycle - Not Disconnect

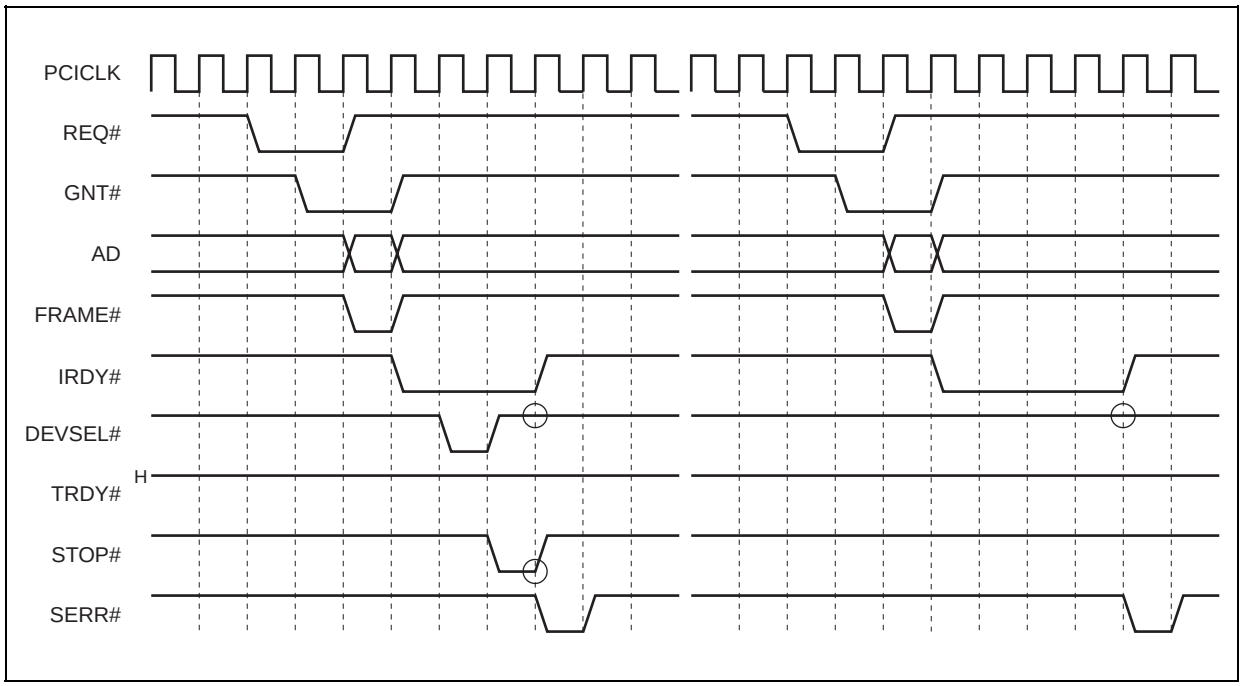
図3－19　Burst Write Cycle



（e）Abort Cycle

Timing Type：Target Abort Cycle & Master Abort Cycle

図3－20　Abort Cycle

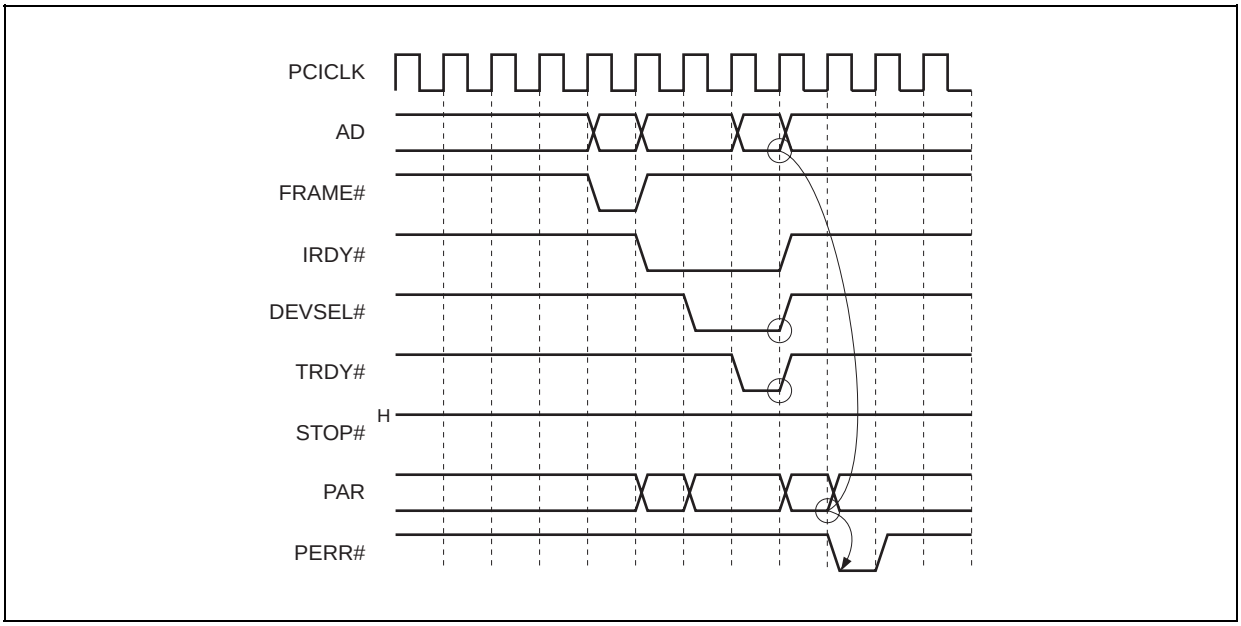


(f) Data Parity Error

(i) Read Data Parity Error 1

Timing Type：Single Read Cycle Data Parity Error

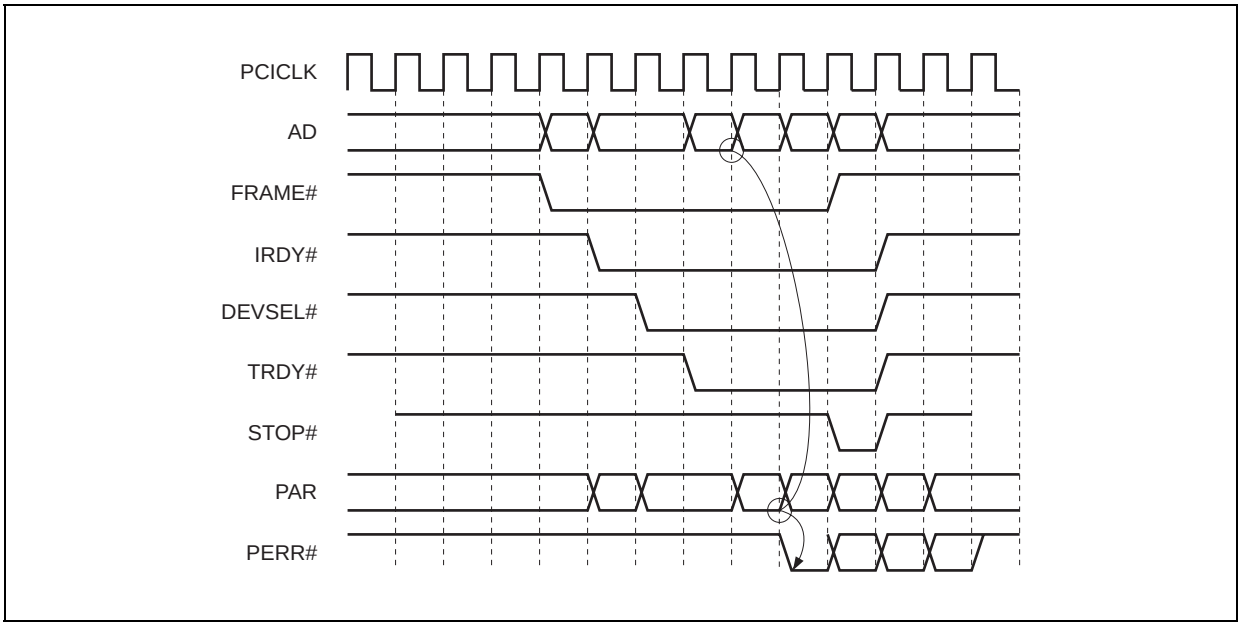
図3－21　Read Data Parity Error



(ii) Read Data Parity Error 2

Timing Type：Burst Read Cycle Data Parity Error

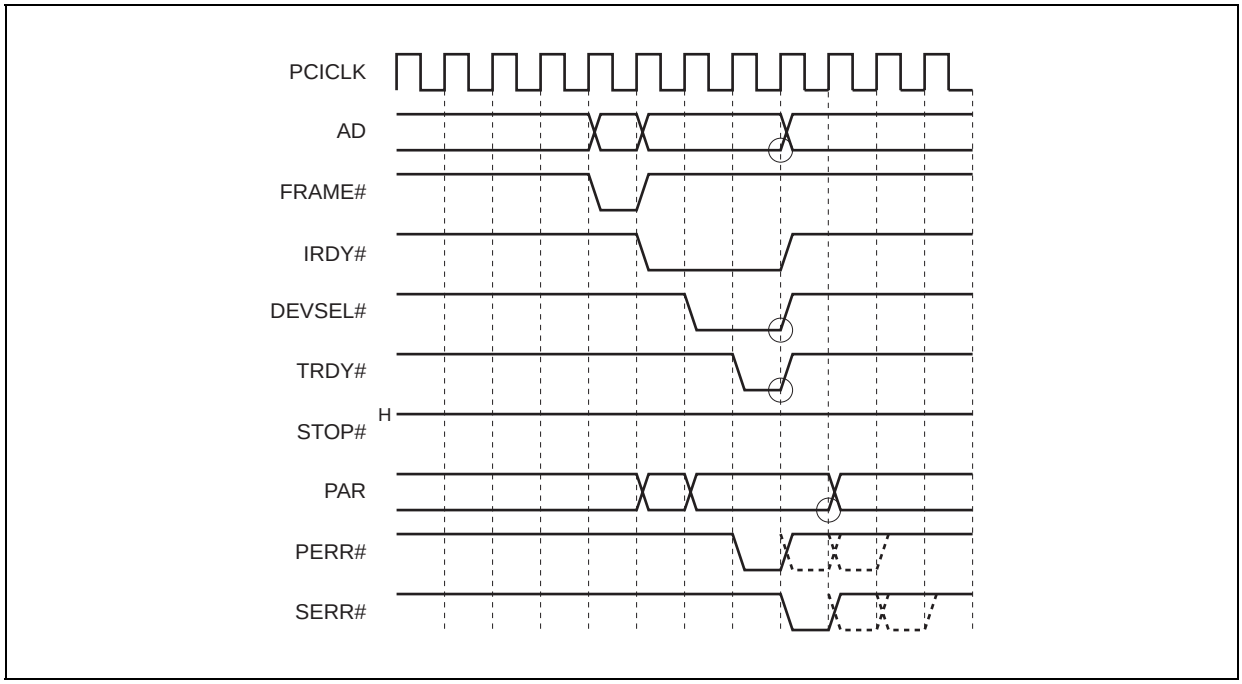
図3－22　Read Data Parity Error 2



（iii）Write Data Parity Error 1

Timing Type：Single Write Cycle Data Parity Error

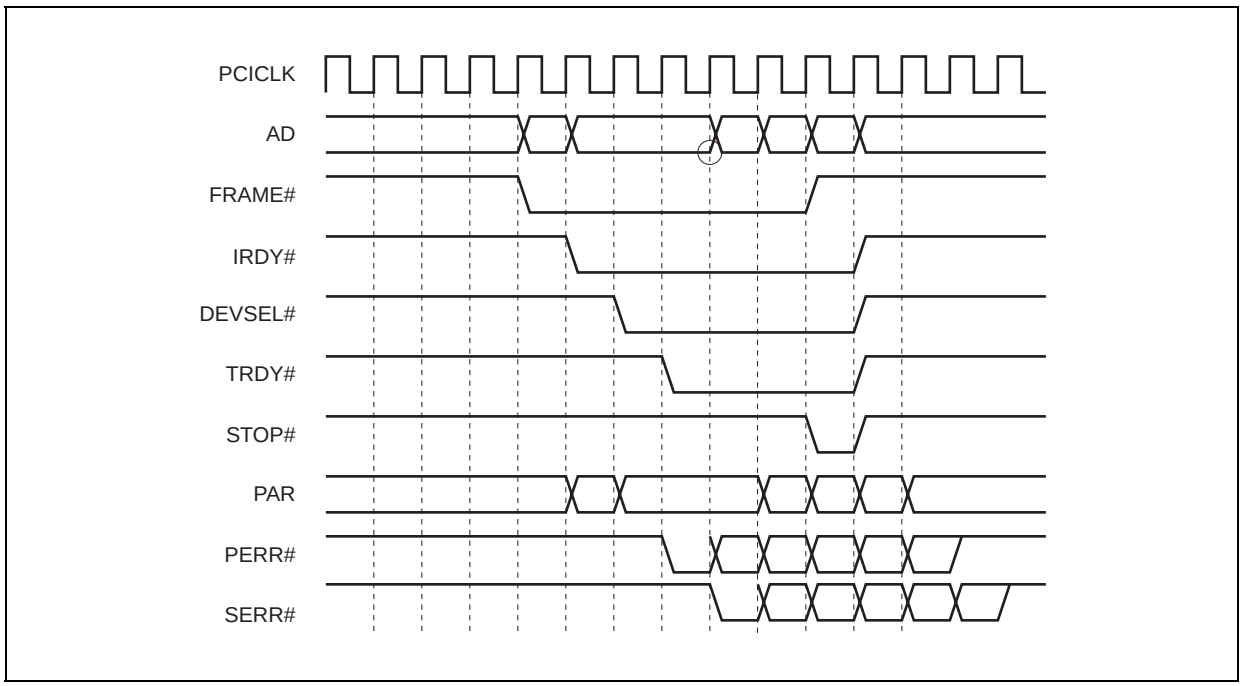
図3－23　Write Data Parity Error 1



（iv）Write Data Parity Error 2

Timing Type：Burst Write Cycle Data Parity Error

図3－24　Write Data Parity Error 2



第4章　FPGAへの組み込み構成例

この章では，PCIホスト・ブリッジ・マクロを，FPGA（Altera社製EP20K200EQC240-1X）に組み込むときの構成例について説明します。

4. 1　FPGA組み込み構成例の条件

構成例の条件を次に示します。

(1) CPU	：V850E/ME2
(2) 外部バス・インタフェースのバス幅	：32ビット
(3) PCIホスト・ブリッジのCS空間	：CSZ6
(4) SDRAMのCS空間	：CSZ3
(5) SDRAM	：16 M×16 SDRAM（4 M×16×4バンク）を2個接続
(6) PCI接続	：2デバイス

4. 2　FPGAトップ階層作成時の留意点

PCIホスト・ブリッジ・マクロをFPGAに組み込むときの留意点を，次に示します。

(1) チップ・セレクトは，アドレスからデコードして作成します。

- I_CPU_CS0_B：PCIホスト・ブリッジ・レジスタ・チップ・セレクト
  （3. 4　レジスタのオフセット・アドレスになります）
- I_CPU_CS1_B：PCI I/O領域チップ・セレクト
  （図3－3　CPUメモリ空間／PCIバスI/O空間アドレス・マップを参照してください）
- I_CPU_CS2_B：PCIメモリ領域チップ・セレクト
  （図3－4　CPUメモリ空間／PCIバス・メモリ空間アドレス・マップを参照してください）

(2) 拡張バス・インタフェースのアドレス・バス，データ・バスのバッファについては，PCIホスト・ブリッジがSDRAMをコントロールするときに出力するため，セレクタを介して双方向端子となります。
また，次に示すSDRAMを制御する端子は，3ステート出力となります。

