

---

---

Motion *Responsive Multithreaded*  
*Processor* 仕様書  
第 0 版

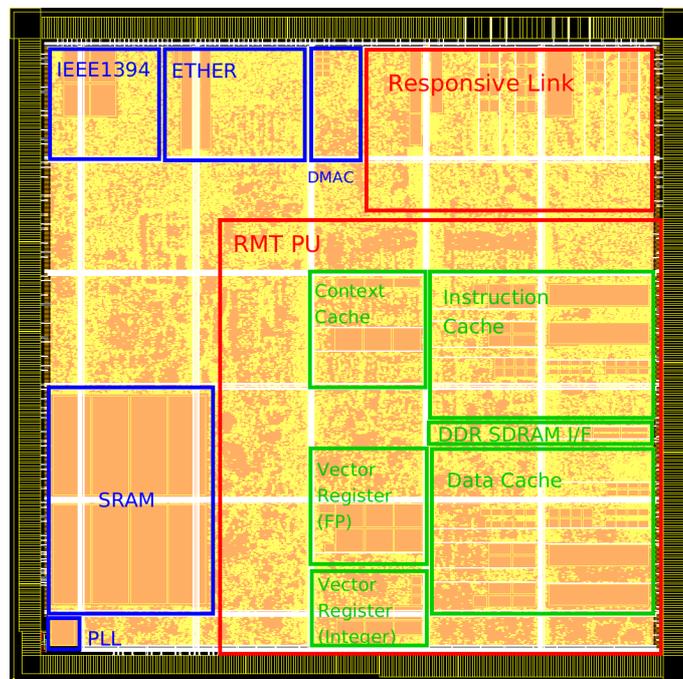
---

---

平成 19 年 5 月 14 日  
慶應義塾大学 理工学部 山崎研究室

---

— NY0020 —



<http://www.ny.ics.keio.ac.jp/research/rmt/>

# 目次

<b>第 1 章 概要</b>	<b>9</b>
1.1 Overview	9
1.2 設計ポリシー	9
1.3 全体構成	10
1.4 Responsive Multithreaded Processing Unit	11
1.4.1 命令発行ユニット	11
1.4.2 命令演算ユニット	13
1.4.3 キャッシュユニット	14
1.5 <i>Responsive Link</i>	15
<b>第 2 章 PIN 配置</b>	<b>17</b>
<b>第 3 章 命令セット</b>	<b>69</b>
3.1 MIPS 互換の命令	69
3.1.1 Load / Store 命令	69
3.1.2 演算命令	86
3.1.3 Jump / 分岐命令	98
3.1.4 浮動小数点命令	108
3.1.5 その他の命令	122
3.2 MIPS 命令と動作の異なる命令	129
3.2.1 演算命令	129
3.2.2 浮動小数点命令	140
3.2.3 その他の命令	143
3.2.4 サポートしていない MIPS II 命令	143
3.3 Responsive Multithreaded Processor 固有の命令	144
3.3.1 Load / Store 命令	145
3.3.2 演算命令	148
3.3.3 転送命令	159
3.3.4 システム制御命令	160
3.3.5 スレッド制御命令	164
3.3.6 SIMD 演算命令	181
3.3.7 同期命令	215
3.3.8 整数ベクトル命令	222
3.3.9 浮動小数点ベクトル命令	320
<b>第 4 章 アドレスデコーダ</b>	<b>375</b>
4.1 レジスタインターフェース	375
4.2 アドレスマップ	377

<b>第 5 章</b>	<b>MMU</b>	<b>379</b>
5.1	TLB エントリ	379
5.2	MMU の制御	383
5.3	MMU が発生させる例外	388
<b>第 6 章</b>	<b>CACHE</b>	<b>389</b>
6.1	キャッシュシステム	389
6.1.1	概要	389
6.1.2	キャッシュ制御	390
6.1.3	victim buffer	390
6.1.4	wait buffer	390
6.1.5	キャッシュのコントロールレジスタ	391
<b>第 7 章</b>	<b>システムレジスタ</b>	<b>395</b>
7.1	レジスタマップ	395
7.1.1	Status Register	398
7.1.2	Thread Table Register	399
7.1.3	Thread ID Register	400
7.1.4	Instruction Counter Register	400
7.1.5	Count Register	401
7.1.6	Compare Register	401
7.1.7	Floating-Point Control Register	401
7.1.8	Issue Mode Register	402
7.1.9	CPU Count Register	403
7.1.10	MMU Register	404
7.1.11	Exception PC Register	404
7.1.12	Exception Cause Register	404
7.1.13	Interruption Wait Register (スレッド毎)	405
7.1.14	External Interruption Level Register (スレッド毎)	405
7.1.15	Interruption Pending Register	406
7.1.16	Interruption Clear Register	406
7.1.17	Exception Base Address Register	406
7.1.18	Event Link In Register	406
7.1.19	Event Link Out Register	406
7.1.20	Instruction Cache Control Register	406
7.1.21	Data Cache Control Register	406
7.1.22	ROM Status	407
7.1.23	EXT Status	407
7.1.24	Multiplexer Arbitor Mode 256bit Bus	407
7.1.25	Multiplexer Arbitor Mode 32bit Bus	408
7.1.26	Multiplexer Watchdog Timer 256bit Bus Enable	408
7.1.27	Multiplexer Watchdog Timer 256bit Bus Mode	408
7.1.28	Multiplexer Watchdog Timer 256bit Bus Reset	408
7.1.29	Multiplexer Watchdog Timer 256bit Bus Count	408
7.1.30	Multiplexer Error Handler State 256bit Bus	408
7.1.31	Multiplexer Error Handler State 32bit Bus	409

7.1.32	Multiplexer Error Handler Instruction Cache . . . . .	409
7.1.33	Multiplexer Error Handler Data Cache . . . . .	409
7.1.34	Multiplexer Error Handler DMAC0 . . . . .	409
7.1.35	Multiplexer Error Handler DMAC1 . . . . .	409
7.1.36	Multiplexer Error Handler DMAC2 . . . . .	409
7.1.37	Multiplexer Error Handler PCI . . . . .	409
7.1.38	Multiplexer Error Handler Bus Interface Unit . . . . .	409
7.1.39	Multiplexer Error Handler MDMAC256 . . . . .	409
7.1.40	Address Decoder Control Register . . . . .	409
7.1.41	Multiplexer Watchdog Timer 32bit Bus Enable . . . . .	409
7.1.42	Multiplexer Watchdog Timer 32bit Bus Mode . . . . .	410
7.1.43	Multiplexer Watchdog Timer 32bit Bus Reset . . . . .	410
7.1.44	Multiplexer Watchdog Timer 32bit Bus Count . . . . .	410
7.1.45	Multiplexer Error Handler MDMAC32 . . . . .	410
7.1.46	Own Status Register . . . . .	410
7.1.47	Own Thread Table Register . . . . .	410
7.1.48	Own Thread ID Register . . . . .	410
7.1.49	Own Instruction Count Register . . . . .	410
7.1.50	Own Count Register . . . . .	410
7.1.51	Own Compare Register . . . . .	410
7.1.52	Own Floating-Point Control Register . . . . .	410
7.1.53	Own Bad Virtual Address Register . . . . .	411
7.1.54	Own Exception PC Register . . . . .	411
7.1.55	Own Exception Cause Register . . . . .	411
7.1.56	Own Interruption Wait Register . . . . .	411
7.1.57	Own External Interruption Level Register . . . . .	411
<b>第 8 章</b>	<b>例外処理</b>	<b>413</b>
8.1	割り込みコントローラ (IRC) . . . . .	413
8.1.1	レジスタマップ . . . . .	413
8.1.2	Trigger Mode Register . . . . .	413
8.1.3	Request Sense Register . . . . .	414
8.1.4	Request Clear Register . . . . .	414
8.1.5	Mask Register . . . . .	414
8.1.6	IRL Latch/Clear . . . . .	415
8.1.7	IRC Mode Register . . . . .	415
8.2	動作/使用方法 . . . . .	415
8.2.1	IRC . . . . .	415
8.2.2	RMT 固有機能 . . . . .	416
8.2.3	例外処理プロセス . . . . .	417
<b>第 9 章</b>	<b>クロックジェネレータ</b>	<b>419</b>
9.1	接続図 . . . . .	419
9.2	制御レジスタ . . . . .	420
9.2.1	Clock Enable . . . . .	420
9.2.2	Soft Reset . . . . .	420

9.2.3	Divider Ratio	421
9.2.4	Clock Synchronization	421
9.2.5	All Reset	422
<b>第 10 章</b>	<b>スレッド制御</b>	<b>423</b>
10.1	スレッドの種類	423
10.2	スレッド制御命令	423
10.2.1	作成・削除	423
10.2.2	状態制御	424
10.2.3	転送	424
10.3	状態遷移	425
<b>第 11 章</b>	<b>同期</b>	<b>427</b>
11.1	共有レジスタ	427
11.2	同期命令	427
<b>第 12 章</b>	<b>Vector Unit</b>	<b>431</b>
12.1	概要	431
12.1.1	Vector Execution Unit	432
12.1.2	命令フォーマット	432
12.2	Reserve/Release 命令	433
12.3	Status Register	435
12.4	複合演算命令	435
<b>第 13 章</b>	<b>Responsive Link</b>	<b>441</b>
13.1	概要	441
13.2	<i>Responsive Link</i> のインタフェース	442
13.3	パケットフォーマット	443
13.3.1	固定長 (64B) のデータパケット	444
13.3.2	固定長 (16B) のイベントパケット	445
13.3.3	優先度による追い越し機構	445
13.4	フレームフォーマット	447
13.5	ルーティング・テーブル	448
13.6	パケットの加減速制御	449
13.7	優先度に従った経路制御	449
13.8	低レベル通信	451
13.8.1	CODEC	451
13.8.2	巡回組織ハミング符号化	451
13.8.3	Bit Stuffing	452
13.8.4	NRZI 符合化	452
13.8.5	セットアップパターン	452
13.8.6	DPLL を用いたビット同期	453
13.8.7	エラーの取扱い	453
13.8.8	通信速度	453
13.9	メモリマップ	454
13.10	レジスタマップ	454

13.10.1	SDRAM モードレジスタ	454
13.10.2	レスポンスリンク速度設定レジスタ	455
13.10.3	レスポンスリンク初期化レジスタ	455
13.10.4	レスポンスリンク割り込みクリアレジスタ	456
13.10.5	レスポンスリンク送信停止割り込みクリアレジスタ	457
13.10.6	レスポンスリンク継続割り込みクリアレジスタ	458
13.10.7	レスポンスリンク致命的エラー割り込みクリアレジスタ	459
13.10.8	レスポンスリンクルーティングテーブル割り込みクリアレジスタ	460
13.10.9	レスポンスリンク SDRAM バスリクエストレジスタ	460
13.10.10	レスポンスリンク SDRAM バスグラントレジスタ	461
13.10.11	レスポンスリンクルーティングテーブルバスリクエストレジスタ	462
13.10.12	レスポンスリンクルーティングテーブルバスグラントレジスタ	463
13.10.13	イベントリンク LRU アドレスレジスタ	464
13.10.14	データリンク LRU アドレスレジスタ	464
13.10.15	レスポンスリンク用割り込みコントローララインエーブルレジスタ	465
13.10.16	イベントリンク用 SDRAM ループカウンタレジスタ	465
13.10.17	データリンク用 SDRAM ループカウンタレジスタ	465
13.10.18	レスポンスリンクスイッチモードレジスタ	466
13.10.19	レスポンスリンク用オフラインレジスタ	466
13.11	DPM (Dual Port Memory)	467
13.11.1	Event Output	468
13.11.2	Event Input	470
13.11.3	Data Output	473
13.11.4	Data Input	475
13.12	通信方法	477
13.12.1	手順	477
13.12.2	相互通信の際の注意点	478
<b>第 14 章</b>	<b>DMAC</b>	<b>479</b>
14.1	レジスタマップ	479
14.1.1	DMA 制御レジスタ	480
14.1.2	DMA 割り込みクリアレジスタ	480
14.1.3	ポート/ソースアドレスレジスタ	480
14.1.4	メモリ/デスティネーションアドレスレジスタ	481
14.1.5	転送レンジレジスタ	481
14.1.6	データバッファレジスタ	481
14.1.7	転送モード制御レジスタ	482
14.1.8	ステータスレジスタ	484
<b>第 15 章</b>	<b>バスサイジング機能付き DMA</b>	<b>485</b>
15.1	本 DMA の特徴	485
15.2	制御レジスタ	485
15.3	制御レジスタ詳細	485
15.3.1	PSA レジスタ	485
15.3.2	MDA レジスタ	486
15.3.3	LENGTH レジスタ	486

15.3.4	MODE レジスタ	486
15.4	注意事項	487
<b>第 16 章</b>	<b>パルスカウンタ</b>	<b>489</b>
16.1	パルスカウンタ概要	489
16.2	レジスタインタフェース	489
16.2.1	パルスカウンタ制御レジスタ	489
16.2.2	コンペアデータレジスタ	491
16.2.3	カウンタレジスタ	491
16.2.4	タイマレジスタ	492
<b>第 17 章</b>	<b>PWM 発生器</b>	<b>493</b>
17.1	PWM 発生器概要	493
17.2	PWM コントロールレジスタ	494
17.3	PWM 周期制御レジスタ	495
17.4	PWM 反転制御レジスタ	496
17.5	デッドタイムレジスタ	496
<b>第 18 章</b>	<b>PWM 入力器</b>	<b>499</b>
18.1	PWM 入力器概要	499
18.2	PWMIN コントロールレジスタ	499
18.3	PWMIN HIGH レジスタ	500
18.4	PWMIN LOW レジスタ	500
<b>第 19 章</b>	<b>PCI I/F</b>	<b>501</b>
19.1	アドレスマップ	501
19.1.1	Local Bus	501
19.1.2	PCI Bus	502
19.2	PCI I/F レジスタマップ	504
19.2.1	割り込み制御レジスタ	504
19.2.2	プログラム制御レジスタ	506
19.2.3	MailboxA	507
19.2.4	MailboxB	507
19.2.5	Local AD	508
19.2.6	Local Bus Access Port	508
19.2.7	PCI Bus Access Port	509
19.2.8	Current Local AD	509
19.3	Master Transaction 用 DMA レジスタマップ	509
19.3.1	Address Register	510
19.3.2	Data Count Register	510
19.3.3	DMA Control Register	511
19.3.4	DMA Stop/Reset Register	512
19.3.5	FIFO Data Register	513
19.3.6	FIFO Request Parameter Register	513
19.3.7	FIFO Control Register	514
19.3.8	FIFO Stop/Reset Register	515