- DQM0-DQM3, SDCKE, SDCS, SDRAS, SDCAS, SDWEZ

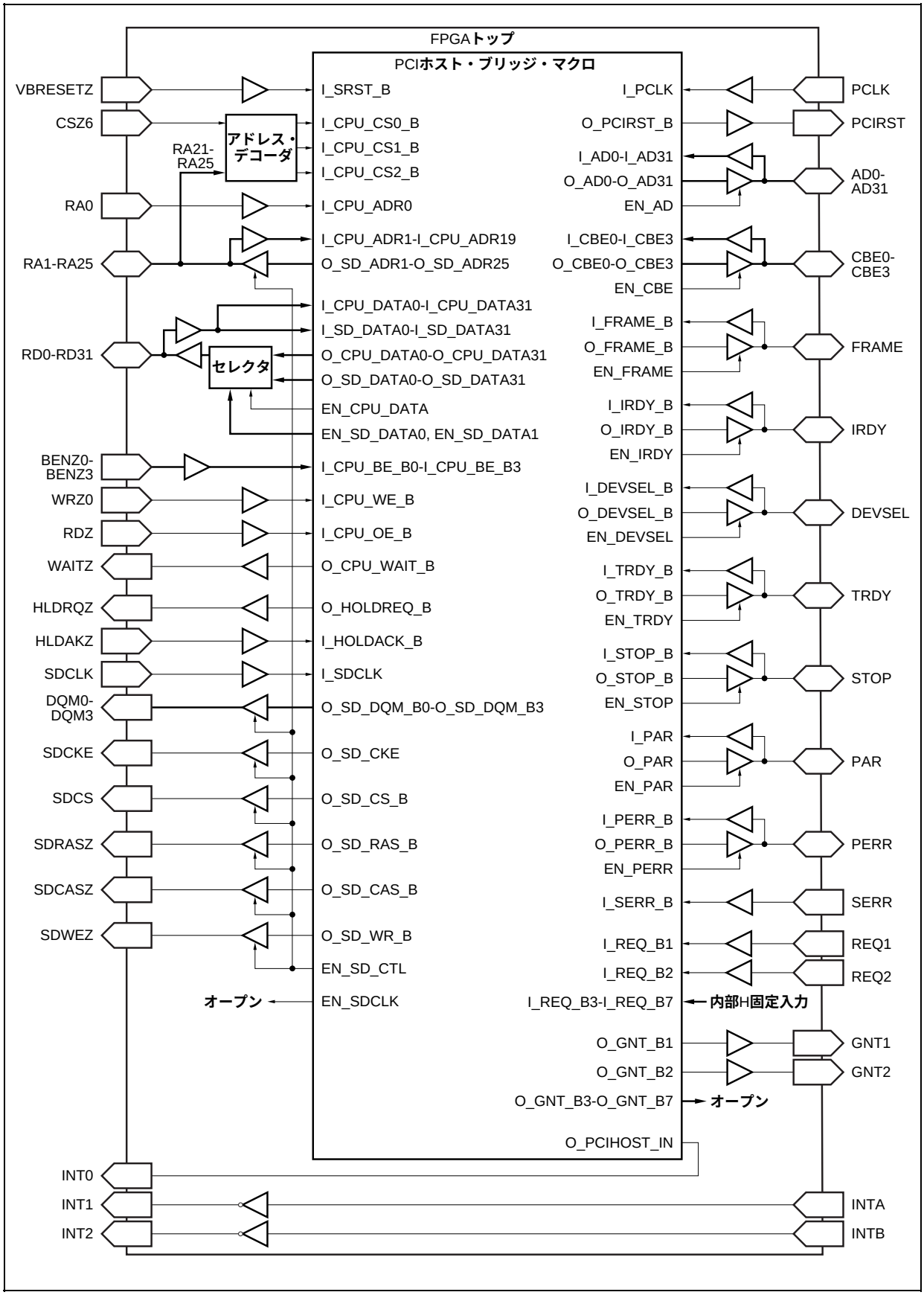
(3) 次に示すPCIバス・インタフェースの端子は，双方向端子となります。

- AD, CBE, FRAME, IRDY, DEVSEL, TRDY, STOP, PAR, PERR

(4) 割り込み要求出力信号は3本です。1本はPCIホスト・ブリッジから出力されます。残り2本は，外部PCIスロットからのINTA, INTB信号で，CPUへ直結となります。

4. 3　FPGAトップ接続参考図

FPGAトップ階層によるPCIホスト・ブリッジ・マクロ接続参考図を，次に示します。



4. 4　FPGAトップ端子機能

PCIホスト・ブリッジ・マクロをFPGAに組み込んだときの端子情報を，次に示します。

4. 4. 1　CPUバス・スレーブ・インタフェース端子

端子名称	入出力	機　　能
VBRESETZ	入力	システム・リセット入力
CSZ6	入力	PCIホスト・ブリッジ・チップ・セレクト入力
RA0-RA25	入出力	CPUアドレス入出力
RD0-RD31	入出力	CPUデータ入出力
BENZ0-BENZ3	入力	CPUデータ・バイト・イネーブル入力
WRZ	入力	CPUデータ・ライト・イネーブル入力
RDZ	入力	CPUデータ・リード・イネーブル入力
WAITZ	出力	CPUデータ・ウエイト出力
INT0	出力	PCIホスト・ブリッジ割り込み出力

4. 4. 2　SDRAMバス・インタフェース端子

端子名称	入出力	機　　能
HLDREQZ	出力	SDRAMバス・ホールド・リクエスト出力
HLDACKZ	入力	SDRAMバス・ホールド・アクノリッジ入力
SDCLK	入力	SDRAMクロック入力
SDCKE	出力	SDRAMクロック・イネーブル出力
SDCS	出力	SDRAMチップ・セレクト出力
SDRASZ	出力	SDRAMロウ・アドレス・ストローブ出力
SDCASZ	出力	SDRAMカラム・アドレス・ストローブ出力
SDWEZ	出力	SDRAMリード／ライト出力
DQM0-DQM3	出力	SDRAMアウトプット・ディスエーブル出力

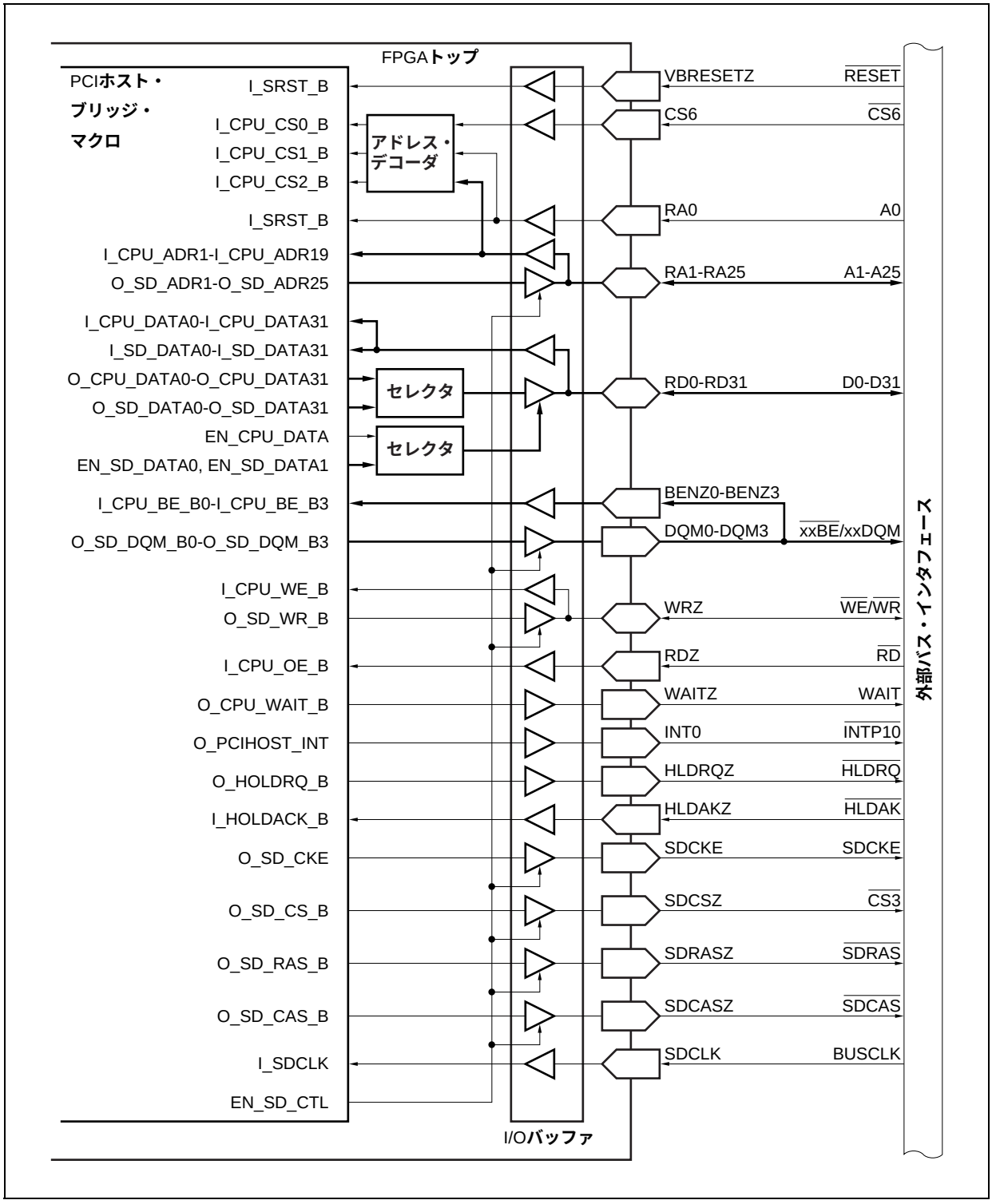
4. 4. 3　PCIバス・インタフェース端子

端子名称	入出力	機　　能
PCLK	入力	PCIクロック入力
PCIRST	出力	PCIリセット出力
AD0-AD31	入出力	PCIアドレス／データ入出力
CBE0-CBE3	入出力	PCIコマンド／バイト・イネーブル入出力
FRAME	入出力	PCIフレーム入出力
IRDY	入出力	PCIイニシエータ・レディ入出力
DEVSEL	入出力	PCIデバイス・セレクト入出力
TRDY	入出力	PCIターゲット・レディ入出力
STOP	入出力	PCIストップ入出力
PAR	入出力	PCIパリティ入出力
PERR	入出力	PCIパリティ・エラー入出力
SERR	入力	PCIシステム・エラー入力
REQ1, REQ2	入力	PCIリクエスト入力
GNT1, GNT2	出力	PCIグラント出力
INT1, INT2	出力	PCI INTA, INTB出力