19.4	動作/使用方法 . . . . .	515
19.4.1	Target Transaction (PCI → Local) . . . . .	515
19.4.2	Master Transaction( Local → PCI ) . . . . .	516
<b>第 20 章</b>	<b>IEEE1394</b>	<b>517</b>
20.1	概要 . . . . .	517
20.1.1	特徴 . . . . .	517
20.1.2	関連資料 . . . . .	517
20.2	構成 . . . . .	518
20.2.1	全体構成 . . . . .	518
20.2.2	入出力端子説明 . . . . .	519
20.2.3	内部構成図 . . . . .	523
20.2.4	構成図説明 . . . . .	524
20.3	レジスタ . . . . .	525
20.3.1	レジスタ一覧 . . . . .	525
20.3.2	レジスタ内容 . . . . .	528
20.4	データフォーマット . . . . .	552
20.4.1	アシンクロナス データフォーマット . . . . .	552
20.4.2	アシンクロナス ストリーム データフォーマット . . . . .	560
20.4.3	アイソクロナス データフォーマット . . . . .	562
20.4.4	自己識別パケット送信 . . . . .	563
20.4.5	PHY コントロールパケット . . . . .	564
20.4.6	コード説明 . . . . .	564
20.5	機能 . . . . .	567
20.5.1	ホストインターフェース . . . . .	567
20.5.2	PHY インターフェース . . . . .	570
20.5.3	バッファのコントロール . . . . .	575
20.5.4	制御フロー . . . . .	578
20.5.5	Asynchronous パケット送信時の BusNumber . . . . .	583
<b>第 21 章</b>	<b>Universal Asynchronous Receiver/Transmitter</b>	<b>585</b>
21.1	アドレスマップ . . . . .	585
21.1.1	Receiver Buffer (RB) / Transmitter Holding Register (THR) . . . . .	585
21.1.2	Interrupt Enable Register (IER) . . . . .	586
21.1.3	Interrupt Identification Register (IIR) . . . . .	586
21.1.4	FIFO Control Register (FCR) . . . . .	587
21.1.5	Line Control Register (LCR) . . . . .	588
21.1.6	Modem Control Register (MCR) . . . . .	589
21.1.7	Line Status Register (LSR) . . . . .	590
21.1.8	Modem Status Register (MSR) . . . . .	592
21.1.9	Divisor Latches (DL) . . . . .	592
21.2	動作/使用方法 . . . . .	593
21.2.1	Initialization . . . . .	593

---

<b>第 22 章 DDR SDRAM I/F</b>	<b>595</b>
22.1 レジスタマップ	595
22.1.1 主記憶 I/F 幅設定レジスタ	596
22.1.2 I/F 起動レジスタ	596
22.1.3 メモリモジュール設定レジスタ	596
22.1.4 EMRS 設定レジスタ	597
22.1.5 MRS 設定レジスタ	597
22.1.6 DDR 設定レジスタ 1	598
22.1.7 DDR 設定レジスタ 2	598
22.1.8 リフレッシュインターバル設定レジスタ	598
<b>第 23 章 Serial Peripheral Interface Unit</b>	<b>601</b>
23.1 Outline	601
23.2 Interface	601
23.2.1 Address Format	601
23.2.2 Control Register	601
23.3 Operation	609
23.3.1 Manual Mode	609
23.3.2 Auto Mode	609
<b>第 24 章 Parallel I/O Unit</b>	<b>611</b>
24.1 Outline	611
24.2 Interface	611
24.2.1 Address Format	611
24.2.2 Control Register	611
24.3 Operation	614
<b>第 25 章 更新履歴</b>	<b>615</b>

# 1

## 概要

---

### 1.1 Overview

*RMT Processor* は、分散リアルタイムシステムを実現するために、リアルタイム通信・処理・制御を同時にハードウェアレベルで行うことを目的にして設計を行ったシステム LSI である。分散リアルタイムシステムを容易かつ効率的に実現するには、リアルタイム通信及びリアルタイム処理を行なうための基本機能を有した共通プラットフォームを用意し、それらをブロックを組み立てるように組み合わせてシステムを構築できるようにすれば良いと考えられる。プラットフォームに必要な機能としては、リアルタイム処理機能、リアルタイム通信機能、コンピュータ用周辺機能、各種周辺制御機能が考えられる。プラットフォームとして様々なシステムの中に容易に組み込んで使用できるようにするために、*RMT Processor* は以下の機能を全て 1 チップに集積 (System-on-a-chip) している。

- リアルタイム処理機能 (*RMT Processing Unit*)
- リアルタイム通信機能 (*Responsive Link*)
- コンピュータ用周辺機能 (PCI-X, IEEE1394, DDR SDRAM I/Fs, DMAC, etc.)
- 各種周辺制御機能 (PWM Generators, Pulse Counters, etc.)

システム設計者は本チップに必要な I/O (センサ, アクチュエータ, デジタルカメラ等) を接続するだけで必要な機能を実現できる。それら I/O を接続し固有の機能を有した *RMT Processor* をそのシステムにふさわしいトポロジで *Responsive Link* を用いて複数個接続することによって、分散リアルタイムシステムを構築する。

### 1.2 設計ポリシー

*RMT Processor* はリアルタイム処理・通信の理論をそのまま実現できることを目標にして設計されている。リアルタイムスケジューラには、動的スケジューリングとして EDF (Earliest Deadline First) 等があり、静的スケジューリングとして RM (Rate Monotonic) 等があるが、ほぼ全てのリアルタイムスケジューリングは、優先度に従ってプリエンブションを行いながら実行や通信を行うことを要求する。プリエンブションは、演算 (処理) の場合はコンテキストスイッチに相当し、通信の場合はパケットの追い越しに相当する。

従って、処理の場合は優先度付きスレッドのハードウェアによる優先実行やコンテキストスイッチのオーバヘッドの削減等を実現する．通信の場合は、従来までの通信では実現されていなかった、優先度付きパケットの追い越し機構等を実現する．

*RMT Processor* はこれらの機能を実装することにより、リアルタイムスケジューリング理論を背景に設計されたリアルタイムスケジューラ（ソフトウェア）により優先度付けされた処理や通信を、そのまま理論通りにリアルタイム処理および通信を実現することのできるハードウェアを実現している．

### 1.3 全体構成

図 1.1 に *RMT Processor* のブロック図を示す．*RMT PU* は、256bit のバスを介して DDR SDRAM I/F と接続している．バンド幅の広いバスを用いてプロセッシングコアとメインメモリを接続することにより、命令フェッチや後に述べるベクトル演算において、メモリアクセスのスループットを改善している．

*Responsive Link*、各種 I/O は 32bit バスに接続されている．32bit バスと 256bit バスはゲートウェイ（GW）を介して接続されている．それぞれのバスを流れるデータはゲートウェイにおいてバスサイジングが行われ、もう片方のバスに送られる．また、*Responsive Link* のイベントリンクは *RMT PU* のメモリアクセスユニットに直接接続され、プロセッシングコアからはバスを介さず、制御レジスタの一部としてアクセスすることができる．これにより、高速にイベントリンクにアクセスすることが可能である．

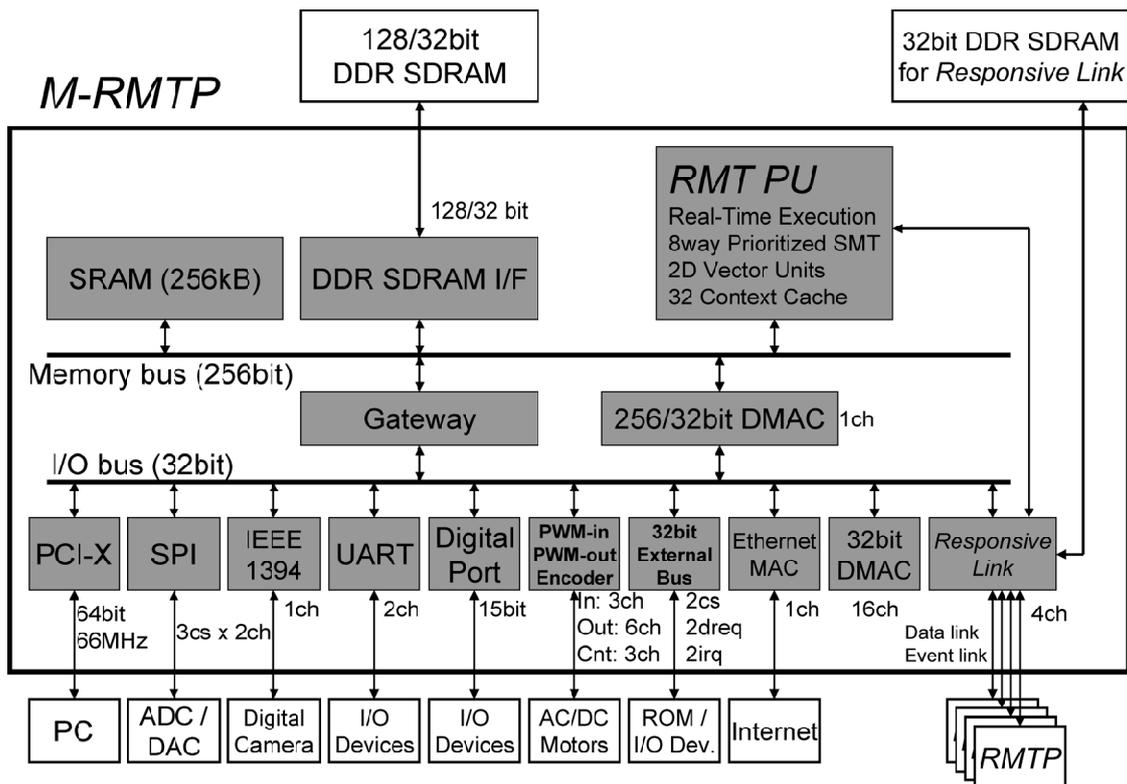


図 1.1: *RMT Processor* のブロック図

以下に *RMT Processor* が持つ I/O を示す．

- *Responsive Link* (4ch)

- PCI-X : 64bit, 66MHz (1ch)
- Serial Peripheral Interface (3cs × 2ch)
- IEEE1394 (1ch)
- UART (2ch)
- Digital Port (15bit)
- PWM Encoder
  - Input (3ch)
  - Output (6ch)
  - Pulse Counter (3ch)
- External Bus I/F (2cs, 2dreq, 2irq)
- Ethernet MAC (1ch)
- 32bit DMA Controller (4ch × 4)
- 256/32bit DMA Controller (1ch)
- 128bit DDR SDRAM I/F (1ch)
- SRAM (256kB)

## 1.4 Responsive Multithreaded Processing Unit

*RMT PU*は8wayの細粒度マルチスレッディングに優先度を用いた制御を行うことにより、ハードウェアで様々なレベルのリアルタイム処理を支援する。マルチスレッドアーキテクチャでは複数のスレッドが並列に実行されるため、スレッド間で演算器やキャッシュシステム等の計算資源の競合が起こる。競合が起こった場合、*RMT PU*はスレッド毎に設定された優先度を基に、優先度のより高い命令に対して先に計算資源を割り当てる。これにより並列に実行しているスレッドの中で、優先度の高いスレッドから優先的に実行する。

図 1.2 に *RMT PU* のブロック図を示す。*RMT PU* は命令発行ユニット (Issue Unit)、命令演算ユニット (Execution Unit)、キャッシュユニット (Cache Unit) の大きく 3 つに分かれる。命令発行ユニットは各スレッドの実行を制御し、優先度に従って命令演算ユニットに対して各スレッドの命令を送る。命令演算ユニットは命令発行ユニットから送られてきた命令を演算する。キャッシュユニットは命令発行ユニットからの命令フェッチ要求、命令演算ユニットからのデータアクセス要求を処理する。

### 1.4.1 命令発行ユニット

命令発行ユニットの役割は各スレッドの実行を制御し、命令演算ユニットに対して命令を発行することである。表 1.1 に命令発行ユニットの概要を示す。

アクティブスレッドとはプロセッサ内に保持されているスレッドで、すぐに実行を開始することができる。キャッシュスレッドとは後で述べるコンテキストキャッシュ内に保持されているスレッドを示す。優先度は 8bit を用いて 256level で表し、値が大きいくほど優先度は高くなる。