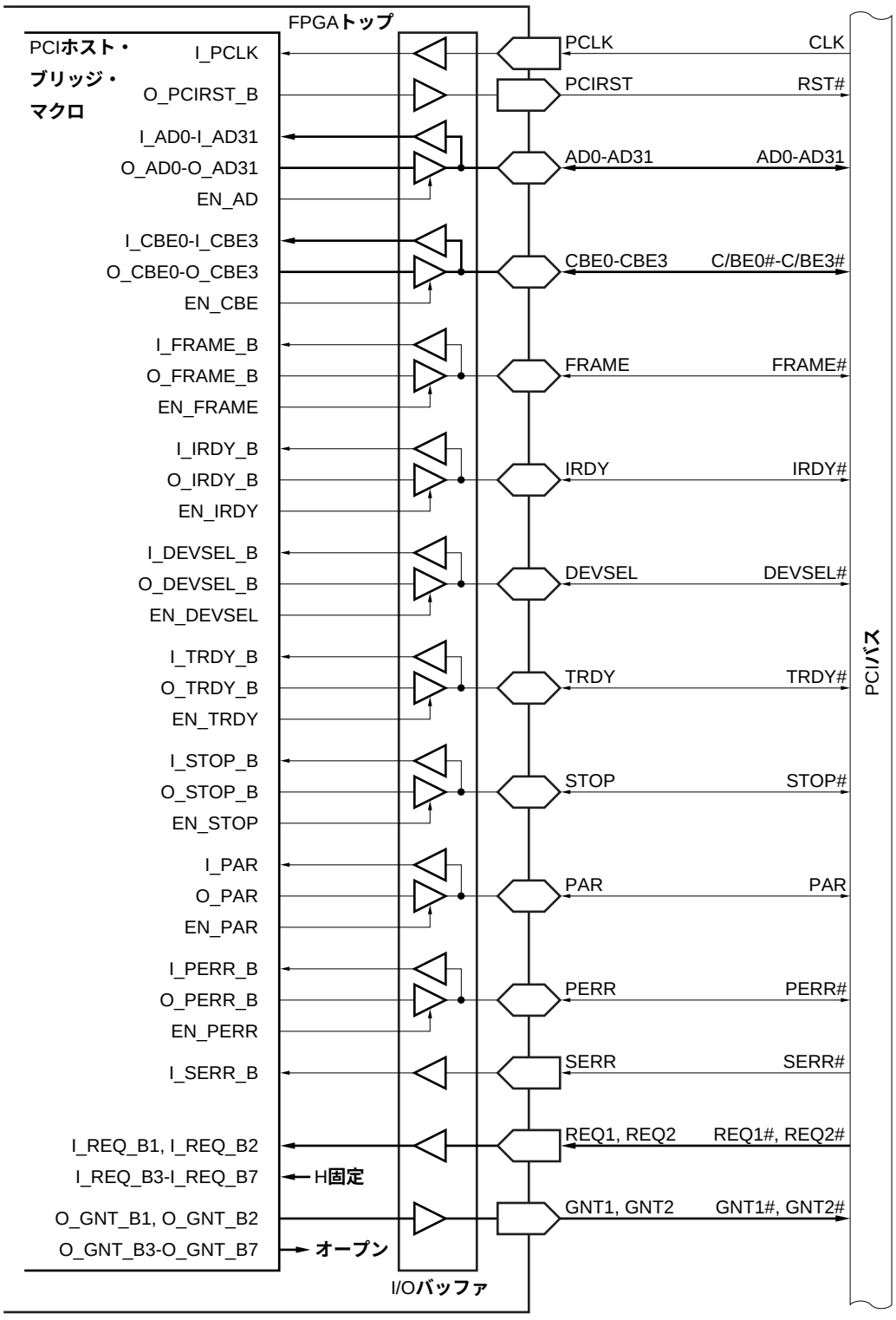
4. 5　FPGAトップ端子接続図

FPGA でのPCIホスト・ブリッジ・マクロ端子の接続図を次に示します。

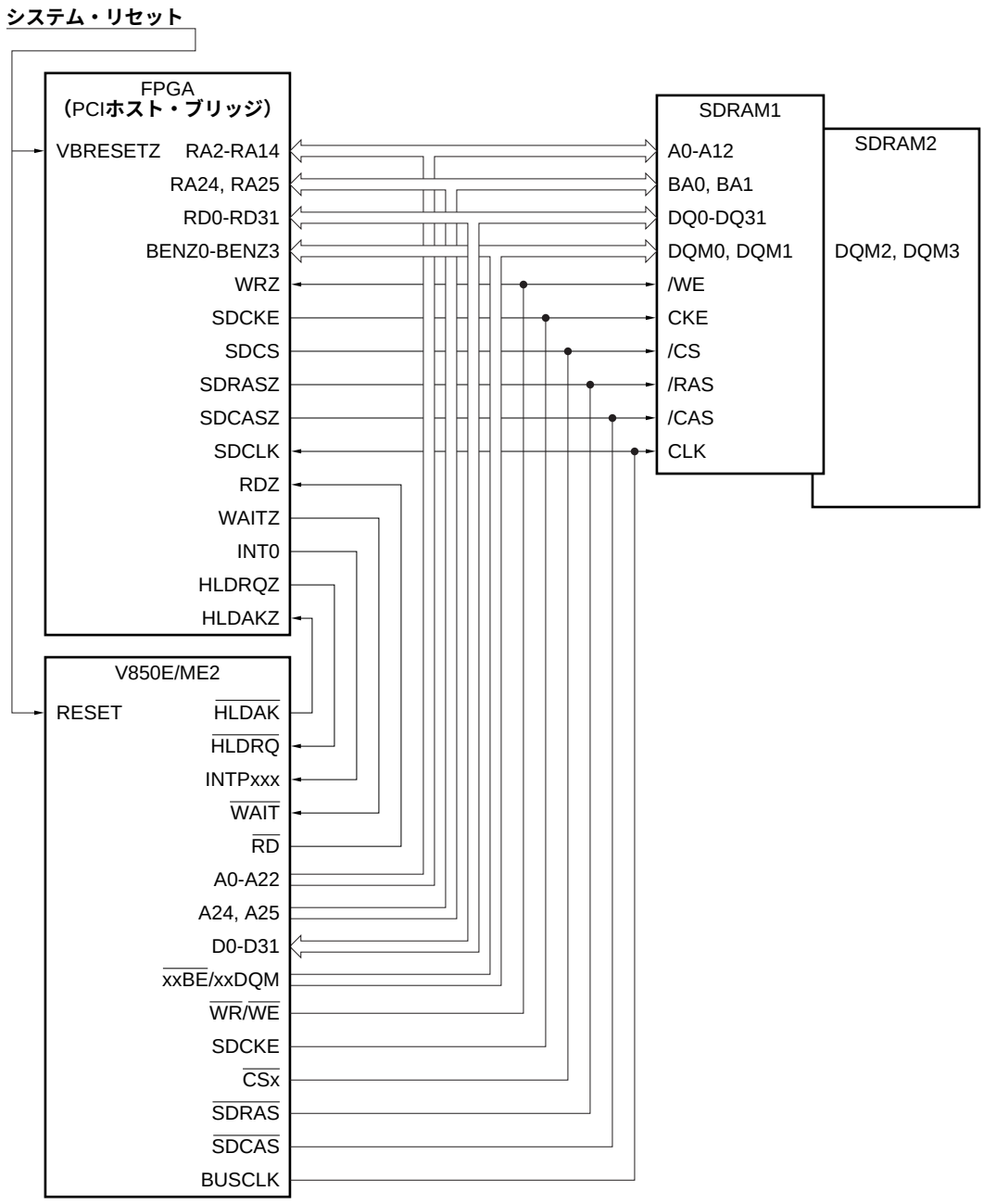
4. 5. 1　外部バス・インタフェース内部接続図



4. 5. 2　PCIバス・インタフェース内部接続図



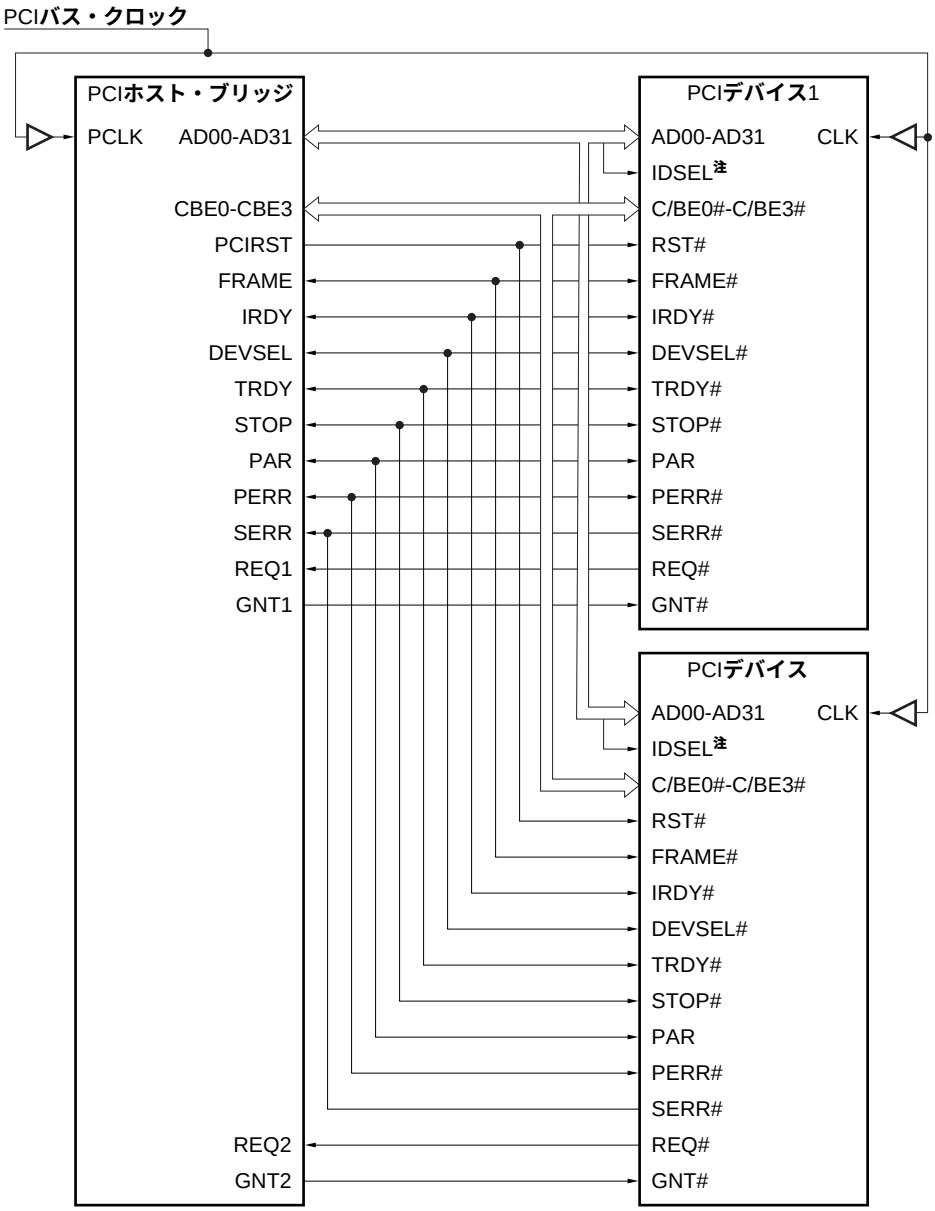
4. 5. 3　外部バス・インタフェース外部接続図（V850E/ME2との接続例）



備考1.　4 Mワード×16ビット×4バンク（ロウ・アドレス：13ビット，カラム・アドレス：9ビット）のSDRAMを2個使用する例です。

2.　xx：LL, LU, UL, UU

4. 5. 4　PCIバス・インタフェース外部接続図



注　各PCIデバイスのIDSEL端子には，AD31-AD11信号のうちの1つを重複しないように接続してください。

4. 6　FPGA設計における注意点

Altera社の「Quartus IIデザイン・ソフトウェア」でFittingするときの注意点を，次に示します。

4. 6. 1　FPGA Fitting設計について

(1) 次に示すPCIバス・インタフェース端子については，“ I/O Standard ”のバッファ・タイプを“ 3.3-V PCI ”に設定してください。

Pin Name/Usage	Dir	I/O Standard
INTA	input	3.3-V PCI
INTB	input	3.3-V PCI
FRAME	bidir	3.3-V PCI
DEVSEL	bidir	3.3-V PCI
REQ1	input	3.3-V PCI
REQ2	input	3.3-V PCI
GNT1	output	3.3-V PCI
GNT2	output	3.3-V PCI
IRDY	bidir	3.3-V PCI
TRDY	bidir	3.3-V PCI
STOP	bidir	3.3-V PCI
PCIRST	output	3.3-V PCI
AD0-AD31	bidir	3.3-V PCI
CBE0-CBE3	bidir	3.3-V PCI
PAR	bidir	3.3-V PCI
PERR	bidir	3.3-V PCI
SERR	input	3.3-V PCI

(2) PCIバス・インタフェース端子については，等長配線を考慮してピン配置を決定してください。

(3) “ PCLK ”，“ SDCLK ”信号は，Global CLKに指定してください。

4. 6. 2　PCIバス・インタフェースTiming Parameterについて（PCI CLK = 33 MHzの制約条件として）

次に示すPCIの規格値を満足するようにTimingを調整してください。

(1) Input Setup Time To CLK Point to Point

端子	Setup	Hold
REQ1, REQ2	10 ns	0 ns
他PCI端子	7 ns	0 ns

（2）CLK to Signal Valid Delay signals

端子	MIN.	MAX.
全PCI端子	2 ns	11 ns

PCIバスのタイミングについては，次に示す規格値があります（PCI CLK = 33 MHz）。

図4－1　Output Timing

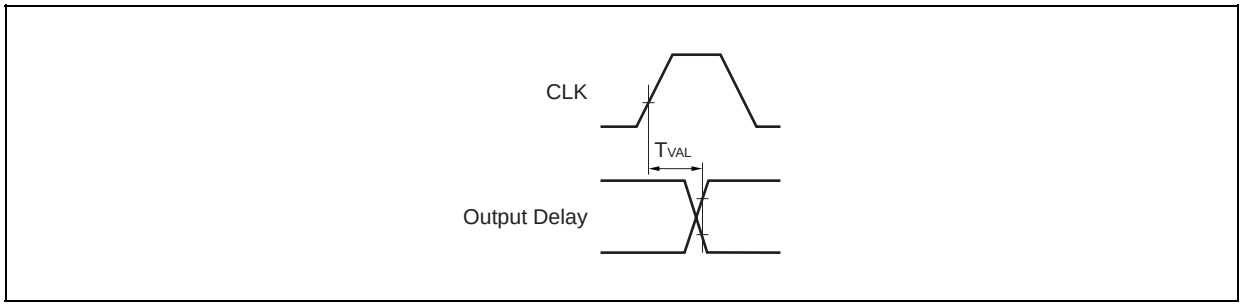


図4－2　Input Timing

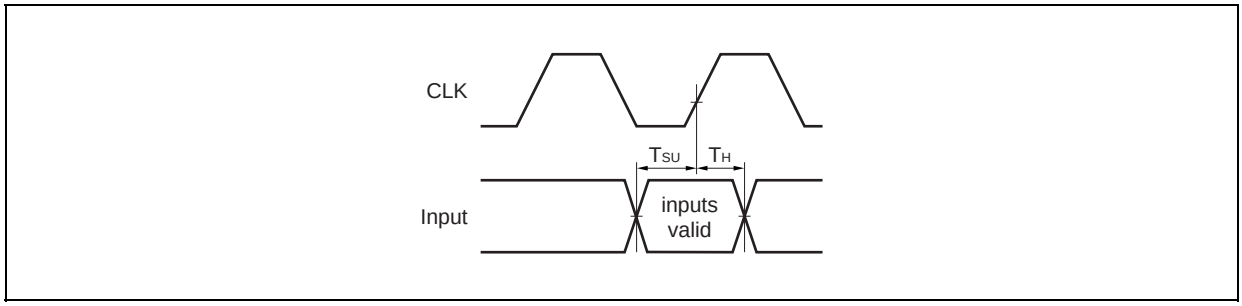


表4－1　33 MHz Timing Parameters

Symbol	Parameter	MIN.（ns）	MAX.（ns）
T_{VAL}	CLK to Signal Valid Delay bused signals	2	11
T_{VAL}（ptp）	CLK to Signal Valid Delay point to point signals	2	12
T_{SU}	Input Setup Time to CLK bused signals	7	
T_{SU}（ptp）	Input Setup Time to CLK point to point signals	10	
T_H	Input Hold Time from CLK	0	

4. 6. 3　SDRAMインタフェース・タイミングについて

SDRAMへのタイミングについては，外部バス・インタフェースや接続するSDRAMに依存します。
システムにあわせて調整してください。

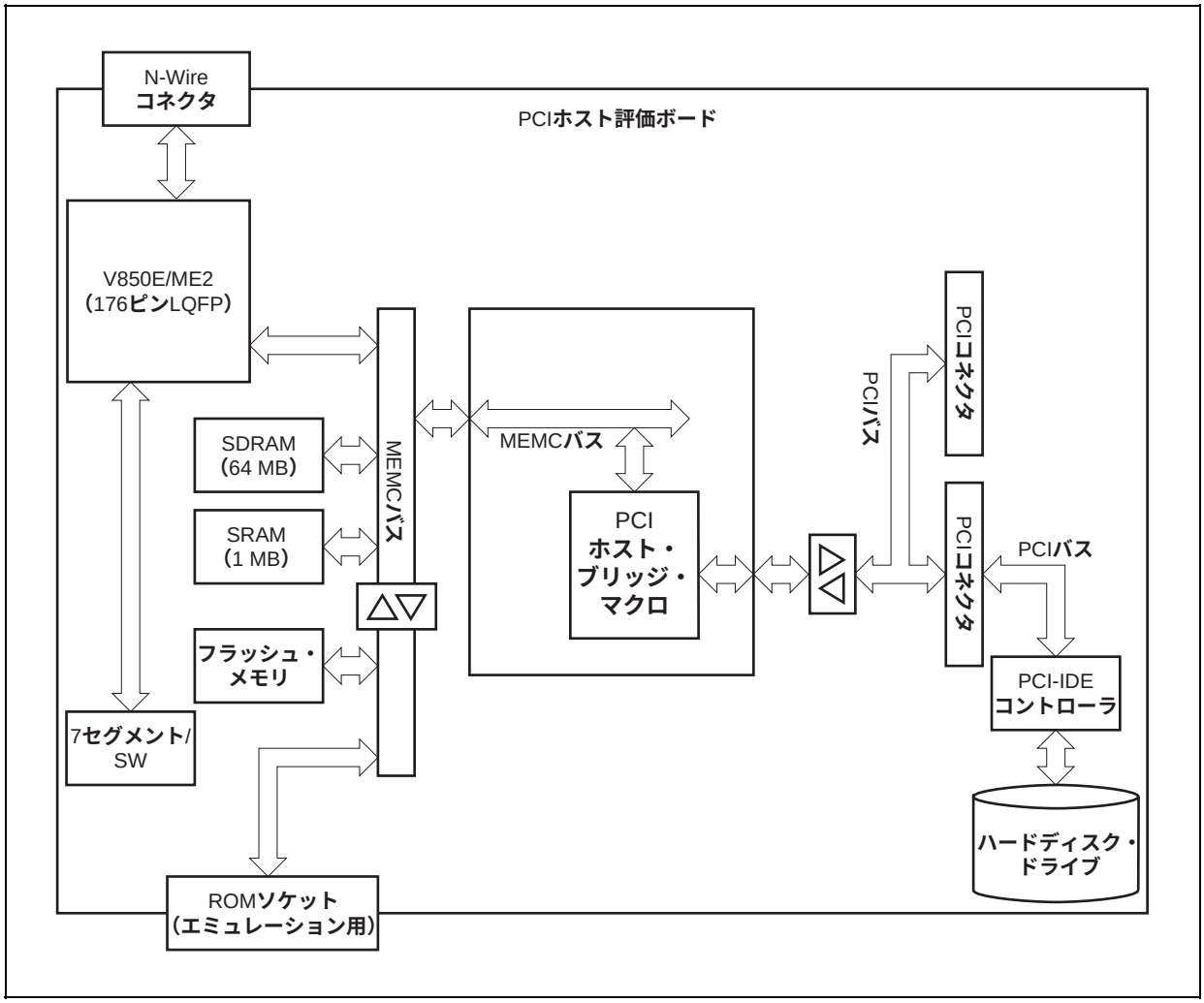
第5章　アプリケーション例

この章では，V850E/ME2を搭載した評価ボードの構成と，プログラム例を紹介します。
PCIコネクタにIDEコントローラを実装し，HDDを動作させるアプリケーション例です。