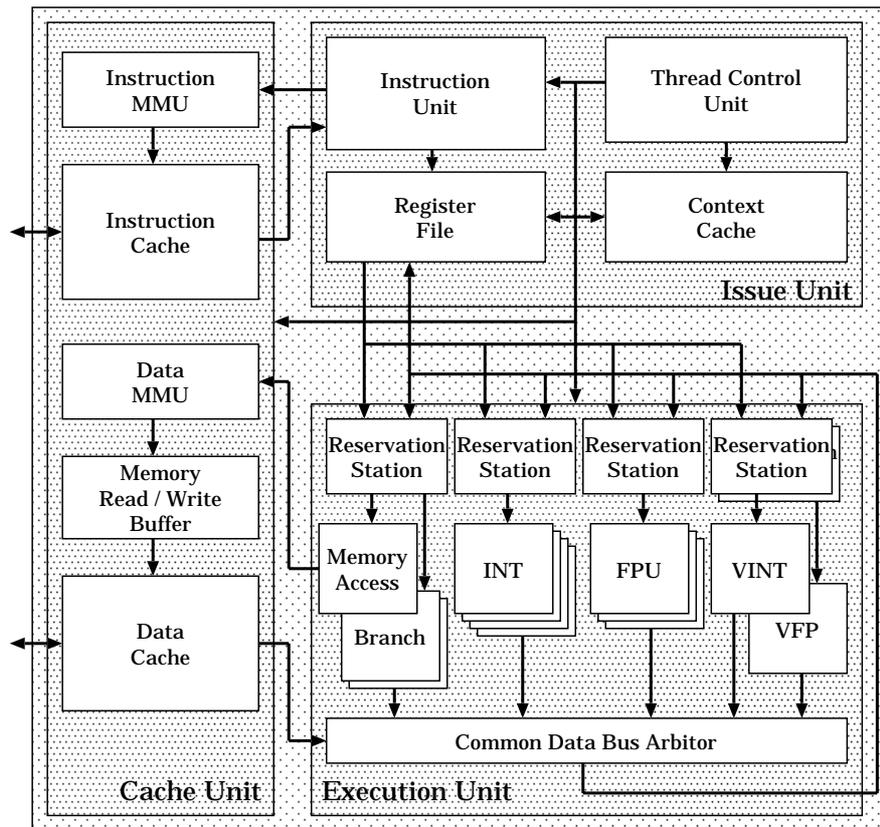


図 1.2: RMT PU のブロック図

13	12:9	8	7:0
ENABLE	STATE	KEEP	PRIORITY

図 1.3: スレッドテーブルのフォーマット

各スレッドの制御はスレッド制御ユニットで行う。アクティブスレッドはスレッド制御ユニット内にあるスレッドテーブルによって管理される。スレッドテーブルのフォーマットを図 1.3 に示す。ENABLE フィールドはアクティブスレッドが有効であるかどうかを示す。STATE フィールドはアクティブスレッドの状態を示し、実行中、停止中、後述するコンテキストキャッシュへの退避中等といった状態を示す。KEEP フィールドはアクティブスレッドをプロセッサ内に保持しつづけるかどうかを示す。PRIORITY フィールドはスレッドの優先度を示し、この値が RMT PU 全体で使用される。

RMT PU ではスレッドの生成、削除、実行、停止、優先度の設定等のために新たに命令を追加した。スレッド制御ユニットはこれらの命令が発行されると、命令に応じてスレッドテーブルを書き換え、アクティブスレッドの制御を行う。

先に述べた通り、RMT PU では 8 つのコンテキストをプロセッサ内に保持して実行することができる。しかしそれ以上のスレッドを実行する場合、コンテキストスイッチが発生する。コンテキストスイッチは現在実行しているスレッドのコンテキストをメモリに退避し、新しく実行するスレッドのコンテキストをメモリから復帰しなければならないため、オーバーヘッドが大きくなる。

RMT PU ではコンテキストを格納するための専用キャッシュをオンチップに用意し、レジスタファイル

表 1.1: 命令発行ユニットの概要

アクティブスレッド数	8
キャッシュスレッド数	32
優先度の指定	256level
命令フェッチ数	8
同時命令発行数	4
同時命令完了数	4
整数レジスタ数	32bit × 32entry × 8set
整数リネームレジスタ数	32bit × 64entry
浮動小数点レジスタ数	64bit × 8entry × 8set
浮動小数点リネームレジスタ数	64bit × 64entry

表 1.2: 命令演算ユニットの概要

整数演算器	4 + 1 ( Divider )
浮動小数点演算器	2 + 1 ( Divider )
64bit 整数演算器	1
整数ベクトル演算器	1 ( 8IU × 2 line )
浮動小数点ベクトル演算器	1 ( 4FPU × 2 line )
分岐ユニット	2
メモリアクセスユニット	1
同期ユニット	1

との間を広いバス ( GPR:256bit , FPR:128bit ) で接続している。コンテキストキャッシュは 32 個のコンテキストを格納することができ、コンテキストスイッチをハードウェアにより 4 クロックサイクルで行う。これによりコンテキストスイッチにかかるオーバーヘッドを大幅に削減する。

アクティブスレッドのコンテキストキャッシュへの退避、キャッシュスレッドのプロセッサ内への復帰、アクティブスレッドとキャッシュスレッドの入れ替えは新たに追加した命令により、スレッド制御ユニットが行う。スレッド制御ユニットはスレッドの退避命令や復帰命令、入れ替え命令を受け取ると、内部に保持しているキャッシュスレッドのテーブルを検索し、コンテキストキャッシュをアクセスするためのアドレスを生成して、コンテキストキャッシュをアクセスする。

命令発行ユニットは命令キャッシュアクセスと命令演算ユニットへの命令発行スロットで、優先度を用いた調停を行う。

### 1.4.2 命令演算ユニット

命令演算ユニットの役割は命令発行ユニットから送られてくる命令を演算することである。表 1.2 に命令演算ユニットの概要を示す。

*RMT PU* はリザベーションステーションとリオーダバッファを用いて、アウトオブオーダ実行を行う。*RMT PU* では複数のスレッドが並列に実行されているため、各演算器においてスレッド間で競合が起こる。命令演算ユニットではリザベーションステーションにおいて、優先度による制御を行う。リザベーションステーションでは演算に必要なオペランドがそろうまで命令は保持される。演算に必要なオペランドがそ

表 1.3: キャッシュユニットの概要

TLB エントリ (命令, データ)	64 entry
キャッシュサイズ (命令, データ)	32K byte
victim cache (命令, データ)	512 byte

るい命令の実行が可能になると、各演算器に対して命令が発行される。*RMT PU*では複数の命令が実行可能になった場合、リザベーションステーションは、各命令の優先度を調べ、優先度の高い命令から先に演算器に発行する。これにより優先度の高いスレッドの命令に対して、先に演算器を割り当てる。

一方、マルチメディア処理のようなソフトリアルタイム処理では多くのデータを繰り返し演算しなければならないため、高い演算性能が要求される。このような処理ではデータの並列性を利用して演算性能を高めることができる。

*RMT PU*ではベクトル演算機構を用いている。ベクトル演算により、少ない命令スロットを有効に活用し、ソフトリアルタイム処理に要求される高い演算性能を実現する。また、ベクトル演算を行うスレッドの数やプログラムによって必要とされるベクトルレジスタの構成は異なってくる。そこで *RMT PU*では整数、浮動小数点共に 512 セットあるベクトルレジスタを、ベクトル長やレジスタの個数等の構成を動的に変更してスレッド間で共有することにより、複数のスレッドで柔軟なベクトル演算を可能としている。

ベクトル演算は整数演算、浮動小数点演算共に 2 つの演算パイプラインが並列に動作することにより、複数のスレッドで並列してベクトル演算を行うことができる。各演算パイプラインは、整数演算パイプラインで 8 個、浮動小数点演算パイプラインで 4 個の演算器を持つことにより、複数のベクトル要素を並列に演算する。また、プログラマが複合演算を定義し、定義した命令を 1 命令で実行することにより、ベクトル演算器の使用率を向上させ、ベクトル演算の性能を向上させている。

### 1.4.3 キャッシュユニット

キャッシュユニットの役割は命令発行ユニットから送られてくる命令フェッチ要求と、命令実行ユニットから送られてくるデータアクセス要求を処理することである。表 1.3 にキャッシュユニットの概要を示す。

キャッシュユニットは MMU (Memory Management Unit) を持ち、ハードウェアでアドレス変換を行うため、各スレッドは仮想アドレスを用いてプログラミングを行うことができる。

MMU が置かれる場所により、仮想アドレスでキャッシュをアクセスするか物理アドレスでキャッシュをアクセスするかが決まる。図 1.2 に示した通り、*RMT PU*では MMU はキャッシュよりも前に置かれ、キャッシュアクセスを行う前にアドレス変換を行う。よってキャッシュは物理アドレスを用いてアクセスされる。キャッシュアクセスの前にアドレス変換を行うため、キャッシュアクセスにかかるレイテンシが増加するが、実行するスレッドがコンテキストスイッチにより切り替わった場合でもキャッシュをフラッシュする必要がなくなる。また、複数のスレッドでメモリ領域を共有する場合、仮想アドレスでキャッシュをアクセスすると同一の物理メモリのデータが複数キャッシュされる問題 (synonym) が起こるが、物理アドレスを用いてキャッシュをアクセスすることによりその問題を回避することができる。

MMU における TLB エントリには仮想ページ番号、物理ページ番号の他に、複数スレッドで共有するための共有情報、コンテキストグループ番号を指定する。*RMT PU*は最大 8 つのスレッドが動作するため、TLB エントリの実効率が低くなるのが考えられる。共有情報を用いることにより、複数のスレッドで TLB エントリを共有し、使用する TLB のエントリ数を削減することができる。

共有情報を設定した後に、新しいスレッドを共有情報に追加する場合はコンテキストグループ番号を用いる。TLB を設定する場合、コンテキストグループ番号を指定することにより、コンテキストグループ番

号の一致するエントリの共有情報に自身のスレッドを追加する．これにより，使用する TLB のエントリ数を増やすことなく TLB を有効化することができる．

キャッシュは命令キャッシュ，データキャッシュ共に 8way の set-associative 方式，ブロックサイズは 32byte で，キャッシュアクセスはパイプライン化されている．キャッシュミスが起こった場合，入れ換えるブロックの選択方法は LRU と優先度がある．優先度を基に入れ換えるブロックを選択する場合，より優先度の低いスレッドが使用しているブロックから先にキャッシュを追い出される．これにより，優先度の高いスレッドのキャッシュブロックが追い出されることを防ぐ．

victim cache は，キャッシュブロックの入れ換えに伴ないキャッシュを追い出されたブロックを，full associative 方式で保持する．キャッシュミスを起した場合，victim cache にデータが残っていれば，そのブロックをキャッシュに戻すことにより，キャッシュミスによる内部バスへの要求を減らし，メモリアクセスの遅延を減少させる．

キャッシュミス等によりバスを介して下位メモリをアクセスする場合にも優先度を用いた制御を行う．メモリアクセスはキャッシュよりも低速なため，待ち行列が発生する．この場合，より優先度の高いスレッドからバスを使用して下位メモリにアクセスする．

## 1.5 *Responsive Link*

*Responsive Link* は，柔軟なリアルタイム通信を実現するために，ソフトリアルタイム通信（データリンク）とハードリアルタイム通信（イベントリンク）の分離，パケットに優先度を付加しノード毎に高優先度パケットが低優先度パケットの追い越し，パケットの優先度が異なると優先度毎に別経路を設定して専用回線や迂回路を実現可能，ノード毎に優先度を付け替えることができ分散管理型でパケットの加減速を制御可能，ハードウェアによるエラー訂正，通信速度を動的に変更可能，トポロジーフリー，Hot-Plug&Play 等の様々な機能を実現する．

*Responsive Link* は国内では情報処理学会試行標準 (IPJS-TS 2003:0006) として標準化されており，国際的にはでは ISO/IEC JTC1 SC25 WG4 において標準化作業が行われている．



# 2

## PIN 配置

チップのピン配置の一覧を以下に示す。

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
1		LEFT	ext_iopad0_data24	pnl_tf12it0nn8	
		INOUT	外部バスデータ [24]		
2		LEFT	ext_iopad0_data23	pnl_tf12it0nn8	
		INOUT	外部バスデータ [23]		
3		LEFT	vss_go_l16	pnl_go	135
			IO digital VSS(0V)		
4		LEFT	ext_iopad0_data22	pnl_tf12it0nn8	
		INOUT	外部バスデータ [22]		
5		LEFT	vdd_vop_l15	pnl_vop	135
			IO digital VDD(3.3V)		
6		LEFT	ext_iopad0_data21	pnl_tf12it0nn8	
		INOUT	外部バスデータ [21]		
7		LEFT	ext_iopad0_data20	pnl_tf12it0nn8	
		INOUT	外部バスデータ [20]		
8		LEFT	ext_iopad0_data19	pnl_tf12it0nn8	
		INOUT	外部バスデータ [19]		
9		LEFT	ext_iopad0_data18	pnl_tf12it0nn8	
		INOUT	外部バスデータ [18]		
10		LEFT	ext_iopad0_data17	pnl_tf12it0nn8	
		INOUT	外部バスデータ [17]		
11		LEFT	vdd_vc_l27	pnl_vc	135
			Core digital VDD(1.0V)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
12		LEFT	ext_iopad0_data16	pnl_tf12it0nn8	
	INOUT	外部バスデータ [16]			
13		LEFT	vss_gcs_l39	pnl_gcs	135
		Core digital VSS(0V)			
14		LEFT	ext_iopad0_data15	pnl_tf12it0nn8	
	INOUT	外部バスデータ [15]			
15		LEFT	vss_go_l15	pnl_go	135
		IO digital VSS(0V)			
16		LEFT	ext_iopad0_data14	pnl_tf12it0nn8	
	INOUT	外部バスデータ [14]			
17		LEFT	vdd_vop_l14	pnl_vop	135
		IO digital VDD(3.3V)			
18		LEFT	ext_iopad0_data13	pnl_tf12it0nn8	
	INOUT	外部バスデータ [13]			
19		LEFT	ext_iopad0_data12	pnl_tf12it0nn8	
	INOUT	外部バスデータ [12]			
20		LEFT	ext_iopad0_data11	pnl_tf12it0nn8	
	INOUT	外部バスデータ [11]			
21		LEFT	vss_gcs_l38	pnl_gcs	135
		Core digital VSS(0V)			
22		LEFT	ext_iopad0_data10	pnl_tf12it0nn8	
	INOUT	外部バスデータ [10]			
23		LEFT	vdd_vc_l26	pnl_vc	135
		Core digital VDD(1.0V)			
24		LEFT	ext_iopad0_data9	pnl_tf12it0nn8	
	INOUT	外部バスデータ [9]			
25		LEFT	vss_go_l14	pnl_go	135
		IO digital VSS(0V)			
26		LEFT	ext_iopad0_data8	pnl_tf12it0nn8	
	INOUT	外部バスデータ [8]			
27		LEFT	vdd_vop_l13	pnl_vop	135
		IO digital VDD(3.3V)			
28		LEFT	ext_iopad0_data7	pnl_tf12it0nn8	
	INOUT	外部バスデータ [7]			
29		LEFT	vss_gcs_l37	pnl_gcs	135
		Core digital VSS(0V)			
30		LEFT	ext_iopad0_data6	pnl_tf12it0nn8	
	INOUT	外部バスデータ [6]			
31		LEFT	ext_iopad0_data5	pnl_tf12it0nn8	
	INOUT	外部バスデータ [5]			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
32		LEFT	ext_iopad0_data4	pnl_tf12it0nn8	
		INOUT	外部バスデータ [4]		
33		LEFT	vdd_vc_l25	pnl_vc	135
			Core digital VDD(1.0V)		
34		LEFT	ext_iopad0_data3	pnl_tf12it0nn8	
		INOUT	外部バスデータ [3]		
35		LEFT	vss_gcs_l36	pnl_gcs	135
			Core digital VSS(0V)		
36		LEFT	ext_iopad0_data2	pnl_tf12it0nn8	
		INOUT	外部バスデータ [2]		
37		LEFT	vss_go_l13	pnl_go	135
			IO digital VSS(0V)		
38		LEFT	ext_iopad0_data1	pnl_tf12it0nn8	
		INOUT	外部バスデータ [1]		
39		LEFT	vdd_vop_l12	pnl_vop	135
			IO digital VDD(3.3V)		
40		LEFT	ext_iopad0_data0	pnl_tf12it0nn8	
		INOUT	外部バスデータ [0]		
41		LEFT	ext_iopad0_err	pnl_tf12it0nn8	
		INOUT	外部バスエラー (外部バス)		
42		LEFT	ext_iopad0_ready	pnl_tf12it0nn8	
		INOUT	外部バスレディ(外部バス)		
43		LEFT	vss_gcs_l35	pnl_gcs	135
			Core digital VSS(0V)		
44		LEFT	ext_iopad0_br_ack1	pnl_tf12it0nn8	
		INOUT	外部バスバーストリクエスト Acknowledge[1] (外部バス)		
45		LEFT	vdd_vc_l24	pnl_vc	135
			Core digital VDD(1.0V)		
46		LEFT	ext_iopad0_br_ack0	pnl_tf12it0nn8	
		INOUT	外部バスバーストリクエスト Acknowledge[0] (外部バス)		
47		LEFT	ext_iopad0_be3	pnl_tf12it0nn8	
		INOUT	外部バスバイトイネーブル [3](外部バス)		
48		LEFT	ext_iopad0_be2	pnl_tf12it0nn8	
		INOUT	外部バスバイトイネーブル [2](外部バス)		
49		LEFT	vss_gcs_l34	pnl_gcs	135
			Core digital VSS(0V)		
50		LEFT	ext_iopad0_be1	pnl_tf12it0nn8	
		INOUT	外部バスバイトイネーブル [1](外部バス)		
51		LEFT	vss_go_l12	pnl_go	135
			IO digital VSS(0V)		

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
52		LEFT	ext_iopad0_be0	pnl_tf12it0nn8	
		INOUT	外部バスバイトイネーブル [0](外部バス)		
53		LEFT	vdd_vop_l11	pnl_vop	135
			IO digital VDD(3.3V)		
54		LEFT	ext_iopad0_burst1	pnl_tf12it0nn8	
		INOUT	外部バスバーストモード [1](外部バス)		
55		LEFT	ext_iopad0_burst0	pnl_tf12it0nn8	
		INOUT	外部バスバーストモード [0](外部バス)		
56		LEFT	ext_iopad0_rw	pnl_tf12it0nn8	
		INOUT	外部バス Read/Write(外部バス)		
57		LEFT	vdd_vc_l23	pnl_vc	135
			Core digital VDD(1.0V)		
58		LEFT	ext_iopad0_as	pnl_tf12it0nn8	
		INOUT	外部バスアドレスストローブ (外部バス)		
59		LEFT	vss_gcs_l33	pnl_gcs	135
			Core digital VSS(0V)		
60		LEFT	ext_iopad0_addr31	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [31](外部バス)		
61		LEFT	vss_go_l11	pnl_go	135
			IO digital VSS(0V)		
62		LEFT	ext_iopad0_addr30	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [30](外部バス)		
63		LEFT	vdd_vop_l10	pnl_vop	135
			IO digital VDD(3.3V)		
64		LEFT	ext_iopad0_addr29	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [29](外部バス)		
65		LEFT	vss_gcs_l32	pnl_gcs	135
			Core digital VSS(0V)		
66		LEFT	ext_iopad0_addr28	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [28](外部バス)		
67		LEFT	ext_iopad0_addr27	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [27](外部バス)		
68		LEFT	ext_iopad0_addr26	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [26](外部バス)		
69		LEFT	vdd_vc_l22	pnl_vc	135
			Core digital VDD(1.0V)		
70		LEFT	ext_iopad0_addr25	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [25](外部バス)		
71		LEFT	vss_gcs_l31	pnl_gcs	135
			Core digital VSS(0V)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
72		LEFT	ext_iopad0_addr24	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [24](外部バス)			
73		LEFT	vss_go_l10	pnl_go	135
		IO digital VSS(0V)			
74		LEFT	ext_iopad0_addr23	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [23](外部バス)			
75		LEFT	vdd_vop_l9	pnl_vop	135
		IO digital VDD(3.3V)			
76		LEFT	ext_iopad0_addr22	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [22](外部バス)			
77		LEFT	vss_gcs_l30	pnl_gcs	135
		Core digital VSS(0V)			
78		LEFT	ext_iopad0_addr21	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [21](外部バス)			
79		LEFT	ext_iopad0_addr20	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [20](外部バス)			
80		LEFT	ext_iopad0_addr19	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [19](外部バス)			
81		LEFT	vdd_vc_l21	pnl_vc	135
		Core digital VDD(1.0V)			
82		LEFT	ext_iopad0_addr18	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [18](外部バス)			
83		LEFT	vss_gcs_l29	pnl_gcs	135
		Core digital VSS(0V)			
84		LEFT	ext_iopad0_addr17	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [17](外部バス)			
85		LEFT	vss_go_l9	pnl_go	135
		IO digital VSS(0V)			
86		LEFT	ext_iopad0_addr16	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [16](外部バス)			
87		LEFT	vdd_vop_l8	pnl_vop	135
		IO digital VDD(3.3V)			
88		LEFT	ext_iopad0_addr15	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [15](外部バス)			
89		LEFT	ext_iopad0_addr14	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [14](外部バス)			
90		LEFT	ext_iopad0_addr13	pnl_tf12it0nn8	
	INOUT	外部バスアドレス [13](外部バス)			
91		LEFT	vss_gcs_l28	pnl_gcs	135
		Core digital VSS(0V)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
92		LEFT	ext_iopad0_addr12	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [12](外部バス)		
93		LEFT	vdd_vc_l20	pnl_vc	135
			Core digital VDD(1.0V)		
94		LEFT	ext_iopad0_addr11	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [11](外部バス)		
95		LEFT	vss_go_l8	pnl_go	135
			IO digital VSS(0V)		
96		LEFT	ext_iopad0_addr10	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [10](外部バス)		
97		LEFT	vdd_vop_l7	pnl_vop	135
			IO digital VDD(3.3V)		
98		LEFT	ext_iopad0_addr9	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [9](外部バス)		
99		LEFT	vss_gcs_l27	pnl_gcs	135
			Core digital VSS(0V)		
100		LEFT	ext_iopad0_addr8	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [8](外部バス)		
101		LEFT	ext_iopad0_addr7	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [7](外部バス)		
102		LEFT	ext_iopad0_addr6	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [6](外部バス)		
103		LEFT	vdd_vc_l19	pnl_vc	135
			Core digital VDD(1.0V)		
104		LEFT	ext_iopad0_addr5	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [5](外部バス)		
105		LEFT	ext_iopad0_addr4	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [4](外部バス)		
106		LEFT	ext_iopad0_addr3	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [3](外部バス)		
107		LEFT	vss_gcs_l26	pnl_gcs	135
			Core digital VSS(0V)		
108		LEFT	ext_iopad0_addr2	pnl_tf12it0nn8	
		INOUT	外部バスアドレス [2](外部バス)		
109		LEFT	ext_iopad0_grant	pnl_tf12it0nn8	
		OUTPUT	外部バスグラント (外部バス)		
110		LEFT	ext_iopad0_cs1	pnl_tf12it0nn8	
		OUTPUT	外部バスチップセレクト 1(外部バス)		
111		LEFT	vdd_vc_l18	pnl_vc	135
			Core digital VDD(1.0V)		

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
112		LEFT	ext_iopad0_cs0	pnl_tf12it0mn8	
		OUTPUT	外部バスチップセレクト 0(外部バス)		
113		LEFT	vss_go_l17	pnl_go	135
			IO digital VSS(0V)		
114		LEFT	ext_iopad0_dack1	pnl_tf12it0mn8	
		OUTPUT	外部バス DMA Acknowledge1 (外部バス)		
115		LEFT	vdd_vop_l6	pnl_vop	135
			IO digital VDD(3.3V)		
116		LEFT	ext_iopad0_dack0	pnl_tf12it0mn8	
		OUTPUT	外部バス DMA Acknowledge0 (外部バス)		
117		LEFT	vss_gcs_l25	pnl_gcs	135
			Core digital VSS(0V)		
118		LEFT	ext_iopad0_data_dir	pnl_tf12it0mn8	
		OUTPUT	外部バス Data Direction (外部バス)		
119		LEFT	vdd_vc_l17	pnl_vc	135
			Core digital VDD(1.0V)		
120		LEFT	ext_iopad0_ie	pnl_tf12it0mn8	
		OUTPUT	外部バス Input Enable (外部バス)		
121		LEFT	vss_go_l6	pnl_go	135
			IO digital VSS(0V)		
122		LEFT	ext_iopad0_oe	pnl_tf12it0mn8	
		OUTPUT	外部バス Output Enable (外部バス)		
123		LEFT	vdd_vop_l5	pnl_vop	135
			IO digital VDD(3.3V)		
124		LEFT	ext_iopad0_irq1	pnl_it2mn8	
		INPUT	外部割込み 1 (外部バス)		
125		LEFT	vss_gcs_l24	pnl_gcs	135
			Core digital VSS(0V)		
126		LEFT	ext_iopad0_irq0	pnl_it2mn8	
		INPUT	外部割込み 0 (外部バス)		
127		LEFT	ext_iopad0_req	pnl_it2mn8	
		INPUT	外部バスリクエスト (外部バス)		
128		LEFT	ext_iopad0_dreq1	pnl_it2mn8	
		INPUT	外部バス DMA リクエスト 2(外部バス)		
129		LEFT	vdd_vc_l16	pnl_vc	135
			Core digital VDD(1.0V)		
130		LEFT	ext_iopad0_auto_rdy_en	pnl_it2pd2	
		INPUT	外部バスレディ(外部バス)		
131		LEFT	vss_gcs_l23	pnl_gcs	135
			Core digital VSS(0V)		

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
132		LEFT	ext_iopad0_dreq0	pnl_it2nn8	
		INPUT	外部バス DMA リクエスト 1(外部バス)		
133		LEFT	vss_go_l5	pnl_go	135
			IO digital VSS(0V)		
134		LEFT	ext_iopad0_bit16	pnl_it2nn8	
		INPUT	外部バスサイズの指定: 16bit(外部バス)		
135		LEFT	vdd_vop_l4	pnl_vop	135
			IO digital VDD(3.3V)		
136		LEFT	ext_iopad0_bit8	pnl_it2nn8	
		INPUT	外部バスサイズの指定: 8bit(外部バス)		
137		LEFT	vss_gcs_l22	pnl_gcs	135
			Core digital VSS(0V)		
138		LEFT	pp_iopad0_pp_pwm_in_pad1	pnl_it2nn8	
		INPUT	PWM 入力 channel1 (PWM ジェネレータ)		
139		LEFT	vdd_vc_l15	pnl_vc	135
			Core digital VDD(1.0V)		
140		LEFT	pp_iopad0_pp_pwm_in_pad0	pnl_it2nn8	
		INPUT	PWM 入力 channel0 (PWM ジェネレータ)		
141		LEFT	vss_go_l4	pnl_go	135
			IO digital VSS(0V)		
142		LEFT	pp_iopad0_pp_pwm_out_pad8	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel8 (PWM ジェネレータ)		
143		LEFT	vss_gcs_l21	pnl_gcs	135
			Core digital VSS(0V)		
144		LEFT	pp_iopad0_pp_pwm_out_pad7	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel7 (PWM ジェネレータ)		
145		LEFT	pp_iopad0_pp_pwm_out_pad6	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel6 (PWM ジェネレータ)		
146		LEFT	pp_iopad0_pp_pwm_out_pad5	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel5 (PWM ジェネレータ)		
147		LEFT	vss_gcs_l20	pnl_gcs	135
			Core digital VSS(0V)		
148		LEFT	pp_iopad0_pp_pwm_out_pad4	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel4 (PWM ジェネレータ)		
149		LEFT	vdd_vc_l14	pnl_vc	135
			Core digital VDD(1.0V)		
150		LEFT	pp_iopad0_pp_pwm_out_pad3	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel3 (PWM ジェネレータ)		
151		LEFT	vss_gcs_l19	pnl_gcs	135
			Core digital VSS(0V)		