5. 1　評価ボードのブロック図

評価ボードのブロック図を次に示します。

図5－1　評価ボードのブロック図



5. 2　評価ボードの仕様

評価ボードの仕様を次に示します。

表5－1　評価ボードの仕様一覧

項　　目		説　　明
CPU		V850E/ME2
	CPU動作周波数	30 MHz
	MEMCバス動作周波数	30 MHz
評価ボード搭載メモリ		
	フラッシュ・メモリ	CSZ0領域（32ビット幅）：8 Mバイト
	SRAM	CSZ1領域（32ビット幅）：1 Mバイト
	SDRAM	CSZ3領域（32ビット幅）：64 Mバイト
評価ボード搭載周辺I/O		
	PCIホスト・ブリッジ	CSZ6領域（32ビット幅）：PCI Rev.2.1準拠ホスト・インタフェース（33 MHz）
その他		
	7セグメント表示	V850E/ME2汎用ポートにより，7セグメント表示×2個を表示制御可能

PCIバスのデバイス番号は，次に示すように割り当てています。

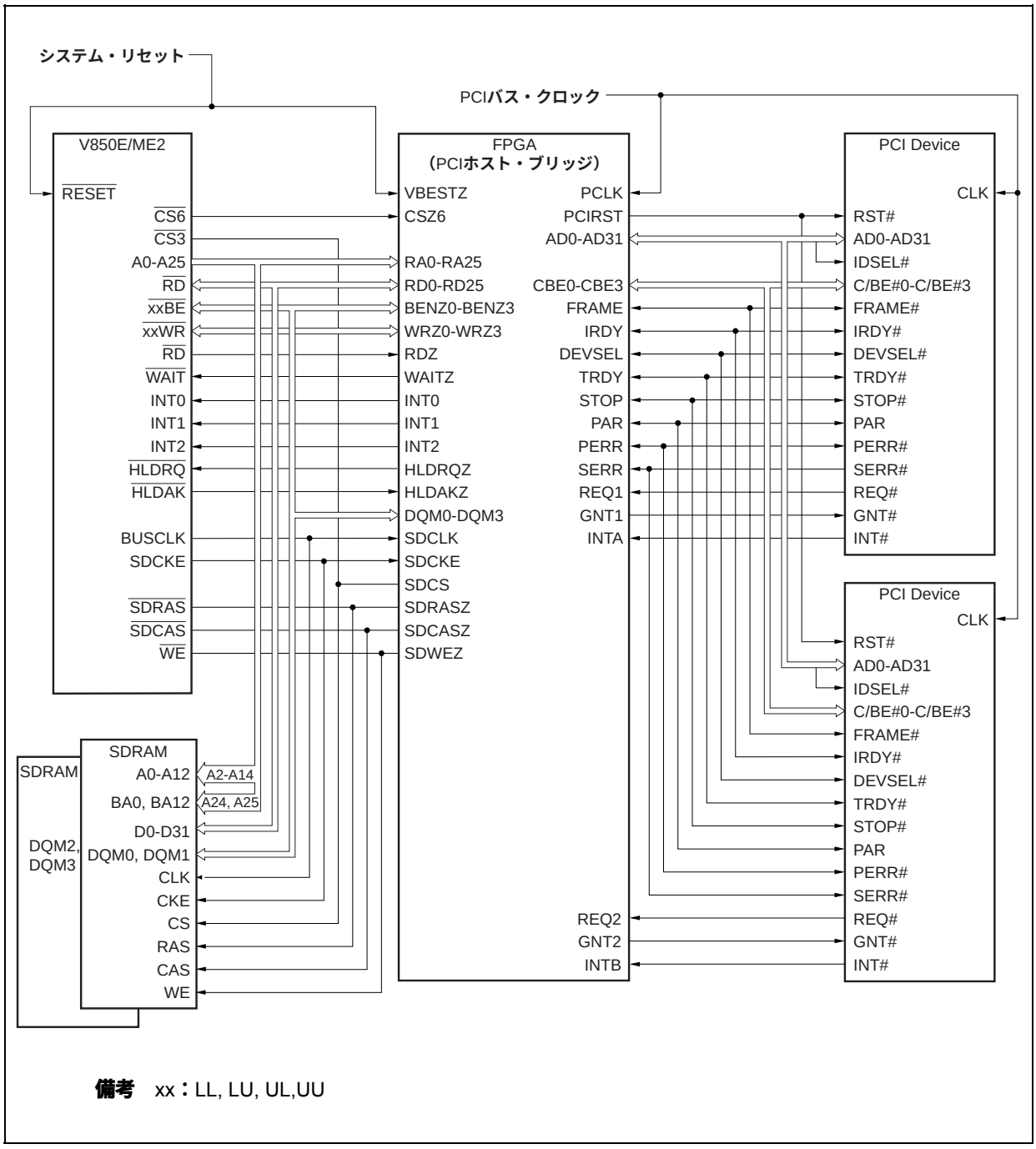
表5－2　IDSEL接続について

Slot	デバイス番号	備　　考
PCI Slot 1（J2）	AD31	IDSELにAD31を接続
PCI Slot 2（J3）	AD30	IDSELにAD30を接続

5. 3　評価ボードの接続回路例

V850E/ME2と，SDRAM, FPGA, PCIデバイス（スロット）への接続回路例を，次に示します。

図5－2　評価ボードの接続回路例

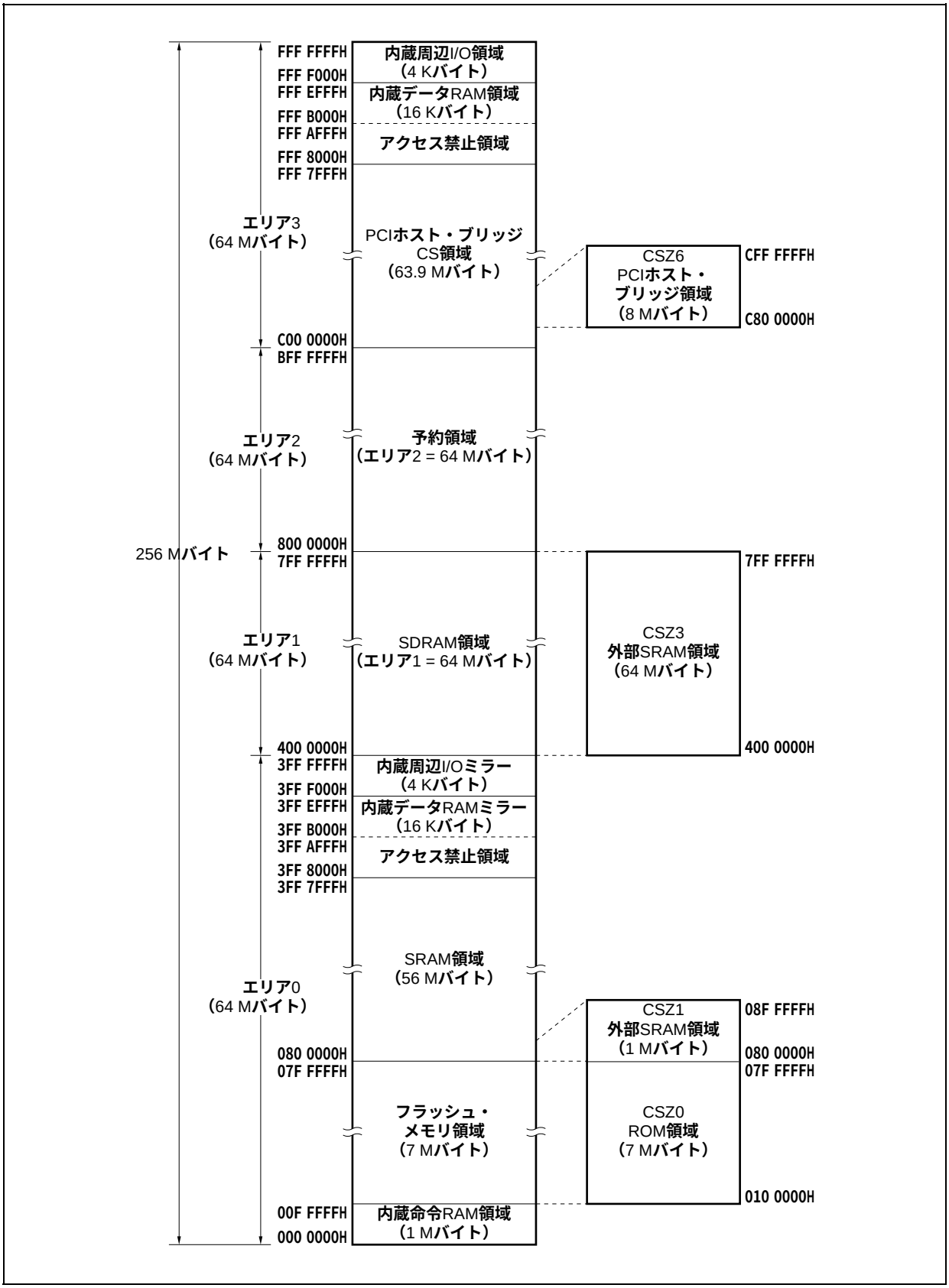


備考　xx：LL, LU, UL,UU

5. 4　評価ボードのメモリ空間

評価ボードのメモリ空間を次に示します。

図5－3　評価ボードのメモリ空間一覧



PCIメモリI/O空間は，CSZ6領域に割り当てられます。

PCIメモリ空間のベース・アドレスは，CC0 0000Hに設定されます。

PCI I/O空間のベース・アドレスは，C800000Hに設定されます。

図5－4　CPUメモリ空間とPCIメモリ空間の対比

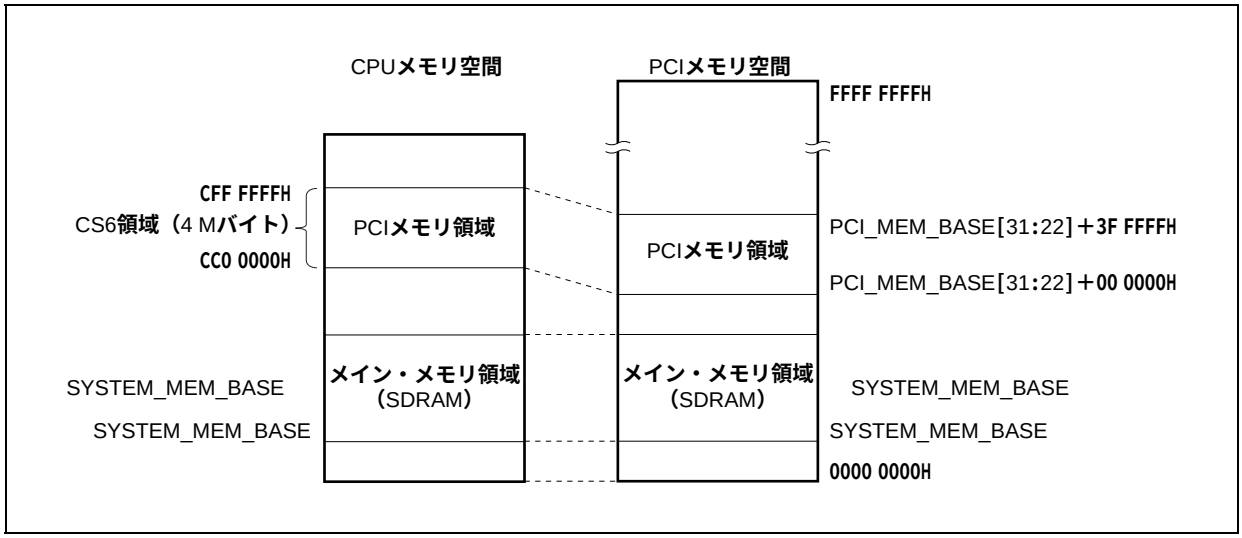
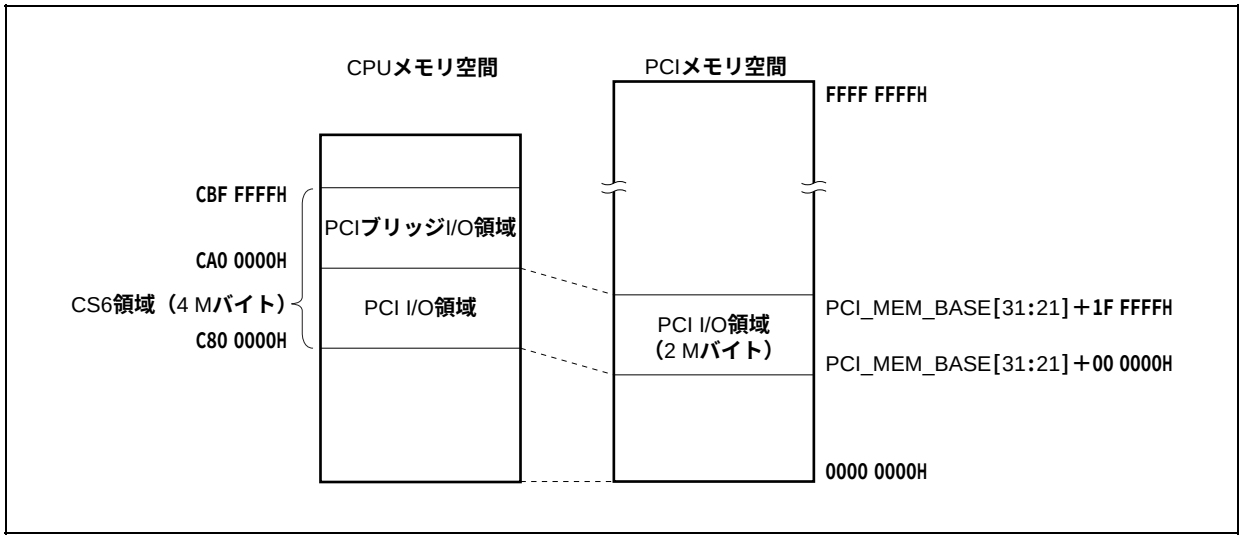


図5－5　CPUメモリ空間とPCI I/O空間の対比



5. 5　サンプル・プログラム例

このサンプル・プログラムは，V850E/ME2評価ボードにPCIデバイスとしてPCI-IDEボードを接続した環境を想定しています。

PCI-IDEボードにはIDE HDDを接続し，サンプル・プログラムはIDE HDDにアクセスします。

5. 5. 1　開発ツール

（1）MULTI™ 1.8.9

Green Hills Software™, Inc. 製統合開発環境です。

（2）PCI-IDEボード

評価ボードに接続するPCIインタフェースのIDEカードです。

このアプリケーションでは，IDE HDDを接続して使用します。

5. 5. 2　プログラムの構成

サンプル・プログラムの構成を次に示します。

（1）PCIホスト・ブリッジ・マクロ初期化サンプル・プログラム・リスト

CPUがPCI領域にアクセスするために，まずPCIホスト・ブリッジ・マクロを初期化する必要があります。

PCIメモリ空間とCPUメモリ空間の対応づけや，PCIからの割り込み，CPUからPCIメモリ空間へのアクセス制御などを設定します。

設定は，PCIホスト・ブリッジ・マクロのレジスタで行います。

（2）PCIコンフィギュレーション空間アクセス・サンプル・プログラム・リスト

PCIホスト・ブリッジ・マクロの初期化が終わると，次はPCIバスに接続された個々のPCIデバイスの初期化を行います。初期化は，主に各PCIデバイスに存在するコンフィギュレーション空間レジスタを設定することで行います。コンフィギュレーション空間レジスタは，レジスタ説明に記述してあるPCI_CONFIG_ADDレジスタとPCI_CONFIG_DATAレジスタを使用して，コンフィギュレーション・サイクルを実行することによってだけアクセスできます。

（3）IDE HDDアクセス・サンプル・プログラム・リスト

PCIホスト・ブリッジ・マクロの初期化，PCIデバイスの初期化が終わると，実際にPCIデバイスを操作できるようになります。PCIデバイスの操作は，コンフィギュレーション空間やPCI IO領域に割り付けられたレジスタを設定することで行います。

このサンプル・コードでは，PCI-IDEボードのレジスタを操作することでIDEバスの設定を行い，実際にIDE HDDへのアクセスを行います。

5. 5. 3　V850E/ME2 PCIホスト・ブリッジ・マクロ初期化サンプル・プログラム・リスト

```
//////////////////////////////////////////////////////////////////
//  V850E/ME2 - PCI Host Bridge Macro 初期化サンプル              //
//    概要： PCI Bridge IO 領域のレジスタ群を設定することで、      //
//          PCI Host Bridge Macro の初期化を行います。           //
//          関数 PCI_HBM_Init() に具体的な初期化を記述しています。 //
//                                                              //
//    PCI_HBM_Init() は、CPU や周辺I/Oなど、Host Bridge Macro への //
//    アクセスに必要な機能の初期化終了後に呼び出すものとします。    //
//                                                              //
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
//  PCI領域、SDRAM領域のベースアドレスを定義します。              //
//  本アプリケーションでは、PCI 領域の先頭アドレスを0C80_0000H、  //
//  SDRAM領域の先頭アドレスを 0400_0000H とします。              //
//////////////////////////////////////////////////////////////////
#define BASE_ADDRESS_ME2PCIIF           (0x0C800000)

#define BASE_ADDRESS_PCI_IO             (BASE_ADDRESS_ME2PCIIF)
#define BASE_ADDRESS_PCI_BRIDGE_IO      (BASE_ADDRESS_ME2PCIIF + 0x00200000)
#define BASE_ADDRESS_PCI_MEM            (BASE_ADDRESS_ME2PCIIF + 0x00400000)

#define BASE_ADDRESS_SDRAM              (0x04000000)
#define RANGE_SDRAM                     (0x03FFFFFF)  // 64MB

////////////////////////////////////////////
//  PCI Host Bridge Macro レジスタ アドレス定義  //
////////////////////////////////////////////
#define PHBMR_PCI_CONFIG_DATA           (BASE_ADDRESS_PCI_BRIDGE_IO+0x00)
#define PHBMR_PCI_CONFIG_ADD            (BASE_ADDRESS_PCI_BRIDGE_IO+0x04)
#define PHBMR_PCI_CONTROL               (BASE_ADDRESS_PCI_BRIDGE_IO+0x08)
#define PHBMR_PCI_IO_BASE               (BASE_ADDRESS_PCI_BRIDGE_IO+0x10)
#define PHBMR_PCI_MEM_BASE              (BASE_ADDRESS_PCI_BRIDGE_IO+0x14)
#define PHBMR_PCI_INT_CTL               (BASE_ADDRESS_PCI_BRIDGE_IO+0x18)
#define PHBMR_PCI_ERR_ADD               (BASE_ADDRESS_PCI_BRIDGE_IO+0x1C)
#define PHBMR_SYSTEM_MEM_BASE           (BASE_ADDRESS_PCI_BRIDGE_IO+0x40)
#define PHBMR_SYSTEM_MEM_RANGE          (BASE_ADDRESS_PCI_BRIDGE_IO+0x44)
#define PHBMR_SDRAM_CTL                 (BASE_ADDRESS_PCI_BRIDGE_IO+0x48)
```

```
//////////////////////////////
//  レジスタアクセス用マクロ定義  //
//////////////////////////////
#define V850EME2_REGW(x)        *((volatile unsigned int *)((int)x))

/////////////////////////////////////////////////////////////////
// 関数名：PCI_HBM_Init                                        //
//   機能：PCI Host Bridge Macro を初期化します。               //
//   引数：無し                                                //
// 戻り値：無し                                                //
//   備考：本初期化サンプルの各ベースアドレスは次の通りです。  //
//         - PCI I/O 空間のベース・アドレス：      0C80_0000H    //
//         - PCI メモリ空間のベース・アドレス：    0CC0_0000H    //
//         - メイン・メモリ(SDRAM)をマッピングする               //
//           PCIバス・メモリ空間上のベース･アドレス：0400_0000H  //
//         - メイン・メモリ(SDRAM)をマッピングする               //
//           PCIバス・メモリ空間の範囲：           03FF_FFFFH    //
//        また、その他の設定もシステム要求や実装に応じて、設定する//
//        必要があります。                                       //
/////////////////////////////////////////////////////////////////
void PCI_HBM_Init(void)
{

    V850EME2_REGW(PHBMR_PCI_CONTROL) = 0x07000110;
    // PCI_CONTROL レジスタ
    //   bit 31-24:  PCI_PARKCNT = 1
    //               (バスパーキングに移行する時間を7に設定します)
    //   bit 15-08:  PCI_REQ = 1          (I_REQ_B0 を有効にします)
    //   bit     4:  PCI_RESET ビット = 1  (PCIバスのリセットを解除します)

    V850EME2_REGW(PHBMR_PCI_IO_BASE) = BASE_ADDRESS_PCI_IO;
    // PCI_IO_BASE レジスタ
    //   PCI I/O空間ベース・アドレスを C800000H に設定します。

    V850EME2_REGW(PHBMR_PCI_MEM_BASE) = BASE_ADDRESS_PCI_MEM;
    // PCI_MEM_BASE レジスタ
    //   PCI メモリ空間ベース・アドレスを CC00000H に設定します。

    V850EME2_REGW(PHBMR_PCI_CONTROL) = 0x07000113;
    // PCI_CONTROL レジスタ
    //   bit 31-24:  PCI_PARKCNT = 1
    //               (バスパーキングに移行する時間を7に設定します)
    //   bit 15-08:  PCI_REQ = 1          (I_REQ_B0 を有効にします)
```

```
    //  bit     4:  PCI_RESET ビット = 1  (PCIバスのリセットを解除します)
    //  bit     1:  PCI_MEM_EN ビット = 1
    //              (CPUからPCIメモリ領域へのアクセスを許可します)
    //  bit     0:  PCI_IO_EN ビット = 1
    //              (CPUからPCI I/O領域へのアクセスを許可します)

    V850EME2_REGW(PHBMR_SYSTEM_MEM_BASE) = BASE_ADDRESS_SDRAM;
    // SYSTEM_MEM_BASE レジスタ
    //   メイン・メモリ(SDRAM)をマッピングするPCIバス・メモリ空間上の
    //   ベース・アドレスを 4000000H に設定します。

    V850EME2_REGW(PHBMR_SYSTEM_MEM_RANGE) = RANGE_SDRAM;
    // SYSTEM_MEM_RANGE レジスタ
    //   メイン・メモリ(SDRAM)をマッピングするPCIバス・メモリ空間の
    //   範囲を 3FFFFFFH(64MB) に設定します。

    V850EME2_REGW(PHBMR_SDRAM_CTL) = 0x00071211;
    // SDRAM_CTL レジスタ
    //  bit 23-16:  CYCLE_LATENCY = 07H
    //              ( PCIデバイスからの連続したメイン・メモリ(SDRAM)
    //                アクセスに対するレイテンシを 210ns に設定します)
    //  bit    12:  BUS_SIZE = 1B
    //              (データ・バスのビット幅を32ビット幅に設定します)
    //  bit 09-08:  CAS_LATENCY = 10B (CASレイテンシを 2 に設定します)
    //  bit 05-04:  WAIT_STATE = 01B
    //              (ACT→CMD, PRE→ACT, CMD→ACT のウェイト間隔を
    //               1クロックに設定します)
    //  bit 01-00:  COLUMN_SIZE = 01B
    //              (カラム・アドレスのビット幅を9ビット幅に設定します)

    V850EME2_REGW(PHBMR_PCI_CONTROL) = 0x07000717;
    //  bit 31-24:  PCI_PARKCNT = 1
    //              (バスパーキングに移行する時間を7に設定します)
    //  bit 15-08:  PCI_REQ = 1          (I_REQ_B0 を有効にします)
    //  bit     4:  PCI_RESET ビット = 1  (PCIバスのリセットを解除します)
    //  bit     1:  PCI_MEM_EN ビット = 1
    //              (CPUからPCIメモリ領域へのアクセスを許可します)
    //  bit     0:  PCI_IO_EN ビット = 1
    //              (CPUからPCI I/O領域へのアクセスを許可します)

    return;
}
```

```
//////////////////////////////////////////////////////////////////
// 関数名：main                                                 //
//   機能：PCI Host Bridge Macro を初期化します。               //
//   引数：無し                                                 //
// 戻り値：0 :正常終了                                          //
//////////////////////////////////////////////////////////////////
int main(void)
{
    // PCI Host Bridge Macro を初期化します。
    PCI_HBM_Init();

    return 0;
}
```

5. 5. 4　PCIコンフィギュレーション空間アクセス・サンプル・プログラム・リスト

```
//////////////////////////////////////////////////////////////////////
//  PCIコンフィギュレーション空間アクセスサンプル                      //
//    概要：コンフィギュレーション空間へのアクセスは、下記の手順で     //
//          行われます。                                              //
//          1) PCI Host Bridge Macro の PCI_CONFIG_ADD レジスタに、    //
//             アクセスするPCIデバイス、機能番号、レジスタ番号を      //
//             示す 32bit値を Write します。                          //
//          2) コンフィギュレーション空間レジスタをReadする場合、     //
//             PCI Host Bridge Macro の PCI_CONFIG_DATA レジスタを    //
//             32bit Read (ワードアクセス)します。                    //
//             コンフィギュレーション空間レジスタをWrite したい場合、 //
//             PCI Host Bridge Macro の PCI_CONFIG_DATA レジスタを    //
//             に、32bit値を Write (ワードアクセス)します。           //
//                                                                    //
//          この手順を関数にまとめたものが、下記の関数PCI_ConfigRead  //
//          と、関数 PCI_ConfigWrite です。                           //
//          関数 PCI_Config_BaseAddressInit では、関数PCI_ConfigWrite //
//          を使い、コンフィギュレーション空間内のベースアドレス      //
//          レジスタの設定を行っています。                            //
//                                                                    //
//////////////////////////////////////////////////////////////////////

/////////////
// 型宣言 //
/////////////
typedef char                            BYTE;
typedef short int                       HWORD;
typedef int                             WORD;
typedef unsigned char                   UBYTE;
typedef unsigned short int              UHWORD;
typedef unsigned int                    UWORD;
typedef volatile unsigned char          VUBYTE;
typedef volatile unsigned short int     VUHWORD;
typedef volatile unsigned int           VUWORD;

//////////////////////////////////////////////////////////////////
// 関数名：PCI_ConfigRead                                       //
//   機能：PCI コンフィギュレーション空間に 32bit値をReadします。//
//   引数：ConfigAdd：コンフィギュレーション空間のレジスタアドレス//
// 戻り値：Read した コンフィギュレーション空間レジスタデータ    //
//////////////////////////////////////////////////////////////////
```

```
UWORD PCI_ConfigRead(UWORD ConfigAdd)
{
        V850EME2_REGW(PHBMR_PCI_CONFIG_ADD)  = ConfigAdd;
        return V850EME2_REGW(PHBMR_PCI_CONFIG_DATA);
}

////////////////////////////////////////////////////////////////
// 関数名：PCI_ConfigWrite                                    //
//   機能：PCI コンフィギュレーション空間に 32bit値をWriteします。  //
//   引数：ConfigAdd: コンフィギュレーション空間レジスタアドレス     //
//         ConfigData: コンフィギュレーション空間レジスタデータ     //
// 戻り値：無し                                               //
////////////////////////////////////////////////////////////////
void PCI_ConfigWrite(UWORD ConfigAdd, UWORD ConfigData)
{
        V850EME2_REGW(PHBMR_PCI_CONFIG_ADD)  = ConfigAdd;
        V850EME2_REGW(PHBMR_PCI_CONFIG_DATA) = ConfigData;

        return;
}

////////////////////////////////////////////////////////////////////////
// 関数名：PCI_Config_BaseAddressInit                                 //
//   機能：コンフィギュレーション空間のベースアドレスを設定します。   //
//   引数：無し                                                       //
// 戻り値：無し                                                       //
//   詳細：AD30信号にIDSEL接続されたPCIデバイスのコンフィギュレーション //
//         空間のオフセット10H～24Hのベースアドレスレジスタを下記のよう //
//         に設定します。                                             //
//                                                                    //
//      ATA Command Register Base Address (10H)      :  0C80_0000H    //
//      ATA Control Register Base Address (14H)      :  0C80_0008H    //
//      Bus Master Control Register Base Address(18H) :  0C80_0010H   //
//                                                                    //
////////////////////////////////////////////////////////////////////////
void PCI_Config_BaseAddressInit(void)
{
   UWORD ConfigAddress;
   UWORD ConfigData;

   ////////////////////////////////////
   // ATA Command Register Base Address //
   ////////////////////////////////////
```

```
    ConfigAddress = 0x40000010;
    // bit 31-11 : IDSEL 指定 = 01000000000000000000b
    //             AD30 に接続されたPCIデバイスを選択
    // bit 10-08 : 機能番号 = 00b
    // bit 07-02 : レジスタ番号 = 4 (000100b),
    //             -> ATA Command Register Base Address
    //                (本アプリケーションで使用するPCI-IDE ASICボードの場合)
    // bit 01-00 : 00b (固定)

    PCI_ConfigWrite(ConfigAddress, 0x0C800000);

    ////////////////////////////////////
    // ATA Control Register Base Address //
    ////////////////////////////////////
    ConfigAddress = 0x40000014;
    // bit 31-11 : IDSEL 指定 = 01000000000000000000b
    //             AD30 に接続されたPCIデバイスを選択
    // bit 10-08 : 機能番号 = 00b
    // bit 07-02 : レジスタ番号 = 5 (000101b),
    //             -> ATA Control Register Base Address
    //                (本アプリケーションで使用するPCI-IDE ASICボードの場合)
    // bit 01-00 : 00b (固定)

    PCI_ConfigWrite(ConfigAddress, 0x0C800008);

    ////////////////////////////////////////////
    // Bus Master Control Register Base Address //
    ////////////////////////////////////////////
    ConfigAddress = 0x40000018;
    // bit 31-11 : IDSEL 指定 = 01000000000000000000b
    //             AD30 に接続されたPCIデバイスを選択
    // bit 10-08 : 機能番号 = 00b
    // bit 07-02 : レジスタ番号 = 6 (000110b)
    //             -> Bus Master Control Register Base Address
    //                (本アプリケーションで使用するPCI-IDE ASICボードの場合)
    // bit 01-00 : 00b (固定)

    PCI_ConfigWrite(ConfigAddress, 0x0C800010);

    return;
}
```

```
//////////////////////////////////////////////////////////////////
// 関数名：main                                                 //
//   機能：コンフィギュレーション空間のベースアドレスを設定します。 //
//   引数：無し                                                 //
// 戻り値：0 :正常終了                                          //
//////////////////////////////////////////////////////////////////
int main(void)
{
    // PCI Host Bridge Macro を初期化します。
    PCI_HBM_Init();

    //  コンフィギュレーション空間のベースアドレスを設定します。
    PCI_Config_BaseAddressInit();

    return 0;
}
```