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
152		LEFT	pp_iopad0_pp_pwm_out_pad2	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel2 (PWM ジェネレータ)		
153		LEFT	vss_go_l3	pnl_go	135
			IO digital VSS(0V)		
154		LEFT	pp_iopad0_pp_pwm_out_pad1	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel1 (PWM ジェネレータ)		
155		LEFT	vdd_vop_l3	pnl_vop	135
			IO digital VDD(3.3V)		
156		LEFT	pp_iopad0_pp_pwm_out_pad0	pnl_tf04it0nn2	
		OUTPUT	PWM 出力 channel0 (PWM ジェネレータ)		
157		LEFT	vss_gcs_l18	pnl_gcs	135
			Core digital VSS(0V)		
158		LEFT	pp_iopad0_pp_pz_pad8	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel8 (パルスカウンタ)		
159		LEFT	pp_iopad0_pp_pz_pad7	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel7 (パルスカウンタ)		
160		LEFT	pp_iopad0_pp_pz_pad6	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel6 (パルスカウンタ)		
161		LEFT	vss_gcs_l17	pnl_gcs	135
			Core digital VSS(0V)		
162		LEFT	pp_iopad0_pp_pz_pad5	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel5 (パルスカウンタ)		
163		LEFT	vdd_vc_l13	pnl_vc	135
			Core digital VDD(1.0V)		
164		LEFT	pp_iopad0_pp_pz_pad4	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel4 (パルスカウンタ)		
165		LEFT	pp_iopad0_pp_pz_pad3	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel3 (パルスカウンタ)		
166		LEFT	pp_iopad0_pp_pz_pad2	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel2 (パルスカウンタ)		
167		LEFT	vss_gcs_l16	pnl_gcs	135
			Core digital VSS(0V)		
168		LEFT	pp_iopad0_pp_pz_pad1	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel1 (パルスカウンタ)		
169		LEFT	pp_iopad0_pp_pz_pad0	pnl_it2nn8	
		INPUT	エンコーダ Z フェーズ入力 channel0 (パルスカウンタ)		
170		LEFT	pp_iopad0_pp_pb_pad8	pnl_it2nn8	
		INPUT	エンコーダ B フェーズ入力 channel8 (パルスカウンタ)		
171		LEFT	vss_gcs_l15	pnl_gcs	135
			Core digital VSS(0V)		