5. 5. 5　IDE HDDアクセス・サンプル・プログラム・リスト

```
//////////////////////////////////////////////////////////////////////
//  IDE HDD アクセスサンプル                                        //
//    概要： 評価ボードのPCIスロットに接続した PCI-IDE ASIC ボード  //
//          を介しATAデバイスであるHDDにATAコマンドを発行します。  //
//          発行するATAコマンドは以下の通りです。                  //
//                                                                  //
//           IDLE IMMEDIATE, IDENTIFY DEVICE, SET FEATURE,          //
//           READ SECTOR(S), WRITE SECTOR(S), READ DMA, WRITE DMA   //
//                                                                  //
//          ATAコマンドの実行は、まず デバイスセレクションプロト    //
//          コルを行うことで、Master Device と Slave Device のど    //
//          ちらにコマンドを発行するかを決定します。そして、各ATA   //
//          コマンドが対応する転送プロトコルを用いて、ATAコマンド   //
//          の発行とデータ転送を行います。転送プロトコルには、      //
//          PIO datain 転送、PIO dataout 転送、PIO nondata 転送、   //
//          DMA転送の４つがあります。                               //
//                                                                  //
//          本サンプルプログラムでは、デバイスセレクションプロト    //
//          コルと４つの転送プロトコルをそれぞれ関数として実装し    //
//          ています。そして、各ATA コマンドを処理する関数から、    //
//          対応する転送プロトコルの関数を呼び出しています。        //
//                                                                  //
//////////////////////////////////////////////////////////////////////

/////////////////////////////////////////
//  PCI-IDE ASIC ボード レジスタ アドレス定義  //
/////////////////////////////////////////

//////////////////////
// IDE Command Area //
//////////////////////
#define IDEREG_DATA                     ((VUWORD*)(BASE_ADDRESS_PCI_IO + 0x00))
#define IDEREG_ERROR                    ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x01))
#define IDEREG_ERROR_ERR_BIT            (0x01)
#define IDEREG_FEATURES                 ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x01))
#define IDEREG_SECTOR_COUNT             ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x02))
#define IDEREG_SECTOR_NUMBER            ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x03))
#define IDEREG_CYLINDER_LOW             ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x04))
#define IDEREG_CYLINDER_HIGH            ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x05))
#define IDEREG_DEVICE_HEAD              ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x06))
#define IDEREG_STATUS                   ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x07))
```

```
#define IDEREG_COMMAND                  ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x07))

//////////////////////
// IDE Control Area //
//////////////////////
#define IDEREG_ALTERNATE_STATUS         ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x0E))
#define IDEREG_DEVICE_CONTROL           ((VUBYTE*)(BASE_ADDRESS_PCI_IO + 0x0E))

//////////////////////////
// Bus Master I/O Area //
//////////////////////////
#define IDEREG_BUSMASTER_START_STOP     ((VUWORD*)(BASE_ADDRESS_PCI_IO + 0x10))
#define IDEREG_DSCTBL_START_ADDRESS     ((VUWORD*)(BASE_ADDRESS_PCI_IO + 0x14))
#define IDEREG_INTERRUPT_CONTROL        ((VUWORD*)(BASE_ADDRESS_PCI_IO + 0x18))

//////////////////
// エラーコード定義 //
//////////////////
#define STATUS_SUCCESS                   0
#define STATUS_TIMEOUT_BSY0_DRQ0         1
#define STATUS_TIMEOUT_DEVICE_SELECTION  1
#define STATUS_TIMEOUT_DRDY1             2
#define STATUS_TIMEOUT_BSY0              3
#define STATUS_TIMEOUT_INTRQ             4
#define STATUS_TIMEOUT_BMEND             5
#define STATUS_IDE_ERROR(IDE_ERROR_REG)  (0x10000000 | (UWORD)(IDE_ERROR_REG))

////////////////////////////////
// 転送モード・タイミング設定値定義 //
////////////////////////////////

// 下記、転送モード・タイミング設定値の詳細はIDEの仕様書などを参照ください。

// Set_Transfer_mode()でSET_FEATURESコマンドに渡す設定値です。
#define PIO_MODE0       0x08
#define UDMA_MODE0      0x40

// タイミングレジスタの設定値(IDE の動作クロックが33MHzの場合)
#define IDE_PIO_TIMING_IDE33MHz_MODE0    (0x00020906)
#define IDE_UDMA_TIMING1_IDE33MHz_MODE0  (0x00000202)
#define IDE_UDMA_TIMING2_IDE33MHz_MODE0  (0x00000005)
```

```
////////////////
//  構造体宣言  //
////////////////

//////////////////////////
// ATAコマンド発行用 構造体 //
//////////////////////////
typedef struct{
    UBYTE features;             // Featuresレジスタ
    UBYTE sector_count;         // Sector Countレジスタ
    UBYTE sector_number;        // Sector Numberレジスタ
    UBYTE cylinder_low;         // Cylinder Lowレジスタ
    UBYTE cylinder_high;        // Cylinder Highレジスタ
    UBYTE device_head;          // Device/Headレジスタ
    UBYTE command;              // Commandレジスタ
} ATA_COMMAND;

////////////////////////////////////
// UltraDMA 転送用ディスクリプタテーブル //
////////////////////////////////////
typedef struct{
    UWORD transfer_address;     // 転送アドレス
    UWORD transfer_byte;        // 転送バイト数
    UWORD next_table_address;   // 次テーブルアドレス
} DISCRIPTOR_TABLE;

//////////////
// 初期化関数 //
//////////////

/////////////////////////////////////////////////////////////////
// 関数名：PCI_Config_ModeInit                                  //
//   機能：PCI-IDE ASIC ボードの初期設定を行います。             //
//   引数：無し                                                //
// 戻り値：無し                                                //
//   詳細：割り込みやエラーの扱い、暗号化などの設定を行い、最後に  //
//         IDEバスリセットを行います。                          //
//                                                             //
/////////////////////////////////////////////////////////////////
void PCI_Config_ModeInit(void)
{
    UWORD ConfigAddress;
    UWORD ConfigData;
```

```
////////////////////
// PCI 各機能の設定 //
////////////////////
ConfigAddress = 0x40000004;
// bit 31-11 : IDSEL 指定 = 01000000000000000000b
//             AD30 に接続されたPCIデバイスを選択
// bit 10-08 : 機能番号 = 00b
// bit 07-02 : レジスタ番号 = 1 (000001b)
//             -> Status / Command
// bit 01-00 : 00b (固定)

ConfigData = 0x02000145;
// bit 26-25 : DEVSEL timing = 01b (medium固定)
// bit     8 : SERR Enable = 1b : pci_serrを出力します。
// bit     6 : Parity Error Response = 1b : Parity Error 検出時に
//             pci_serr を出力します。
// bit     2 : Bus Master = 1b : PCI Bus Master 転送を許可
// bit     0 : IO Space = 1b : PCI-IDE ASIC ボードへのIOアクセス許可

PCI_ConfigWrite(ConfigAddress, ConfigData);

////////////////////
// DES を無効に設定 //
////////////////////
ConfigAddress = 0x40000058;
// bit 31-11 : IDSEL 指定 = 01000000000000000000b
//             AD30 に接続されたPCIデバイスを選択
// bit 10-08 : 機能番号 = 00b
// bit 07-02 : レジスタ番号 = 22 (010110b),
//             -> IDE Bus Master Control
//                (本アプリケーションで使用するPCI-IDE ASICボードの場合)
// bit 01-00 : 00b (固定)

// IDE Bus Master Control
//   DES を無効にします( bit16 des_on を 0 に設定)
ConfigData = PCI_ConfigRead(ConfigAddress);
PCI_ConfigWrite(ConfigAddress, ConfigData & 0xFFFEFFFF);

////////////////////////////////////
// Interrupt Control レジスタの設定 //
////////////////////////////////////
*IDEREG_INTERRUPT_CONTROL &= 0xFFFCFFFF;
```

```
    // bit    17 : PCI Bus Master End Interrupt Mask = 0b (割り込み許可)
    // bit    16 : PCI I/F Interrupt Mask = 0b (割り込み許可)

    ////////////////////////////////
    // Device Command レジスタの設定 //
    ////////////////////////////////
    *IDEREG_DEVICE_CONTROL = 0x00;
    // bit     2 : nIEN = 0b (INTRQ信号を有効に設定)

    ////////////////////
    // IDE Bus リセット //
    ////////////////////
    ConfigAddress = 0x40000044;
    // bit 31-11 : IDSEL 指定 = 01000000000000000000b
    //             AD30 に接続されたPCIデバイスを選択
    // bit 10-08 : 機能番号 = 00b
    // bit 07-02 : レジスタ番号 = 17 (010001b),
    //             -> IDE Reset Register
    //                (本アプリケーションで使用するPCI-IDE ASICボードの場合)
    // bit 01-00 : 00b (固定)

    ConfigData = 0x00000001;
    // bit     0 : IDE I/F RESET Port   = 1b :
    //             IDE I/F 上に出力されるIDE RESETX信号をH出力します。

    PCI_ConfigWrite(ConfigAddress, ConfigData);

    return;
}

//////////////////////
// 転送モード設定関数 //
//////////////////////

/////////////////////////////////////////////////////////////////////////
// 関数名：Set_Transfer_Mode                                           //
//   機能：転送モードの設定                                            //
//   引数：dev_num : デバイス選択（0:Master/1:Slave）                  //
//         mode : 転送モード                                           //
// 戻り値：                                                            //
//   STATUS_SUCCESS : 正常終了                                         //
//   STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了      //
//   STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了               //
```

```
//    STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了                    //
//    STATUS_IDE_ERROR : コマンド実行後エラー終了                           //
//                                                                          //
//////////////////////////////////////////////////////////////////////////
int Set_Transfer_Mode(int dev_num, UBYTE mode)
{
    status = ATA_Set_Features(dev_num, 0x03, mode);
    return status;
}

//////////////////////////////////////////////////////
// 関数名：Set_PIO_Timing                           //
//   機能：PIO Timing レジスタの設定                //
//   引数：pio_timing : PIO Timingレジスタに設定する値 //
// 戻り値：無し                                     //
//                                                  //
//////////////////////////////////////////////////////
void Set_PIO_Timing(UWORD pio_timing)
{
    UWORD ConfigAddress;
    UWORD ConfigData;

    ConfigAddress = 0x40000048;
    // bit 31-11 : IDSEL 指定 = 01000000000000000000b
    //             AD30 に接続されたPCIデバイスを選択
    // bit 10-08 : 機能番号 = 00b
    // bit 07-02 : レジスタ番号 = 18 (010010b)
    //             -> PIO Timing (本アプリケーションで使用するPCI-IDE ASICボードの場合)
    // bit 01-00 : 00b (固定)

    PCI_ConfigWrite( ConfigAddress, pio_timing );

    return;
}

//////////////////////////////////////////////////////////////
// 関数名：Set_UDMA_Timing                                  //
//   機能：UltraDMA Timing1, 2レジスタの設定                //
//   引数：udma_timing1 :UltraDMA Timing1レジスタに設定する値 //
//         udma_timing2 :UltraDMA Timing2レジスタに設定する値 //
// 戻り値：無し                                             //
//                                                          //
//////////////////////////////////////////////////////////////
```

```
void Set_UDMA_Timing(UWORD udma_timing1、UWORD udma_timing2)
{
   UWORD ConfigAddress;
   UWORD ConfigData;

   ConfigAddress = 0x4000004C;
   // bit 31-11 : IDSEL 指定 = 01000000000000000000b
   //             AD30 に接続されたPCIデバイスを選択
   // bit 10-08 : 機能番号 = 00b
   // bit 07-02 : レジスタ番号 = 19 (010011b)
   //             -> UltraDMA Timing1
   //                (本アプリケーションで使用するPCI-IDE ASICボードの場合)
   // bit 01-00 : 00b (固定)

   PCI_ConfigWrite( ConfigAddress, udma_timing11 );

   ConfigAddress = 0x40000050;
   // bit 31-11 : IDSEL 指定 = 01000000000000000000b
   //             AD30 に接続されたPCIデバイスを選択
   // bit 10-08 : 機能番号 = 00b
   // bit 07-02 : レジスタ番号 = 20 (010100b)
   //             ->  UltraDMA Timing2
   //                (本アプリケーションで使用するPCI-IDE ASICボードの場合)
   // bit 01-00 : 00b (固定)

   PCI_ConfigWrite( ConfigAddress, udma_timing2 );

   return;
}

////////////////////////
// ATA コマンド 実装関数 //
////////////////////////

//////////////////////////////////////////////////////////////////////////
// 関数名：ATA_Set_Features                                             //
//   機能：SET FEATURES コマンド(Protocol:ND, Command:EFh)を実行します。  //
//   引数：dev_num : デバイス選択（0:Master/1:Slave）                    //
// 戻り値：                                                             //
//   STATUS_SUCCESS : 正常終了                                          //
//   STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了       //
//   STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了                //
//   STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了                 //
```

```
//    STATUS_IDE_ERROR : コマンド実行後エラー終了                              //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
int ATA_Set_Features(int dev_num, int sub_cmd, int mode)
{
    int status;
    ATA_COMMAND ac;

    ac.features     = sub_cmd;          // Featuresレジスタ
    ac.sector_count  = mode;            // SectorCountレジスタ
    ac.sector_number = 0x00;            // SectorNumberレジスタ
    ac.cylinder_low  = 0x00;            // CylinderLowレジスタ
    ac.cylinder_high = 0x00;            // CylinderHighレジスタ
    ac.device_head   = dev_num<<4;      // Device/Headレジスタ
    ac.command      = 0xEF;             // Commandレジスタ

    status = ATA_PIO_nondata(&ac);
    return status;
}

////////////////////////////////////////////////////////////////////////////////
// 関数名：ATA_Idle_Immediate                                                 //
//   機能：IDLE IMMEDIATEコマンド(Protocol:ND, Command:E1h)を実行します。      //
//   引数：dev_num : デバイス選択（0:Master/1:Slave）                         //
// 戻り値：                                                                   //
//    STATUS_SUCCESS : 正常終了                                               //
//    STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了            //
//    STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了                     //
//    STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了                      //
//    STATUS_IDE_ERROR : コマンド実行後エラー終了                             //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////
int ATA_Idle_Immediate(int dev_num)
{
    ATA_COMMAND ac;

    ac.features         = 0x00;         // Featuresレジスタ
    ac.sector_count     = 0x00;         // SectorCountレジスタ
    ac.sector_number    = 0x00;         // SectorNumberレジスタ
    ac.cylinder_low     = 0x00;         // CylinderLowレジスタ
    ac.cylinder_high    = 0x00;         // CylinderHighレジスタ
    ac.device_head      = dev_num<<4;   // Device/Headレジスタ
    ac.command          = 0xE1;         // Commandレジスタ
```

```
    status = ATA_PIO_nondata(&ac);

    return status;
}

//////////////////////////////////////////////////////////////////////
// 関数名：ATA_Identify_Device                                      //
//   機能：IDENTIFY DEVICEコマンド(Protocol:PI, Command:ECh)を実行します。  //
//   引数：dev_num : デバイス選択 (0:Master/1:Slave)                  //
//         buff : Bufferのポインタ                                    //
// 戻り値：                                                         //
//   STATUS_SUCCESS : 正常終了                                      //
//   STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了    //
//   STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了             //
//   STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了              //
//   STATUS_IDE_ERROR : コマンド実行後エラー終了                     //
//                                                                  //
//////////////////////////////////////////////////////////////////////
int ATA_Identify_Device(int dev_num, void *buff)
{
    ATA_COMMAND ac;
    int status;

    ac.features          = 0x00; // Featuresレジスタ
    ac.sector_count      = 0x00; // SectorCountレジスタ
    ac.sector_number     = 0x00; // SectorNumberレジスタ
    ac.cyliner_low       = 0x00; // CylinderLowレジスタ
    ac.cylinder_high     = 0x00; // CylinderHighレジスタ

    ac.dev_head          = dev_num << 4; // Device/Headレジスタ
    ac.command           = 0xEC;         // Commandレジスタ

    status = ATA_PIO_datain(&ac, 1, buff);

    return status;
}


//////////////////////////////////////////////////////////////////////
// 関数名：ATA_Read_Sector                                          //
//   機能：READ SECTOR(S)コマンド(Protocol:PI, Command:20h)を実行します。  //
//   引数：dev_num : デバイス選択（0:Master/1:Slave）                //
```

```
//           lba : LBA                                                    //
//       sec_cnt : セクタ数                                               //
//          buff : Bufferのポインタ                                       //
// 戻り値：                                                               //
//   STATUS_SUCCESS : 正常終了                                            //
//   STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了         //
//   STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了                  //
//   STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了                   //
//   STATUS_IDE_ERROR : コマンド実行後エラー終了                          //
//                                                                        //
////////////////////////////////////////////////////////////////////////////
int ATA_Read_Sector(int dev_num, UWORD lba, UHWORD sec_cnt, void *buff)
{
   int status;
   ATA_COMMAND ac;

   ac.features         = 0x00;                                  // Featuresレジスタ
   ac.sector_count     = sector_count;                          // SectorCountレジスタ
   ac.sector_number    = (lba & 0xFF);                          // SectorNumberレジスタ
   ac.cylinder_low     = (lba>>8  & 0xFF);                      // CylinderLowレジスタ
   ac.cylinder_high    = (lba>>16 & 0xFF);                      // CylinderHighレジスタ
   ac.device_head      = 0x40|(dev_num<<4)|(lba>>24 & 0x0F);    // Device/Headレジスタ
   ac.command          = 0x20;                                  // Commandレジスタ

   status = ATA_PIO_datain(&ac, sec_cnt, buff);
   return status;
}

////////////////////////////////////////////////////////////////////////////
// 関数名：ATA_Write_Sector                                               //
//   機能：WRITE SECTOR(S)コマンド(Protocol:PO, Command:30h)を実行します  //
//   引数：dev_num : デバイス選択（0:Master/1:Slave）                     //
//           lba : LBA                                                    //
//       sec_cnt : セクタ数                                               //
//          buff : Bufferのポインタ                                       //
// 戻り値：                                                               //
//   STATUS_SUCCESS : 正常終了                                            //
//   STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了         //
//   STATUS_TIMEOUT_BSY0_DRQ0 : BSY=0,DRQ=0 タイムアウトエラー終了        //
//   STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了                  //
//   STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了                   //
//   STATUS_IDE_ERROR : コマンド実行後エラー終了                          //
//                                                                        //
```

```
//////////////////////////////////////////////////////////////////////
int ATA_Write_Sector(int dev_num, UWORD lba, UHWORD sec_cnt, void *buff)
{
    int status;
    ATA_COMMAND ac;

    ac.features         = 0x00;                                  // Featuresレジスタ
    ac.sector_count     = sector_count;                          // SectorCountレジスタ
    ac.sector_number    = (lba & 0xFF);                          // SectorNumberレジスタ
    ac.cylinder_low     = (lba>>8  & 0xFF);                      // CylinderLowレジスタ
    ac.cylinder_high    = (lba>>16 & 0xFF);                      // CylinderHighレジスタ
    ac.device_head      = 0x40|(dev_num<<4)|(lba>>24 & 0x0F);    // Device/Headレジスタ
    ac.command          = 0x30;                                  // Commandレジスタ

    status = ATA_PIO_dataout(&ac, sec_cnt, buff);
    return status;
}

//////////////////////////////////////////////////////////////////////
// 関数名：ATA_Read_DMA                                            //
//   機能：READ DMA コマンド(Protocol:DM, Command:C8h)を実行します。  //
//   引数：dev_num : デバイス選択（0:Master/1:Slave）                //
//           lba : LBA                                              //
//       sec_cnt : セクタ数                                         //
// 戻り値：                                                        //
//   STATUS_SUCCESS : 正常終了                                      //
//   STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了   //
//   STATUS_TIMEOUT_BSY0_DRQ0 : BSY=0,DRQ=0 タイムアウトエラー終了  //
//   STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了            //
//   STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了             //
//   STATUS_TIMEOUT_BMEND : BMタイムアウトエラー終了                //
//   STATUS_IDE_ERROR : コマンド実行後エラー終了                    //
//                                                                  //
//////////////////////////////////////////////////////////////////////
int ATA_Read_DMA(int dev_num, UWORD lba, UHWORD sec_cnt)
{
    int status;
    ATA_COMMAND ac;

    ac.features         = 0x00;                                  // Featuresレジスタ
    ac.sector_count     = sector_count;                          // SectorCountレジスタ
    ac.sector_number    = (lba & 0xFF);                          // SectorNumberレジスタ
    ac.cylinder_low     = (lba>>8  & 0xFF);                      // CylinderLowレジスタ
```

```
    ac.cylinder_high    = (lba>>16 & 0xFF);                     // CylinderHighレジスタ
    ac.device_head      = 0x40|(dev_num<<4)|(lba>>24 & 0x0F);   // Device/Headレジスタ
    ac.command          = 0xC8;                                 // Commandレジスタ

    status = ATA_DMA(&ac);
    return status;
}

////////////////////////////////////////////////////////////////////////
// 関数名：ATA_Write_DMA                                              //
//   機能：WRITE DMA コマンド(Protocol:DM, Command:CAh)を実行します。  //
//   引数：dev_num : デバイス選択（0:Master/1:Slave）                 //
//           lba : LBA                                                //
//       sec_cnt : セクタ数                                           //
// 戻り値：                                                           //
//   STATUS_SUCCESS : 正常終了                                        //
//   STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了     //
//   STATUS_TIMEOUT_BSY0_DRQ0 : BSY=0,DRQ=0 タイムアウトエラー終了    //
//   STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了              //
//   STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了               //
//   STATUS_TIMEOUT_BMEND : BMタイムアウトエラー終了                  //
//   STATUS_IDE_ERROR : コマンド実行後エラー終了                      //
//                                                                    //
////////////////////////////////////////////////////////////////////////
int ATA_Write_DMA(int dev_num, UWORD lba, UHWORD sec_cnt)
{
    int status;
    ATA_COMMAND ac;

    ac.features         = 0x00;                                 // Featuresレジスタ
    ac.sector_count     = sector_count;                         // SectorCountレジスタ
    ac.sector_number    = (lba & 0xFF);                         // SectorNumberレジスタ
    ac.cylinder_low     = (lba>>8  & 0xFF);                     // CylinderLowレジスタ
    ac.cylinder_high    = (lba>>16 & 0xFF);                     // CylinderHighレジスタ
    ac.device_head      = 0x40|(dev_num<<4)|(lba>>24 & 0x0F);   // Device/Headレジスタ
    ac.command          = 0xCA;                                 // Commandレジスタ

    status = ATA_DMA(&ac);
    return status;
}

//////////////////////
// プロトコル 実装関数 //
```

```
//////////////////////

////////////////////////////////////////////////////////////////////////
// 関数名：ATA_Device_Selection                                        //
//   機能：デバイスセレクションプロトコルを実行します。                  //
//   引数：  dev_num : (0:Master / 1:Slave)                            //
// 戻り値：                                                           //
//    STATUS_SUCCESS : 正常終了                                       //
//    STATUS_TIMEOUT_BSY0_DRQ0 : BSY=0,DRQ=0 タイムアウトエラー終了   //
//                                                                    //
////////////////////////////////////////////////////////////////////////
int ATA_Device_Selection(int dev_num)
{
    int status;

    status = Wait_IDE_BSY0_DRQ0();              // BSY=0, DRQ=0 までウェイト
    if ( status != 0 ) {
        return STATUS_TIMEOUT_BSY0_DRQ0;        // タイムアウト終了
    }

    *IDEREG_DEVICE_HEAD = dev_num << 4;         // デバイス選択
    wait(TIMER400ns);                           // 400nsウェイト

    status = Wait_IDE_BSY0_DRQ0();              // BSY=0, DRQ=0 までウェイト
    if ( status != 0 ) {
        return STATUS_TIMEOUT_BSY0_DRQ0;        // タイムアウト終了
    }

    return STATUS_SUCCESS;                      // 正常終了
}

////////////////////////////////////////////////////////////////////////
// 関数名：ATA_PIO_datain                                              //
//   機能：PIO data in commandプロトコルを実行します。                 //
//   引数：   atacom : ATA_COMMAND構造体のポインタ                     //
//      sector_count : セクタ数                                       //
//             buff : Bufferのポインタ                                //
// 戻り値：                                                           //
//    STATUS_SUCCESS : 正常終了                                       //
//    STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了    //
//    STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了             //
//    STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了              //
//    STATUS_IDE_ERROR : コマンド実行後エラー終了                     //
```

```
//                                                                      //
//////////////////////////////////////////////////////////////////////////
int ATA_PIO_datain(ATA_COMMAND *atacom, UHWORD sector_count, void *buff)
{
    UBYTE dev, idestat;
    UWORD *buffp;
    int i, j, status;

    buffp = (UWORD*)buff;
    dev = ( atacom->device_head >> 4 ) & 1;

    status = ATA_Device_Selection(dev);                 // DEVICE SELECTION
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DEVICE_SELECTION;         // DEVICE SELECTION タイムアウト
    }

    *IDEREG_FEATURES     = atacom->features;            // Featuresレジスタ
    *IDEREG_SECTOR_COUNT  = atacom->sector_count;       // SectorCountレジスタ
    *IDEREG_SECTOR_NUMBER = atacom->sector_number;      // SectorNumberレジスタ
    *IDEREG_CYLINDER_LOW  = atacom->cylinder_low;       // CylinderLowレジスタ
    *IDEREG_CYLINDER_HIGH = atacom->cylinder_high;      // CylinderHighレジスタ

    status = Wait_IDE_DRDY1();                          // DRDY=1までループ
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DRDY1                     // DRDY1タイムアウト
    }

    *IDEREG_COMMAND = atacom->command;                  // Commandレジスタ
    wait(TIMER400ns);                                   // 400nsウェイト

    for ( i=0; i<sector_count; i++ ) {
        status = Wait_IDE_INTRQ();                      // INTRQアサートを待つ
        if ( status != 0 ) {
            return STATUS_TIMEOUT_INTRQ;                // INTRQタイムアウトエラー
        }

        idestat = *IDEREG_STATUS;                  // Statusレジスタリード(INTRQクリア)

        for ( j=0; j<128; j++ ) {                       // データの読み出し
            *buffp = *IDEREG_DATA;
            buffp++;
        }
    }
```

```
    idestat = *IDEREG_ALTERNATE_STATUS;                 // Alt Statusレジスタ空リード
    idestat = *IDEREG_STATUS;                           // Statusレジスタリード

    if ( idestat & IDEREG_ERROR_ERR_BIT ) {
        return STATUS_IDE_ERROR(*IDEREG_ERROR);         // エラー終了(コマンド実行後)
    }
    return STATUS_SUCCESS;                              // 正常終了
}

////////////////////////////////////////////////////////////////////////
// 関数名：ATA_PIO_dataout                                            //
//   機能：PIO data out commandプロトコルを実行します。                //
//   引数：  atacom : ATA_COMMAND構造体のポインタ                      //
//      sector_count : セクタ数                                       //
//             buff : Bufferのポインタ                                //
// 戻り値：                                                           //
//    STATUS_SUCCESS : 正常終了                                       //
//    STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了    //
//    STATUS_TIMEOUT_BSY0_DRQ0 : BSY=0,DRQ=0 タイムアウトエラー終了   //
//    STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了             //
//    STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了              //
//    STATUS_IDE_ERROR : コマンド実行後エラー終了                     //
//                                                                    //
////////////////////////////////////////////////////////////////////////
int ATA_PIO_dataout(ATA_COMMAND *atacom, UHWORD sector_count, void *buff)
{
    UBYTE dev, idestat;
    UWORD *buffp;
    int i, j, status;

    buffp = (UWORD*)buff;
    dev = ( atacom->device_head >> 4 ) & 1;

    status = ATA_Device_Selection(dev);                 // DEVICE SELECTION
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DEVICE_SELECTION;         // DEVICE SELECTION タイムアウト
    }

    *IDEREG_FEATURES     = atacom->features;            // Featuresレジスタ
    *IDEREG_SECTOR_COUNT  = atacom->sector_count;       // SectorCountレジスタ
    *IDEREG_SECTOR_NUMBER = atacom->sector_number;      // SectorNumberレジスタ
    *IDEREG_CYLINDER_LOW  = atacom->cylinder_low;       // CylinderLowレジスタ
```

```
    *IDEREG_CYLINDER_HIGH = atacom->cylinder_high;      // CylinderHighレジスタ

    status = Wait_IDE_DRDY1();                          // DRDY=1までループ
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DRDY1;                    // DRDYタイムアウト
    }

    *IDEREG_COMMAND = atacom->command;                  // Commandレジスタ
    wait(TIMER400ns);                                   // 400nsウェイト

    status = Wait_IDE_BSY0_DRQ0();                      // BSY=0, DRQ=0 までウェイト
    if ( status != 0 ) {
        return STATUS_TIMEOUT_BSY0_DRQ0;                // BSYタイムアウトエラー終了
    }

    for ( i=0; i<sector_count; i++ ) {
        for ( j=0; j<128; j++ ) {                       // データの書き込み
                        *IDEREG_DATA = *buffp;
                        buffp++;
                }
        status = Wait_IDE_INTRQ();                      //  INTRQアサートを待つ
        if ( status != 0 ) {
            return STATUS_TIMEOUT_INTRQ;                //  INTRQタイムアウトエラー
        }
        idestat = *IDEREG_STATUS;               //  Statusレジスタリード(INTRQクリア)
    }

    if ( idestat & IDEREG_ERROR_ERR_BIT ) {
        return STATUS_IDE_ERROR(*IDEREG_ERROR);         // エラー終了(コマンド実行後)
    }
    return STATUS_SUCCESS;                              // 正常終了
}

//////////////////////////////////////////////////////////////////////
// 関数名：ATA_PIO_nondata                                          //
//   機能：PIO non data commandプロトコルを実行します。             //
//   引数：  atacom : ATA_COMMAND構造体のポインタ                   //
// 戻り値：                                                         //
//    STATUS_SUCCESS : 正常終了                                     //
//    STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了  //
//    STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了           //
//    STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了            //
//    STATUS_IDE_ERROR : コマンド実行後エラー終了                   //
```

```
//                                                                      //
//////////////////////////////////////////////////////////////////////////
int ATA_PIO_nondata(ATA_COMMAND *atacom)
{
    int status;
    UBYTE dev, idestat;

    dev = ( atacom->device_head >> 4 ) & 1;
    status = ATA_Device_Selection(dev);                 // DEVICE SELECTION
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DEVICE_SELECTION;         // DEVICE SELECTION タイムアウト
    }

    *IDEREG_FEATURES     = atacom->features;            // Featuresレジスタ
    *IDEREG_SECTOR_COUNT  = atacom->sector_count;       // SectorCountレジスタ
    *IDEREG_SECTOR_NUMBER = atacom->sector_number;      // SectorNumberレジスタ
    *IDEREG_CYLINDER_LOW  = atacom->cylinder_low;       // CylinderLowレジスタ
    *IDEREG_CYLINDER_HIGH = atacom->cylinder_high;      // CylinderHighレジスタ

    status = Wait_IDE_DRDY1();                          // DRDY=1までループ
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DRDY1;                    // DRDYタイムアウト
    }

    *IDEREG_COMMAND = atacom->command;                  // Commandレジスタ

    wait(TIMER400ns);                                   // 400nsウェイト

    status = Wait_IDE_INTRQ();                          // INTRQアサートを待つ
    if ( status != 0 ) {
        return STATUS_TIMEOUT_INTRQ;                    // INTRQタイムアウトエラー
    }

    idestat = *IDEREG_ALT_STATUS;                       // Alt Statusレジスタ空リード
    idestat = *IDEREG_STATUS;                           // Statusレジスタリード

    if ( idestat & IDEREG_ERROR_ERR_BIT ) {
        return STATUS_IDE_ERROR(*IDEREG_ERROR);         // エラー終了(コマンド実行後)
    }

    return STATUS_SUCCESS;                              // 正常終了
}
```

```
//////////////////////////////////////////////////////////////////////
// 関数名：ATA_DMA                                                  //
//   機能：DMA commandプロトコルを実行します。                      //
//   引数：  atacom : ATA_COMMAND構造体のポインタ                   //
// 戻り値：                                                         //
//    STATUS_SUCCESS : 正常終了                                     //
//    STATUS_TIMEOUT_DEVICE_SELECTION : DEIVCE SELECTIONエラー終了  //
//    STATUS_TIMEOUT_BSY0_DRQ0 : BSY=0,DRQ=0 タイムアウトエラー終了 //
//    STATUS_TIMEOUT_DRDY1 : DRDY=1タイムアウトエラー終了           //
//    STATUS_TIMEOUT_INTRQ : INTRQタイムアウトエラー終了            //
//    STATUS_TIMEOUT_BMEND : BMタイムアウトエラー終了               //
//    STATUS_IDE_ERROR : コマンド実行後エラー終了                   //
//                                                                  //
//////////////////////////////////////////////////////////////////////
int ATA_DMA(ATA_COMMAND *atacom)
{
    int status;
    UBYTE dev, idestat;

    dev = ( atacom->device_head >> 4 ) & 1;
    status = ATA_Device_Selection(dev);                 // DEVICE SELECTION
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DEVICE_SELECTION;         // DEVICE SELECTION タイムアウト
    }

    *IDEREG_FEATURES            = atacom->features;             // Featuresレジスタ
    *IDEREG_SECTOR_COUNT        = atacom->sector_count;         // SectorCountレジスタ
    *IDEREG_SECTOR_NUMBER       = atacom->sector_number;        // SectorNumberレジスタ
    *IDEREG_CYL_LOW             = atacom->cylinder_low;         // CylinderLowレジスタ
    *IDEREG_CYL_HIGH            = atacom->cylinder_high;        // CylinderHighレジスタ

    status = Wait_IDE_DRDY1();                  // DRDY=1までループ
    if ( status != 0 ) {
        return STATUS_TIMEOUT_DRDY1;            // DRDYタイムアウト
    }

    *IDEREG_COMMAND = atacom->command;          // Commandレジスタ

    wait(TIMER400ns);                           // 400nsウェイト
    idestat = *IDEREG_ALT_STATUS;               // Alt Statusレジスタ空リード

    *IDEREG_BUSMASTER_START_STOP |= 0x01;       // Bus Master Start
    status = Wait_IDE_BMEND();
```

```
    if ( status != 0 ) {
        return STATUS_TIMEOUT_BMEND;            // BMEND タイムアウトエラー終了
    }

    status = Wait_IDE_INTRQ();                  // INTRQ アサートを待つ
    if ( status != 0 ) {
        return STATUS_TIMEOUT_INTRQ;            // INTRQ タイムアウトエラー終了
    }

    idestat = *IDEREG_ALT_STATUS;               // Alt Statusレジスタ空リード
    idestat = *IDEREG_STATUS;                   // Statusレジスタリード

    if ( idestat & IDEREG_ERROR_ERR_BIT ) {
        return STATUS_IDE_ERROR(*IDEREG_ERROR); // エラー終了(コマンド実行後)
    }
    return STATUS_SUCCESS;                      // 正常終了
}

////////////////////////////////////////////////////////////////
// 関数名：ATA_Soft_Reset                                      //
//   機能：ソフトウェアリセットを行います。                    //
//   引数：無し                                                //
// 戻り値：                                                    //
//    STATUS_SUCCESS : 正常終了                                //
//    STATUS_TIMEOUT_BSY0 : BSY=0 タイムアウトエラー終了       //
//                                                            //
////////////////////////////////////////////////////////////////
int ATA_soft_reset(void)
{
    int status;

    *IDEREG_DEVICE_CONTROL = 0x04;      // リセット実行
    wait(TIMER5ms);                     // 5msウェイト
    *IDEREG_DEVICE_CONTROL = 0x00;      // リセット解除
    wait(TIMER5ms);                     // 5msウェイト

    status = Wait_IDE_BSY0();           // BSY=0までウェイト
    if ( status != 0 ) {
        return STATUS_TIMEOUT_BSY0;     // タイムアウトエラー終了
    }
    return STATUS_SUCCESS;
}
```

```
////////////////////////////////////////////////////////////////////////////////
// 関数名：main                                                                //
//   機能：PCIバス、PCI-IDE ASICボードを介してIDE HDDにアクセスします。        //
//   引数：無し                                                                //
// 戻り値：0 :正常終了                                                         //
//   概要：IDLE IMMEDIATE, IDENTIFY DEVICE, SET FEATURE, READ SECTOR(S),       //
//         WRITE SECTOR(S), READ DMA, WRITE DMA コマンドの発行を行います。     //
//                                                                            //
////////////////////////////////////////////////////////////////////////////////

int main(void)
{
    int status;
    UBYTE wbuff[4096], rbuff[4096];
    DISCRIPTOR_TABLE* dsc_tbl;

    /////////////////
    // システム初期化 //
    /////////////////
    // PCI Host Bridge Macro を初期化します。
    PCI_HBM_Init();

    // PCI-IDE ASIC ボードの初期設定を行います。
    PCI_Config_BaseAddressInit();
    PCI_Config_ModeInit();

    ATA_soft_reset(void);    // ソフトリセット

    ////////////////////////////
    // IDE HDD にATAコマンドを発行 //
    ////////////////////////////

    ////////////////////
    // IDLE IMMEDIATE //
    ////////////////////
        ATA_Idle_Immediate(0);   // IDLE IMMEDIATE コマンドを発行します。

    ////////////////////
    // IDENTIFY DEVICE //
    ////////////////////
    ATA_Identify_Device(        // IDENTIFY DEVICE コマンドを発行します。
       0,                       // Master Device
       buff                     // 結果を格納するバッファ
```

```
);

/////////////////
// PIO 転送準備 //
/////////////////
// SET_FEATURE コマンドを使い、転送モードをPIO 転送Mode0に設定します。
Set_Transfer_Mode(0, PIO_MODE0);

// PCI-IDE ASIC ボードのコンフィギュレーションレジスタのPIO Timing
// レジスタを設定します。
Set_PIO_Timing(IDE_PIO_TIMING_IDE33MHz_MODE0);

// バッファの初期化
InitBuffer(wbuff, 4096);

//////////////
// PIO 転送 //
//////////////
ATA_Write_Sector(    // WRITE SECTOR コマンドを発行します。
    0,               // Master Device
    0,               // LBA 0
    1,               // 1 Sector
    wbuff            // WRITEする内容を格納するバッファ
);

ATA_Read_Sector(     // READ SECTOR コマンドを発行します。
    0,               // Master Device
    0,               // LBA 0
    1,               // 1 Sector
    rbuff            // READ結果を格納するバッファ
);

status = memcmp(wbuff, rbuff, 512);
if ( status != 0 ) {
    printf("Verify Error!: WRITE SECTOR(S), READ SECTOR(S)¥n");
}

//////////////////////
// UltraDMA 転送準備 //
//////////////////////
// SET_FEATURE コマンドを使い、転送モードをUltraDMA転送Mode0に設定します。
Set_Transfer_Mode(0, UDMA_MODE0);
```

```
// PCI-IDE ASIC ボードのコンフィギュレーションレジスタのUltraDMA
// Timing1, UltraDMA Timing2 レジスタを設定します。
Set_UDMA_Timing(IDE_UDMA_TIMING1_IDE33MHz_MODE0, IDE_UDMA_TIMING2_IDE33MHz_MODE0);

//////////////////////////
// PCI->IDE UltraDMA転送 //
//////////////////////////

// UltraDMA転送時にPCI-IDE ASIC ボードが参照する、ディスクリプタテーブルを設定します。
dsc_tbl = (DISCRIPTOR_TABLE*)(BASE_ADDRESS_SDRAM + 0x02000000)
dsc_tbl->transfer_address   = BASE_ADDRESS_SDRAM;
dsc_tbl->transfer_byte      = 0x1000;              // = 4096byte = 8Sector
dsc_tbl->next_table_address = 0x00000001;          // 最終テーブル
*IDEREG_DSCTBL_START_ADDRESS = dsc_tbl;

// UltraDMA転送の転送方向を設定します。
*IDEREG_BUSMASTER_START_STOP &= 0xFFFFFEFF;        // Ultra DMA (PCI->IDE)

// バッファの初期化
InitBuffer((UBYTE*)dsc_tbl->transfer_address, 512*8);

// UltraDMA転送のコマンドを発行します。
ATA_Write_DMA(
   0,               // Master Device
   0,               // LBA 0
   8,               // 8 Sector
);

//////////////////////////
// PCI<-IDE UltraDMA転送 //
//////////////////////////

// UltraDMA転送時にデバイスが参照するディスクリプタテーブルを設定します。
dsc_tbl->transfer_address   = BASE_ADDRESS_SDRAM + 0x01000000;
dsc_tbl->transfer_byte      = 0x1000;              // = 4096byte = 8Sector
dsc_tbl->next_table_address = 0x00000001;          // 最終テーブル
*IDEREG_DSCTBL_START_ADDRESS= dsc_tbl;

// UltraDMA転送の転送方向を設定します。
*IDEREG_BUSMASTER_START_STOP |= 0x00000100;        // Ultra DMA (PCI<-IDE)

// UltraDMA転送のコマンドを発行します。
ATA_Read_DMA(
```

```
        0,              // Master Device
        0,              // LBA 0
        8,              // 8 Sector
    );

    status = memcmp(
              (UBYTE*)(BASE_ADDRESS_SDRAM),
              (UBYTE*)(BASE_ADDRESS_SDRAM+0x01000000),
               512*8);
    if ( status != 0 ) {
        printf("Verify Error!: WRITE DMA, READ DMA¥n");
    }

    return 0;
}
```