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
172		LEFT	pp_iopad0_pp_pb_pad7	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel7 (パルスカウンタ)			
173		LEFT	vdd_vc_l12	pnl_vc	135
		Core digital VDD(1.0V)			
174		LEFT	pp_iopad0_pp_pb_pad6	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel6 (パルスカウンタ)			
175		LEFT	pp_iopad0_pp_pb_pad5	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel5 (パルスカウンタ)			
176		LEFT	pp_iopad0_pp_pb_pad4	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel4 (パルスカウンタ)			
177		LEFT	vss_gcs_l14	pnl_gcs	135
		Core digital VSS(0V)			
178		LEFT	pp_iopad0_pp_pb_pad3	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel3 (パルスカウンタ)			
179		LEFT	vdd_vop_l2	pnl_vop	135
		IO digital VDD(3.3V)			
180		LEFT	pp_iopad0_pp_pb_pad2	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel2 (パルスカウンタ)			
181		LEFT	vss_gcs_l13	pnl_gcs	135
		Core digital VSS(0V)			
182		LEFT	pp_iopad0_pp_pb_pad1	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel1 (パルスカウンタ)			
183		LEFT	vdd_vc_l11	pnl_vc	135
		Core digital VDD(1.0V)			
184		LEFT	pp_iopad0_pp_pb_pad0	pnl_lit2m8	
	INPUT	エンコーダ B フェーズ入力 channel0 (パルスカウンタ)			
185		LEFT	pp_iopad0_pp_pa_pad8	pnl_lit2m8	
	INPUT	エンコーダ A フェーズ入力 channel8 (パルスカウンタ)			
186		LEFT	pp_iopad0_pp_pa_pad7	pnl_lit2m8	
	INPUT	エンコーダ A フェーズ入力 channel7 (パルスカウンタ)			
187		LEFT	vss_gcs_l12	pnl_gcs	135
		Core digital VSS(0V)			
188		LEFT	pp_iopad0_pp_pa_pad6	pnl_lit2m8	
	INPUT	エンコーダ A フェーズ入力 channel6 (パルスカウンタ)			
189		LEFT	pp_iopad0_pp_pa_pad5	pnl_lit2m8	
	INPUT	エンコーダ A フェーズ入力 channel5 (パルスカウンタ)			
190		LEFT	pp_iopad0_pp_pa_pad4	pnl_lit2m8	
	INPUT	エンコーダ A フェーズ入力 channel4 (パルスカウンタ)			
191		LEFT	vss_gcs_l11	pnl_gcs	135
		Core digital VSS(0V)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
192		LEFT	pp_iopad0_pp_pa_pad3	pnl_it2nm8	
	INPUT		エンコーダ A フェーズ入力 channel3 (パルスカウンタ)		
193		LEFT	vdd_vc_l10	pnl_vc	135
			Core digital VDD(1.0V)		
194		LEFT	pp_iopad0_pp_pa_pad2	pnl_it2nm8	
	INPUT		エンコーダ A フェーズ入力 channel2 (パルスカウンタ)		
195		LEFT	vss_go_l2	pnl_go	135
			IO digital VSS(0V)		
196		LEFT	pp_iopad0_pp_pa_pad1	pnl_it2nm8	
	INPUT		エンコーダ A フェーズ入力 channel1 (パルスカウンタ)		
197		LEFT	vss_gcs_l10	pnl_gcs	135
			Core digital VSS(0V)		
198		LEFT	pp_iopad0_pp_pa_pad0	pnl_it2nm8	
	INPUT		エンコーダ A フェーズ入力 channel0 (パルスカウンタ)		
199		LEFT	vdd_vc_l9	pnl_vc	135
			Core digital VDD(1.0V)		
200		LEFT	vdd_vc_l8	pnl_vc	135
			Core digital VDD(1.0V)		
201		LEFT	vss_gcs_l9	pnl_gcs	135
			Core digital VSS(0V)		
202		LEFT	clk_iopad0_reset_outer	pnl_tf12it0nm8	
	OUTPUT		外部ユニット用リセット		
203		LEFT	vdd_vc_l7	pnl_vc	135
			Core digital VDD(1.0V)		
204		LEFT	vss_gcs_l8	pnl_gcs	135
			Core digital VSS(0V)		
205		LEFT	vss_gcs_l7	pnl_gcs	135
			Core digital VSS(0V)		
206		LEFT	vdd_vc_l6	pnl_vc	135
			Core digital VDD(1.0V)		
207		LEFT	vss_gcs_l6	pnl_gcs	135
			Core digital VSS(0V)		
208		LEFT	vdd_vc_l5	pnl_vc	135
			Core digital VDD(1.0V)		
209		LEFT	vss_gcs_l5	pnl_gcs	135
			Core digital VSS(0V)		
210		LEFT	vss_gcs_l4	pnl_gcs	135
			Core digital VSS(0V)		
211		LEFT	clk_iopad0_fout_a	pnl_tf12it0nm8	
	OUTPUT		PLL クロック出力		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
212		LEFT	vdd_vop_l1	pnl_vop	135
		IO digital VDD(3.3V)			
213		LEFT	vdd_vc_l4	pnl_vc	135
		Core digital VDD(1.0V)			
214		LEFT	vdd_vc_l3	pnl_vc	135
		Core digital VDD(1.0V)			
215		LEFT	vss_gcs_l3	pnl_gcs	135
		Core digital VSS(0V)			
216		LEFT	vdd_vc_l2	pnl_vc	135
		Core digital VDD(1.0V)			
217		LEFT	vss_go_l1	pnl_go	135
		IO digital VSS(0V)			
218		LEFT	vss_gcs_l2	pnl_gcs	135
		Core digital VSS(0V)			
219		LEFT	vdd_vc_l1	pnl_vc	135
		Core digital VDD(1.0V)			
220		LEFT	vss_gcs_l1	pnl_gcs	135
		Core digital VSS(0V)			
221		LEFT	vss_gcs_l0	pnl_gcs	135
		Core digital VSS(0V)			
222		LEFT	vdd_vc_l0	pnl_vc	135
		Core digital VDD(1.0V)			
223		LEFT	clk_iopad0_clk_outer	pnl_tf12it0nn8	
	OUTPUT	外部ユニット用クロック			
224		LEFT	clk_iopad0_reset_in	pnl_it2pu8	
	INPUT	リセット入力			
225		LEFT	clk_iopad0_f_a0	pnl_it2pd8	
	INPUT	PLL 設定 Feedback Divider[0]			
226		LEFT	clk_iopad0_f_a1	pnl_it2pu8	
	INPUT	PLL 設定 Feedback Divider[1]			
227		LEFT	clk_iopad0_f_a2	pnl_it2pu8	
	INPUT	PLL 設定 Feedback Divider[2]			
228		LEFT	clk_iopad0_f_a3	pnl_it2pu8	
	INPUT	PLL 設定 Feedback Divider[3]			
229		LEFT	clk_iopad0_f_a4	pnl_it2pd8	
	INPUT	PLL 設定 Feedback Divider[4]			
230		LEFT	clk_iopad0_f_a5	pnl_it2pd8	
	INPUT	PLL 設定 Feedback Divider[5]			
231		LEFT	vdd_vop_l0	pnl_vop	135
		IO digital VDD(3.3V)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
232		LEFT	clk_iopad0_fn_a	pnl_lit2nn8	
		INPUT	PLL 設定 リファレンスクロック入力		
233		LEFT	clk_iopad0_bp_a	pnl_lit2pd8	
		INPUT	PLL 設定 バイパス信号		
234		LEFT	clk_iopad0_r_a0	pnl_lit2pd8	
		INPUT	PLL 設定 Input Divider[0]		
235		LEFT	clk_iopad0_r_a1	pnl_lit2pu8	
		INPUT	PLL 設定 Input Divider[1]		
236		LEFT	clk_iopad0_r_a2	pnl_lit2pd8	
		INPUT	PLL 設定 Input Divider[2]		
237		LEFT	clk_iopad0_r_a3	pnl_lit2pd8	
		INPUT	PLL 設定 Input Divider[3]		
238		LEFT	vss_go_l0	pnl_go	135
			IO digital VSS(0V)		
239		LEFT	clk_iopad0_oeb_a	pnl_lit2pd8	
		INPUT	PLL 設定 FOUT Enable		
240		LEFT	clk_iopad0_od_a	pnl_lit2pd8	
		INPUT	PLL 設定 Output Divider		
241		LEFT	clk_iopad0_pd_a	pnl_lit2pd8	
		INPUT	PLL 設定 Power Down		
242		LEFT	clk_iopad0_prcut2p1	PRCUT2P	
			Power cut cell		
243		LEFT	clk_iopad0_pvss2p	PVSS2P	
			IO analog VSS		
244		LEFT	clk_iopad0_pvdd2p	PVDD2P	
			IO analog VDD(3.3V)		
245		LEFT	clk_iopad0_pvdd1p1	PVDD1P	
			PLL analog VDD(3.3V)		
246		LEFT	clk_iopad0_pvdd1p0	PVDD1P	
			PLL analog VDD(3.3V)		
247		LEFT	clk_iopad0_pvss1pc0	PVSS1PC	
			PLL digital VSS		
248		LEFT	clk_iopad0_pvss1p1	PVSS1P	
			PLL analog VSS		
249		LEFT	clk_iopad0_pvss1p0	PVSS1P	
			PLL analog VSS		
250		LEFT	clk_iopad0_pvdd1pc0	PVDD1PC	
			PLL digital VDD(1.0V)		
251		LEFT	clk_iopad0_prcut2p0	PRCUT2P	
			Power cut cell		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
252		BOTTOM	pci_poc0	PVDD2POC	45
		POC power rail			
253		BOTTOM	pci_iopad0_uiov_hi_0	PCI66SDGZ	
	INOUT	AD[32] (PCI バス)			
254		BOTTOM	pci_iopad0_uiov_hi_1	PCI66SDGZ	
	INOUT	AD[33] (PCI バス)			
255		BOTTOM	pci_iopad0_uiov_hi_2	PCI66SDGZ	
	INOUT	AD[34] (PCI バス)			
256		BOTTOM	pci_iopad0_uiov_hi_3	PCI66SDGZ	
	INOUT	AD[35] (PCI バス)			
257		BOTTOM	pci_iopad0_uiov_hi_4	PCI66SDGZ	
	INOUT	AD[36] (PCI バス)			
258		BOTTOM	pci_iopad0_uiov_hi_5	PCI66SDGZ	
	INOUT	AD[37] (PCI バス)			
259		BOTTOM	pci_iopad0_uiov_hi_6	PCI66SDGZ	
	INOUT	AD[38] (PCI バス)			
260		BOTTOM	pci_iopad0_uiov_hi_7	PCI66SDGZ	
	INOUT	AD[39] (PCI バス)			
261		BOTTOM	pci_iopad0_uiov_hi_8	PCI66SDGZ	
	INOUT	AD[40] (PCI バス)			
262		BOTTOM	pci_iopad0_uiov_hi_9	PCI66SDGZ	
	INOUT	AD[41] (PCI バス)			
263		BOTTOM	pci_iopad0_uiov_hi_10	PCI66SDGZ	
	INOUT	AD[42] (PCI バス)			
264		BOTTOM	vss_io_b15	PVSS2DGZ	170
		IO digital VSS(0V)			
265		BOTTOM	pci_iopad0_uiov_hi_11	PCI66SDGZ	
	INOUT	AD[43] (PCI バス)			
266		BOTTOM	pci_iopad0_uiov_hi_12	PCI66SDGZ	
	INOUT	AD[44] (PCI バス)			
267		BOTTOM	pci_iopad0_uiov_hi_13	PCI66SDGZ	
	INOUT	AD[45] (PCI バス)			
268		BOTTOM	pci_iopad0_uiov_hi_14	PCI66SDGZ	
	INOUT	AD[46] (PCI バス)			
269		BOTTOM	pci_iopad0_uiov_hi_15	PCI66SDGZ	
	INOUT	AD[47] (PCI バス)			
270		BOTTOM	pci_iopad0_uiov_hi_16	PCI66SDGZ	
	INOUT	AD[48] (PCI バス)			
271		BOTTOM	pci_iopad0_uiov_hi_17	PCI66SDGZ	
	INOUT	AD[49] (PCI バス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
272		BOTTOM	pci_iopad0_uiov_hi_18	PCI66SDGZ	
	INOUT	AD[50] (PCI バス)			
273		BOTTOM	pci_iopad0_uiov_hi_19	PCI66SDGZ	
	INOUT	AD[51] (PCI バス)			
274		BOTTOM	pci_iopad0_uiov_hi_20	PCI66SDGZ	
	INOUT	AD[52] (PCI バス)			
275		BOTTOM	pci_iopad0_uiov_hi_21	PCI66SDGZ	
	INOUT	AD[53] (PCI バス)			
276		BOTTOM	pci_iopad0_uiov_hi_22	PCI66SDGZ	
	INOUT	AD[54] (PCI バス)			
277		BOTTOM	pci_iopad0_uiov_hi_23	PCI66SDGZ	
	INOUT	AD[55] (PCI バス)			
278		BOTTOM	vdd_io_b16	PVDD2DGZ	45
		IO digital VDD(3.3V)			
279		BOTTOM	pci_iopad0_uiov_hi_24	PCI66SDGZ	
	INOUT	AD[56] (PCI バス)			
280		BOTTOM	vss_io_b14	PVSS2DGZ	170
		IO digital VSS(0V)			
281		BOTTOM	pci_iopad0_uiov_hi_25	PCI66SDGZ	
	INOUT	AD[57] (PCI バス)			
282		BOTTOM	vss_core_b35	PVSS1DGZ	40
		Core digital VSS(0V)			
283		BOTTOM	pci_iopad0_uiov_hi_26	PCI66SDGZ	
	INOUT	AD[58] (PCI バス)			
284		BOTTOM	vdd_core_b20	PVDD1DGZ	45
		Core digital VDD(1.0V)			
285		BOTTOM	pci_iopad0_uiov_hi_27	PCI66SDGZ	
	INOUT	AD[59] (PCI バス)			
286		BOTTOM	vss_core_b34	PVSS1DGZ	40
		Core digital VSS(0V)			
287		BOTTOM	pci_iopad0_uiov_hi_28	PCI66SDGZ	
	INOUT	AD[60] (PCI バス)			
288		BOTTOM	pci_iopad0_uiov_hi_29	PCI66SDGZ	
	INOUT	AD[61] (PCI バス)			
289		BOTTOM	pci_iopad0_uiov_hi_30	PCI66SDGZ	
	INOUT	AD[62] (PCI バス)			
290		BOTTOM	vss_core_b33	PVSS1DGZ	40
		Core digital VSS(0V)			
291		BOTTOM	pci_iopad0_uiov_hi_31	PCI66SDGZ	
	INOUT	AD[63] (PCI バス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
292		BOTTOM	vss_io_b13	PVSS2DGZ	170
		IO digital VSS(0V)			
293		BOTTOM	pci_iopad0_uio2	PCI66SDGZ	
	INOUT	PAR64 (PCI バス)			
294		BOTTOM	vdd_io_b15	PVDD2DGZ	45
		IO digital VDD(3.3V)			
295		BOTTOM	pci_iopad0_uiov2_hi_0	PCI66SDGZ	
	INOUT	C/BE[4] (PCI バス)			
296		BOTTOM	vdd_core_b19	PVDD1DGZ	45
		Core digital VDD(1.0V)			
297		BOTTOM	pci_iopad0_uiov2_hi_1	PCI66SDGZ	
	INOUT	C/BE[5] (PCI バス)			
298		BOTTOM	vss_core_b32	PVSS1DGZ	40
		Core digital VSS(0V)			
299		BOTTOM	pci_iopad0_uiov2_hi_2	PCI66SDGZ	
	INOUT	C/BE[6] (PCI バス)			
300		BOTTOM	pci_iopad0_uiov2_hi_3	PCI66SDGZ	
	INOUT	C/BE[7] (PCI バス)			
301		BOTTOM	pci_iopad0_uio9	PCI66SDGZ	
	INOUT	REQ64# (PCI バス)			
302		BOTTOM	vss_core_b31	PVSS1DGZ	40
		Core digital VSS(0V)			
303		BOTTOM	pci_iopad0_uio8	PCI66SDGZ	
	INOUT	ACK64# (PCI バス)			
304		BOTTOM	vss_io_b12	PVSS2DGZ	170
		IO digital VSS(0V)			
305		BOTTOM	pci_iopad0_uiov_lo_0	PCI66SDGZ	
	INOUT	AD[0] (PCI バス)			
306		BOTTOM	vdd_io_b14	PVDD2DGZ	45
		IO digital VDD(3.3V)			
307		BOTTOM	pci_iopad0_uiov_lo_1	PCI66SDGZ	
	INOUT	AD[1] (PCI バス)			
308		BOTTOM	vdd_core_b18	PVDD1DGZ	45
		Core digital VDD(1.0V)			
309		BOTTOM	pci_iopad0_uiov_lo_2	PCI66SDGZ	
	INOUT	AD[2] (PCI バス)			
310		BOTTOM	vss_core_b30	PVSS1DGZ	40
		Core digital VSS(0V)			
311		BOTTOM	pci_iopad0_uiov_lo_3	PCI66SDGZ	
	INOUT	AD[3] (PCI バス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
312		BOTTOM	pci_iopad0_uiov_lo_4	PCI66SDGZ	
	INOUT	AD[4] (PCI バス)			