図5ー6　IDE_Write_DMA関数について

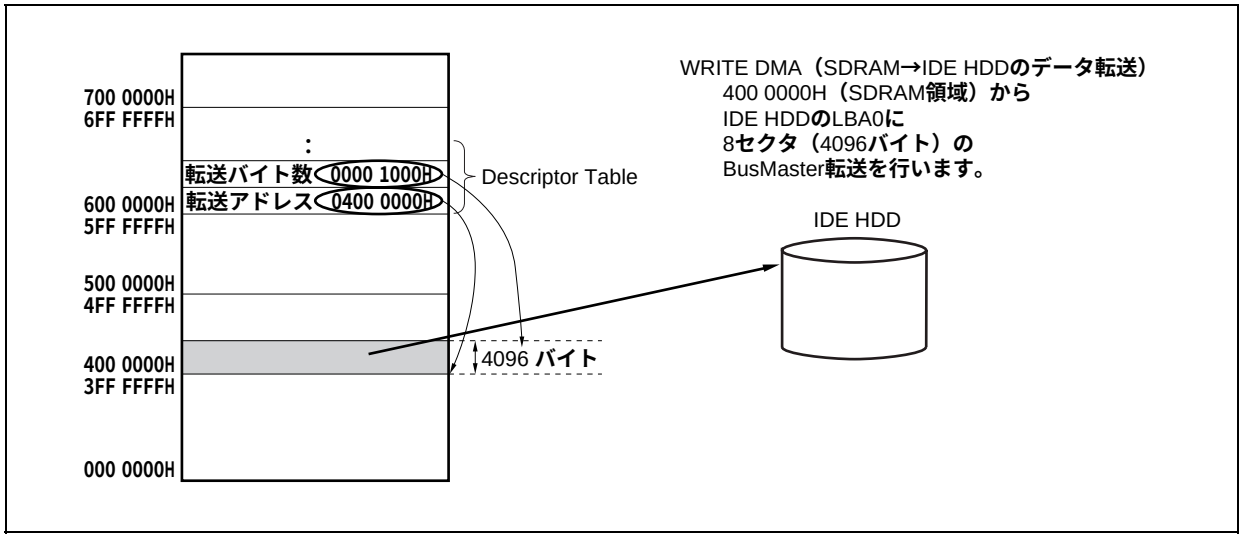
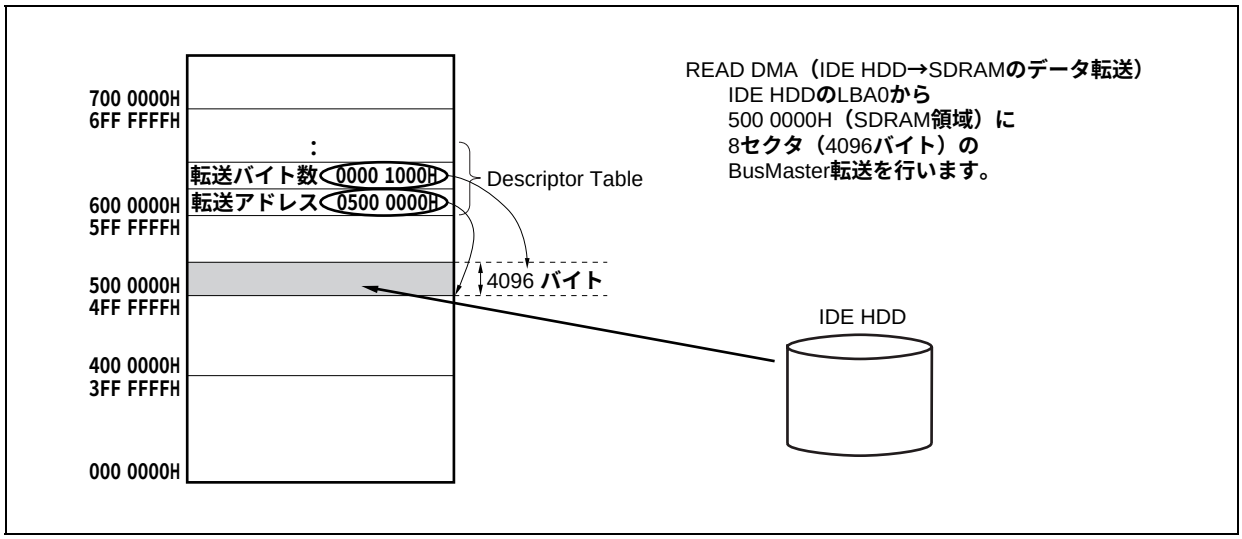


図5ー7　IDE_Read_DMA関数について



〔メ　モ〕

【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。