313		BOTTOM	pci_iopad0_uiov_lo_5	PCI66SDGZ	
	INOUT	AD[5] (PCI バス)			
314		BOTTOM	vss_core_b29	PVSS1DGZ	40
		Core digital VSS(0V)			
315		BOTTOM	pci_iopad0_uiov_lo_6	PCI66SDGZ	
	INOUT	AD[6] (PCI バス)			
316		BOTTOM	vdd_core_b17	PVDD1DGZ	45
		Core digital VDD(1.0V)			
317		BOTTOM	pci_iopad0_uiov_lo_7	PCI66SDGZ	
	INOUT	AD[7] (PCI バス)			
318		BOTTOM	vss_core_b28	PVSS1DGZ	40
		Core digital VSS(0V)			
319		BOTTOM	pci_iopad0_uiov2_lo_0	PCI66SDGZ	
	INOUT	C/BE[0](PCI バス)			
320		BOTTOM	pci_iopad0_uiov_lo_8	PCI66SDGZ	
	INOUT	AD[8] (PCI バス)			
321		BOTTOM	pci_iopad0_uiov_lo_9	PCI66SDGZ	
	INOUT	AD[9] (PCI バス)			
322		BOTTOM	vss_core_b27	PVSS1DGZ	40
		Core digital VSS(0V)			
323		BOTTOM	pci_iopad0_uin4	PCI66SDGZ	
	INOUT	M66EN (PCI バス)			
324		BOTTOM	vdd_core_b16	PVDD1DGZ	45
		Core digital VDD(1.0V)			
325		BOTTOM	pci_iopad0_uiov_lo_10	PCI66SDGZ	
	INOUT	AD[10] (PCI バス)			
326		BOTTOM	vss_core_b26	PVSS1DGZ	40
		Core digital VSS(0V)			
327		BOTTOM	pci_iopad0_uiov_lo_11	PCI66SDGZ	
	INOUT	AD[11] (PCI バス)			
328		BOTTOM	vss_io_b11	PVSS2DGZ	170
		IO digital VSS(0V)			
329		BOTTOM	pci_iopad0_uiov_lo_12	PCI66SDGZ	
	INOUT	AD[12] (PCI バス)			
330		BOTTOM	vdd_io_b13	PVDD2DGZ	45
		IO digital VDD(3.3V)			
331		BOTTOM	pci_iopad0_uiov_lo_13	PCI66SDGZ	
	INOUT	AD[13] (PCI バス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
332		BOTTOM	vss_core_b25	PVSS1DGZ	40
		Core digital VSS(0V)			
333		BOTTOM	pci_iopad0_uio_v_lo_14	PCI66SDGZ	
	INOUT	AD[14] (PCI バス)			
334		BOTTOM	vdd_core_b15	PVDD1DGZ	45
		Core digital VDD(1.0V)			
335		BOTTOM	pci_iopad0_uio_v_lo_15	PCI66SDGZ	
	INOUT	AD[15] (PCI バス)			
336		BOTTOM	vss_core_b24	PVSS1DGZ	40
		Core digital VSS(0V)			
337		BOTTOM	pci_iopad0_uio_v2_lo_1	PCI66SDGZ	
	INOUT	C/BE[1](PCI バス)			
338		BOTTOM	pci_iopad0_uio1	PCI66SDGZ	
	INOUT	PAR (PCI バス)			
339		BOTTOM	pci_iopad0_uout2	PCI66SDGZ	
	INOUT	SERR# (PCI バス)			
340		BOTTOM	vss_core_b23	PVSS1DGZ	40
		Core digital VSS(0V)			
341		BOTTOM	pci_iopad0_uio10	PCI66SDGZ	
	INOUT	PERR# (PCI バス)			
342		BOTTOM	vdd_core_b14	PVDD1DGZ	45
		Core digital VDD(1.0V)			
343		BOTTOM	pci_iopad0_uin3	PCI66SDGZ	
	INOUT	LOCK# (PCI バス)			
344		BOTTOM	vss_io_b10	PVSS2DGZ	170
		IO digital VSS(0V)			
345		BOTTOM	pci_iopad0_uio6	PCI66SDGZ	
	INOUT	STOP# (PCI バス)			
346		BOTTOM	vdd_io_b12	PVDD2DGZ	45
		IO digital VDD(3.3V)			
347		BOTTOM	pci_iopad0_uio7	PCI66SDGZ	
	INOUT	DEVSEL# (PCI バス)			
348		BOTTOM	vss_core_b22	PVSS1DGZ	40
		Core digital VSS(0V)			
349		BOTTOM	pci_iopad0_uio5	PCI66SDGZ	
	INOUT	TRDY# (PCI バス)			
350		BOTTOM	vdd_core_b13	PVDD1DGZ	45
		Core digital VDD(1.0V)			
351		BOTTOM	pci_iopad0_uio4	PCI66SDGZ	
	INOUT	IRDY# (PCI バス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
352		BOTTOM	pci_iopad0_uio3	PCI66SDGZ	
	INOUT	FRAME# (PCIバス)			
353		BOTTOM	pci_iopad0_uiov2_lo_2	PCI66SDGZ	
	INOUT	C/BE[2] (PCIバス)			
354		BOTTOM	vss_core_b21	PVSS1DGZ	40
		Core digital VSS(0V)			
355		BOTTOM	pci_iopad0_uiov_lo_16	PCI66SDGZ	
	INOUT	AD[16] (PCIバス)			
356		BOTTOM	vss_io_b9	PVSS2DGZ	170
		IO digital VSS(0V)			
357		BOTTOM	pci_iopad0_uiov_lo_17	PCI66SDGZ	
	INOUT	AD[17] (PCIバス)			
358		BOTTOM	vdd_io_b11	PVDD2DGZ	45
		IO digital VDD(3.3V)			
359		BOTTOM	pci_iopad0_uiov_lo_18	PCI66SDGZ	
	INOUT	AD[18] (PCIバス)			
360		BOTTOM	vdd_core_b12	PVDD1DGZ	45
		Core digital VDD(1.0V)			
361		BOTTOM	pci_iopad0_uiov_lo_19	PCI66SDGZ	
	INOUT	AD[19] (PCIバス)			
362		BOTTOM	vss_core_b20	PVSS1DGZ	40
		Core digital VSS(0V)			
363		BOTTOM	pci_iopad0_uiov_lo_20	PCI66SDGZ	
	INOUT	AD[20] (PCIバス)			
364		BOTTOM	pci_iopad0_uiov_lo_21	PCI66SDGZ	
	INOUT	AD[21] (PCIバス)			
365		BOTTOM	pci_iopad0_uiov_lo_22	PCI66SDGZ	
	INOUT	AD[22] (PCIバス)			
366		BOTTOM	pci_iopad0_uiov_lo_23	PCI66SDGZ	
	INOUT	AD[23] (PCIバス)			
367		BOTTOM	pci_iopad0_uin1	PCI66SDGZ	
	INOUT	IDSEL(PCIバス)			
368		BOTTOM	vss_core_b19	PVSS1DGZ	40
		Core digital VSS(0V)			
369		BOTTOM	pci_iopad0_uiov2_lo_3	PCI66SDGZ	
	INOUT	C/BE[3](PCIバス)			
370		BOTTOM	pci_iopad0_uiov_lo_24	PCI66SDGZ	
	INOUT	AD[24] (PCIバス)			
371		BOTTOM	pci_iopad0_uiov_lo_25	PCI66SDGZ	
	INOUT	AD[25] (PCIバス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
372		BOTTOM	vdd_core.b11	PVDD1DGZ	45
		Core digital VDD(1.0V)			
373		BOTTOM	pci_iopad0_uiov_lo_26	PCI66SDGZ	
	INOUT	AD[26] (PCI バス)			
374		BOTTOM	vss_io.b8	PVSS2DGZ	170
		IO digital VSS(0V)			
375		BOTTOM	pci_iopad0_uiov_lo_27	PCI66SDGZ	
	INOUT	AD[27] (PCI バス)			
376		BOTTOM	vdd_io.b10	PVDD2DGZ	45
		IO digital VDD(3.3V)			
377		BOTTOM	pci_iopad0_uiov_lo_28	PCI66SDGZ	
	INOUT	AD[28] (PCI バス)			
378		BOTTOM	vss_core.b18	PVSS1DGZ	40
		Core digital VSS(0V)			
379		BOTTOM	pci_iopad0_uiov_lo_29	PCI66SDGZ	
	INOUT	AD[29] (PCI バス)			
380		BOTTOM	pci_iopad0_uiov_lo_30	PCI66SDGZ	
	INOUT	AD[30] (PCI バス)			
381		BOTTOM	pci_iopad0_uiov_lo_31	PCI66SDGZ	
	INOUT	AD[31] (PCI バス)			
382		BOTTOM	vdd_core.b10	PVDD1DGZ	45
		Core digital VDD(1.0V)			
383		BOTTOM	pci_iopad0_uout1	PCI66SDGZ	
	INOUT	REQ# (PCI バス)			
384		BOTTOM	vss_core.b17	PVSS1DGZ	40
		Core digital VSS(0V)			
385		BOTTOM	pci_iopad0_uin2	PCI66SDGZ	
	INOUT	GNT# (PCI バス)			
386		BOTTOM	vss_io.b7	PVSS2DGZ	170
		IO digital VSS(0V)			
387		BOTTOM	pci_iopad0_upclk	PCI66SDGZ	
	INOUT	CLK (PCI バス)			
388		BOTTOM	vdd_io.b9	PVDD2DGZ	45
		IO digital VDD(3.3V)			
389		BOTTOM	pci_iopad0_uprst	PCI66SDGZ	
	INOUT	RST# (PCI バス)			
390		BOTTOM	vss_core.b16	PVSS1DGZ	40
		Core digital VSS(0V)			
391		BOTTOM	pci_iopad0_pinta	PCI66SDGZ	
	INOUT	INTA# (PCI バス)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
392		BOTTOM	vdd_core.b9	PVDD1DGZ	45
		Core digital VDD(1.0V)			
393		BOTTOM	pci_prcut0	PRCUT2P	
		Power cut cell			
394		BOTTOM	sdram_iopad0_oe	pnl_sstl_2classi	
	OUTPUT	Output Enable < action:0 stop:1 > (SDRAM)			
395		BOTTOM	sdram_iopad0_dir	pnl_sstl_2classi	
	OUTPUT	Direction < read:1 write:0 > (SDRAM)			
396		BOTTOM	sdram_iopad0_dq127	pnl_sstl_2classi	
	INOUT	Dq[127] (SDRAM)			
397		BOTTOM	sdram_iopad0_dq063	pnl_sstl_2classi	
	INOUT	Dq[63] (SDRAM)			
398		BOTTOM	vss_gcs_b15	pnl_sstl_gcs	
		Core digital VSS(0V)			
399		BOTTOM	sdram_iopad0_dq123	pnl_sstl_2classi	
	INOUT	Dq[123] (SDRAM)			
400		BOTTOM	sdram_iopad0_dq059	pnl_sstl_2classi	
	INOUT	Dq[59] (SDRAM)			
401		BOTTOM	sdram_iopad0_dq126	pnl_sstl_2classi	
	INOUT	Dq[126] (SDRAM)			
402		BOTTOM	vdd_vc_b8	pnl_sstl_vc	
		Core digital VDD(1.0V)			
403		BOTTOM	sdram_iopad0_dq062	pnl_sstl_2classi	
	INOUT	Dq[62] (SDRAM)			
404		BOTTOM	vss_gcs_b14	pnl_sstl_gcs	
		Core digital VSS(0V)			
405		BOTTOM	sdram_iopad0_dq122	pnl_sstl_2classi	
	INOUT	Dq[122] (SDRAM)			
406		BOTTOM	vss_go_b6	pnl_sstl_go	
		IO digital VSS(0V)			
407		BOTTOM	sdram_iopad0_dq058	pnl_sstl_2classi	
	INOUT	Dq[58] (SDRAM)			
408		BOTTOM	vdd_vq_b8	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
409		BOTTOM	sdram_iopad0_dqm15	pnl_sstl_2classi	
	OUTPUT	Dm[15] (SDRAM)			
410		BOTTOM	vss_gcs_b13	pnl_sstl_gcs	
		Core digital VSS(0V)			
411		BOTTOM	sdram_iopad0_dqm07	pnl_sstl_2classi	
	OUTPUT	Dm[7] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
412		BOTTOM	sdram.iopad0_dqs015	pnl_sstl_2classi	
	INOUT	Dqs[15] (SDRAM)			
413		BOTTOM	sdram.iopad0_dqs007	pnl_sstl_2classi	
	INOUT	Dqs[7] (SDRAM)			
414		BOTTOM	vdd_vc_b7	pnl_sstl_vc	
		Core digital VDD(1.0V)			
415		BOTTOM	sdram.iopad0_dq125	pnl_sstl_2classi	
	INOUT	Dq[125] (SDRAM)			
416		BOTTOM	vss_gcs_b12	pnl_sstl_gcs	
		Core digital VSS(0V)			
417		BOTTOM	sdram.iopad0_dq061	pnl_sstl_2classi	
	INOUT	Dq[61] (SDRAM)			
418		BOTTOM	sdram.iopad0_dq121	pnl_sstl_2classi	
	INOUT	Dq[121] (SDRAM)			
419		BOTTOM	vdd_vp_b7	pnl_sstl_vp	
		SSTL IO digital VDD(2.5V)			
420		BOTTOM	sdram.iopad0_dq057	pnl_sstl_2classi	
	INOUT	Dq[57] (SDRAM)			
421		BOTTOM	sdram.iopad0_dq124	pnl_sstl_2classi	
	INOUT	Dq[124] (SDRAM)			
422		BOTTOM	vss_gcs_b11	pnl_sstl_gcs	
		Core digital VSS(0V)			
423		BOTTOM	sdram.iopad0_dq060	pnl_sstl_2classi	
	INOUT	Dq[60] (SDRAM)			
424		BOTTOM	vdd_vc_b6	pnl_sstl_vc	
		Core digital VDD(1.0V)			
425		BOTTOM	sdram.iopad0_dq120	pnl_sstl_2classi	
	INOUT	Dq[120] (SDRAM)			
426		BOTTOM	vss_go_b5	pnl_sstl_go	
		IO digital VSS(0V)			
427		BOTTOM	sdram.iopad0_dq056	pnl_sstl_2classi	
	INOUT	Dq[56] (SDRAM)			
428		BOTTOM	vdd_vq_b6	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
429		BOTTOM	sdram.iopad0_dq119	pnl_sstl_2classi	
	INOUT	Dq[119] (SDRAM)			
430		BOTTOM	vss_gcs_b10	pnl_sstl_gcs	
		Core digital VSS(0V)			
431		BOTTOM	sdram.iopad0_dq055	pnl_sstl_2classi	
	INOUT	Dq[55] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
432		BOTTOM	sdram.iopad0_dq115	pnl_sstl_2classi	
	INOUT	Dq[115] (SDRAM)			
433		BOTTOM	sdram.iopad0_dq051	pnl_sstl_2classi	
	INOUT	Dq[51] (SDRAM)			
434		BOTTOM	vss_gcs_b9	pnl_sstl_gcs	
		Core digital VSS(0V)			
435		BOTTOM	sdram.iopad0_dq118	pnl_sstl_2classi	
	INOUT	Dq[118] (SDRAM)			
436		BOTTOM	vdd_vc_b5	pnl_sstl_vc	
		Core digital VDD(1.0V)			
437		BOTTOM	sdram.iopad0_dq054	pnl_sstl_2classi	
	INOUT	Dq[54] (SDRAM)			
438		BOTTOM	vss_go_b4	pnl_sstl_go	
		IO digital VSS(0V)			
439		BOTTOM	sdram.iopad0_dq114	pnl_sstl_2classi	
	INOUT	Dq[114] (SDRAM)			
440		BOTTOM	vdd_vq_b5	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
441		BOTTOM	sdram.iopad0_dq050	pnl_sstl_2classi	
	INOUT	Dq[50] (SDRAM)			
442		BOTTOM	vss_gcs_b8	pnl_sstl_gcs	
		Core digital VSS(0V)			
443		BOTTOM	sdram.iopad0_dqm14	pnl_sstl_2classi	
	OUTPUT	Dm[14] (SDRAM)			
444		BOTTOM	sstl_vref_b0	pnl_sstl_vref	
		SSTL VREF(1.25V)			
445		BOTTOM	sdram.iopad0_dqm06	pnl_sstl_2classi	
	OUTPUT	Dm[6] (SDRAM)			
446		BOTTOM	vdd_vc_b4	pnl_sstl_vc	
		Core digital VDD(1.0V)			
447		BOTTOM	sdram.iopad0_dqs014	pnl_sstl_2classi	
	INOUT	Dqs[14] (SDRAM)			
448		BOTTOM	vss_gcs_b7	pnl_sstl_gcs	
		Core digital VSS(0V)			
449		BOTTOM	sdram.iopad0_dqs006	pnl_sstl_2classi	
	INOUT	Dqs[6] (SDRAM)			
450		BOTTOM	sdram.iopad0_dq117	pnl_sstl_2classi	
	INOUT	Dq[117] (SDRAM)			
451		BOTTOM	sdram.iopad0_dq053	pnl_sstl_2classi	
	INOUT	Dq[53] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
452		BOTTOM	vss_go.b3	pnl_sstl_go	
		IO digital VSS(0V)			
453		BOTTOM	sdram_iopad0_dq113	pnl_sstl_2classi	
	INOUT	Dq[113] (SDRAM)			
454		BOTTOM	vdd_vq.b4	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
455		BOTTOM	sdram_iopad0_dq049	pnl_sstl_2classi	
	INOUT	Dq[49] (SDRAM)			
456		BOTTOM	vss_gcs.b6	pnl_sstl_gcs	
		Core digital VSS(0V)			
457		BOTTOM	sdram_iopad0_dq116	pnl_sstl_2classi	
	INOUT	Dq[116] (SDRAM)			
458		BOTTOM	vdd_vc.b3	pnl_sstl_vc	
		Core digital VDD(1.0V)			
459		BOTTOM	sdram_iopad0_dq052	pnl_sstl_2classi	
	INOUT	Dq[52] (SDRAM)			
460		BOTTOM	sdram_iopad0_dq112	pnl_sstl_2classi	
	INOUT	Dq[112] (SDRAM)			
461		BOTTOM	vdd_vp.b3	pnl_sstl_vp	
		SSTL IO digital VDD(2.5V)			
462		BOTTOM	sdram_iopad0_dq048	pnl_sstl_2classi	
	INOUT	Dq[48] (SDRAM)			
463		BOTTOM	sdram_iopad0_dq111	pnl_sstl_2classi	
	INOUT	Dq[111] (SDRAM)			
464		BOTTOM	vss_gcs.b5	pnl_sstl_gcs	
		Core digital VSS(0V)			
465		BOTTOM	sdram_iopad0_dq047	pnl_sstl_2classi	
	INOUT	Dq[47] (SDRAM)			
466		BOTTOM	sdram_iopad0_dq107	pnl_sstl_2classi	
	INOUT	Dq[107] (SDRAM)			
467		BOTTOM	sdram_iopad0_dq043	pnl_sstl_2classi	
	INOUT	Dq[43] (SDRAM)			
468		BOTTOM	vdd_vc.b2	pnl_sstl_vc	
		Core digital VDD(1.0V)			
469		BOTTOM	sdram_iopad0_dq110	pnl_sstl_2classi	
	INOUT	Dq[110] (SDRAM)			
470		BOTTOM	vss_gcs.b4	pnl_sstl_gcs	
		Core digital VSS(0V)			
471		BOTTOM	sdram_iopad0_dq046	pnl_sstl_2classi	
	INOUT	Dq[46] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
472		BOTTOM	vss_go.b2	pnl_sstl_go	
		IO digital VSS(0V)			
473		BOTTOM	sdram_iopad0_dq106	pnl_sstl_2classi	
	INOUT	Dq[106] (SDRAM)			
474		BOTTOM	vdd_vq.b2	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
475		BOTTOM	sdram_iopad0_dq042	pnl_sstl_2classi	
	INOUT	Dq[42] (SDRAM)			
476		BOTTOM	vss_gcs.b3	pnl_sstl_gcs	
		Core digital VSS(0V)			
477		BOTTOM	sdram_iopad0_dqm13	pnl_sstl_2classi	
	OUTPUT	Dm[13] (SDRAM)			
478		BOTTOM	sdram_iopad0_dqm05	pnl_sstl_2classi	
	OUTPUT	Dm[5] (SDRAM)			
479		BOTTOM	sdram_iopad0_dqs013	pnl_sstl_2classi	
	INOUT	Dqs[13] (SDRAM)			
480		BOTTOM	vdd_vc.b1	pnl_sstl_vc	
		Core digital VDD(1.0V)			
481		BOTTOM	sdram_iopad0_dqs005	pnl_sstl_2classi	
	INOUT	Dqs[5] (SDRAM)			
482		BOTTOM	vss_gcs.b2	pnl_sstl_gcs	
		Core digital VSS(0V)			
483		BOTTOM	sdram_iopad0_dq109	pnl_sstl_2classi	
	INOUT	Dq[42] (SDRAM)			
484		BOTTOM	vss_go.b1	pnl_sstl_go	
		IO digital VSS(0V)			
485		BOTTOM	sdram_iopad0_dq045	pnl_sstl_2classi	
	INOUT	Dq[45] (SDRAM)			
486		BOTTOM	vdd_vq.b1	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
487		BOTTOM	sdram_iopad0_dq105	pnl_sstl_2classi	
	INOUT	Dq[105] (SDRAM)			
488		BOTTOM	vss_gcs.b1	pnl_sstl_gcs	
		Core digital VSS(0V)			
489		BOTTOM	sdram_iopad0_dq041	pnl_sstl_2classi	
	INOUT	Dq[41] (SDRAM)			
490		BOTTOM	sdram_iopad0_dq108	pnl_sstl_2classi	
	INOUT	Dq[108] (SDRAM)			
491		BOTTOM	sdram_iopad0_dq044	pnl_sstl_2classi	
	INOUT	Dq[44] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
492		BOTTOM	vdd_vc_b0	pnl_sstl_vc	
		Core digital VDD(1.0V)			
493		BOTTOM	sdram_iopad0_dq104	pnl_sstl_2classi	
	INOUT	Dq[104] (SDRAM)			
494		BOTTOM	vss_gcs_b0	pnl_sstl_gcs	
		Core digital VSS(0V)			
495		BOTTOM	sdram_iopad0_dq040	pnl_sstl_2classi	
	INOUT	Dq[40] (SDRAM)			
496		BOTTOM	sdram_iopad0_dq103	pnl_sstl_2classi	
	INOUT	Dq[103] (SDRAM)			
497		BOTTOM	sdram_iopad0_dq039	pnl_sstl_2classi	
	INOUT	Dq[39] (SDRAM)			
498		BOTTOM	vss_go_b0	pnl_sstl_go	
		IO digital VSS(0V)			
499		BOTTOM	sdram_iopad0_dq099	pnl_sstl_2classi	
	INOUT	Dq[99] (SDRAM)			
500		BOTTOM	vdd_vq_b0	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
501		BOTTOM	sdram_iopad0_dq035	pnl_sstl_2classi	
	INOUT	Dq[35] (SDRAM)			
502		BOTTOM	sdram_iopad0_dq102	pnl_sstl_2classi	
	INOUT	Dq[102] (SDRAM)			
503		RIGHT	sdram_iopad0_dq038	pnl_sstl_2classi	
	INOUT	Dq[38] (SDRAM)			
504		RIGHT	sdram_iopad0_dq098	pnl_sstl_2classi	
	INOUT	Dq[98] (SDRAM)			
505		RIGHT	sdram_iopad0_dq034	pnl_sstl_2classi	
	INOUT	Dq[34] (SDRAM)			
506		RIGHT	sdram_iopad0_dqm12	pnl_sstl_2classi	
	OUTPUT	Dm[12] (SDRAM)			
507		RIGHT	sdram_iopad0_dqm04	pnl_sstl_2classi	
	OUTPUT	Dm[4] (SDRAM)			
508		RIGHT	vdd_vp_r19	pnl_sstll_vp	
		SSTL IO digital VDD(2.5V)			
509		RIGHT	sdram_iopad0_dqs012	pnl_sstl_2classi	
	INOUT	Dqs[12] (SDRAM)			
510		RIGHT	sdram_iopad0_dqs004	pnl_sstl_2classi	
	INOUT	Dqs[14] (SDRAM)			
511		RIGHT	sdram_iopad0_dq101	pnl_sstl_2classi	
	INOUT	Dq[101] (SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
512		RIGHT	sdram_iopad0_dq037	pnl_sstl_2classi	
	INOUT	Dq[37] (SDRAM)			
513		RIGHT	vss_go.r15	pnl_sstl_go	
		IO digital VSS(0V)			
514		RIGHT	sdram_iopad0_dq097	pnl_sstl_2classi	
	INOUT	Dq[97] (SDRAM)			
515		RIGHT	vdd_vq.r18	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
516		RIGHT	sdram_iopad0_dq033	pnl_sstl_2classi	
	INOUT	Dq[33] (SDRAM)			
517		RIGHT	vdd_sstl.r19	pnl_sstl_vc	
		Core digital VDD(1.0V)			
518		RIGHT	sdram_iopad0_dq100	pnl_sstl_2classi	
	INOUT	Dq[100] (SDRAM)			
519		RIGHT	vss_gcs.r32	pnl_sstl_gcs	
		Core digital VSS(0V)			
520		RIGHT	sdram_iopad0_dq036	pnl_sstl_2classi	
	INOUT	Dq[36] (SDRAM)			
521		RIGHT	sdram_iopad0_dq096	pnl_sstl_2classi	
	INOUT	Dq[96] (SDRAM)			
522		RIGHT	sdram_iopad0_dq032	pnl_sstl_2classi	
	INOUT	Dq[32] (SDRAM)			
523		RIGHT	sdram_iopad0_cs1	pnl_sstl_2classi	
	OUTPUT	CS[1] (SDRAM)			
524		RIGHT	sdram_iopad0_cs0	pnl_sstl_2classi	
	OUTPUT	CS[0] (SDRAM)			
525		RIGHT	vss_gcs.r31	pnl_sstl_gcs	
		Core digital VSS(0V)			
526		RIGHT	sdram_iopad0_cas	pnl_sstl_2classi	
	OUTPUT	CAS (SDRAM)			
527		RIGHT	vdd_sstl.r18	pnl_sstl_vc	
		Core digital VDD(1.0V)			
528		RIGHT	sdram_iopad0_we	pnl_sstl_2classi	
	OUTPUT	WE (SDRAM)			
529		RIGHT	vss_go.r14	pnl_sstl_go	
		IO digital VSS(0V)			
530		RIGHT	sdram_iopad0_ras	pnl_sstl_2classi	
	OUTPUT	RAS (SDRAM)			
531		RIGHT	vdd_vq.r17	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
532		RIGHT	sdram_iopad0_bank0	pnl_sstl_2classi	
	OUTPUT	Ba[0] (SDRAM)			
533		RIGHT	vss_gcs_r30	pnl_sstl_gcs	
		Core digital VSS(0V)			
534		RIGHT	sdram_iopad0_bank1	pnl_sstl_2classi	
	OUTPUT	Ba[1] (SDRAM)			
535		RIGHT	sdram_iopad0_addr10	pnl_sstl_2classi	
	OUTPUT	A[10] (SDRAM)			
536		RIGHT	vdd_vp_r16	pnl_sstll_vp	
		SSTL IO digital VDD(2.5V)			
537		RIGHT	sdram_iopad0_addr00	pnl_sstl_2classi	
	OUTPUT	A[0] (SDRAM)			
538		RIGHT	sdram_iopad0_addr01	pnl_sstl_2classi	
	OUTPUT	A[1] (SDRAM)			
539		RIGHT	vss_gcs_r29	pnl_sstl_gcs	
		Core digital VSS(0V)			
540		RIGHT	sdram_iopad0_addr02	pnl_sstl_2classi	
	OUTPUT	A[2] (SDRAM)			
541		RIGHT	vdd_sstlr17	pnl_sstl_vc	
		Core digital VDD(1.0V)			
542		RIGHT	sdram_iopad0_addr03	pnl_sstl_2classi	
	OUTPUT	A[3] (SDRAM)			
543		RIGHT	vss_go_r13	pnl_sstl_go	
		IO digital VSS(0V)			
544		RIGHT	sdram_iopad0_addr04	pnl_sstl_2classi	
	OUTPUT	A[4] (SDRAM)			
545		RIGHT	vdd_vq_r15	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
546		RIGHT	sdram_iopad0_addr05	pnl_sstl_2classi	
	OUTPUT	A[5] (SDRAM)			
547		RIGHT	vss_gcs_r28	pnl_sstl_gcs	
		Core digital VSS(0V)			
548		RIGHT	sdram_iopad0_addr06	pnl_sstl_2classi	
	OUTPUT	A[6] (SDRAM)			
549		RIGHT	sdram_iopad0_addr07	pnl_sstl_2classi	
	OUTPUT	A[7] (SDRAM)			
550		RIGHT	sdram_iopad0_addr08	pnl_sstl_2classi	
	OUTPUT	A[8] (SDRAM)			
551		RIGHT	vdd_sstlr16	pnl_sstl_vc	
		Core digital VDD(1.0V)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
552		RIGHT	sdram.iopad0_addr09	pnl_sstl_2classi	
	OUTPUT	A[9] (SDRAM)			
553		RIGHT	sdram.iopad0_addr11	pnl_sstl_2classi	
	OUTPUT	A[11] (SDRAM)			
554		RIGHT	sdram.iopad0_addr12	pnl_sstl_2classi	
	OUTPUT	A[12] (SDRAM)			
555		RIGHT	vdd_vq_r14	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
556		RIGHT	sdram.iopad0_cke	pnl_sstl_2classi	
	OUTPUT	CKE (SDRAM)			
557		RIGHT	vss_go_r12	pnl_sstl_go	
		IO digital VSS(0V)			
558		RIGHT	sdram.iopad0_dq095	pnl_sstl_2classi	
	INOUT	Dq[95] (SDRAM)			
559		RIGHT	vss_gcs_r27	pnl_sstl_gcs	
		Core digital VSS(0V)			
560		RIGHT	sdram.iopad0_dq031	pnl_sstl_2classi	
	INOUT	Dq[31] (SDRAM)			
561		RIGHT	vdd_sstl_r15	pnl_sstl_vc	
		Core digital VDD(1.0V)			
562		RIGHT	sdram.iopad0_dq091	pnl_sstl_2classi	
	INOUT	Dq[91] (SDRAM)			
563		RIGHT	sdram.iopad0_dq027	pnl_sstl_2classi	
	INOUT	Dq[27] (SDRAM)			
564		RIGHT	sdram.iopad0_dq094	pnl_sstl_2classi	
	INOUT	Dq[94] (SDRAM)			
565		RIGHT	vss_gcs_r26	pnl_sstl_gcs	
		Core digital VSS(0V)			
566		RIGHT	sdram.iopad0_dq030	pnl_sstl_2classi	
	INOUT	Dq[30] (SDRAM)			
567		RIGHT	vss_go_r11	pnl_sstl_go	
		IO digital VSS(0V)			
568		RIGHT	sdram.iopad0_dq090	pnl_sstl_2classi	
	INOUT	Dq[90] (SDRAM)			
569		RIGHT	vdd_vq_r13	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
570		RIGHT	sdram.iopad0_dq026	pnl_sstl_2classi	
	INOUT	Dq[26] (SDRAM)			
571		RIGHT	vss_gcs_r25	pnl_sstl_gcs	
		Core digital VSS(0V)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
572		RIGHT	s dram_iopad0_dqm11	pnl_sstl_2classi	
	OUTPUT	Dm[11] (SDRAM)			
573		RIGHT	vdd_sstlr14	pnl_sstl_vc	
		Core digital VDD(1.0V)			
574		RIGHT	s dram_iopad0_dqm03	pnl_sstl_2classi	
	OUTPUT	Dm[3] (SDRAM)			
575		RIGHT	s dram_iopad0_dqs011	pnl_sstl_2classi	
	INOUT	Dqs[11] (SDRAM)			
576		RIGHT	s dram_iopad0_dqs003	pnl_sstl_2classi	
	INOUT	Dqs[3] (SDRAM)			
577		RIGHT	vss_gcs_r24	pnl_sstl_gcs	
		Core digital VSS(0V)			
578		RIGHT	s dram_iopad0_dq093	pnl_sstl_2classi	
	INOUT	Dq[93] (SDRAM)			
579		RIGHT	s dram_iopad0_dq029	pnl_sstl_2classi	
	INOUT	Dq[29] (SDRAM)			
580		RIGHT	sstl_vref_r0	pnl_sstl_vref	
		SSTL VREF(1.25V)			
581		RIGHT	s dram_iopad0_dq089	pnl_sstl_2classi	
	INOUT	Dq[89] (SDRAM)			
582		RIGHT	s dram_iopad0_dq025	pnl_sstl_2classi	
	INOUT	Dq[25] (SDRAM)			
583		RIGHT	vdd_sstlr13	pnl_sstl_vc	
		Core digital VDD(1.0V)			
584		RIGHT	s dram_iopad0_dq092	pnl_sstl_2classi	
	INOUT	Dq[92] (SDRAM)			
585		RIGHT	vss_gcs_r23	pnl_sstl_gcs	
		Core digital VSS(0V)			
586		RIGHT	s dram_iopad0_dq028	pnl_sstl_2classi	
	INOUT	Dq[28] (SDRAM)			
587		RIGHT	s dram_iopad0_dq088	pnl_sstl_2classi	
	INOUT	Dq[88] (SDRAM)			
588		RIGHT	vdd_vp_r12	pnl_sstll_vp	
		SSTL IO digital VDD(2.5V)			
589		RIGHT	s dram_iopad0_dq024	pnl_sstl_2classi	
	INOUT	Dq[24] (SDRAM)			
590		RIGHT	s dram_iopad0_dq087	pnl_sstl_2classi	
	INOUT	Dq[87] (SDRAM)			
591		RIGHT	vss_gcs_r22	pnl_sstl_gcs	
		Core digital VSS(0V)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
592		RIGHT	sdram_iopad0_dq023	pnl_sstl_2classi	
		INOUT	Dq[23] (SDRAM)		
593		RIGHT	vdd_sstlr12	pnl_sstl_vc	
			Core digital VDD(1.0V)		
594		RIGHT	sdram_iopad0_dq083	pnl_sstl_2classi	
		INOUT	Dq[83] (SDRAM)		
595		RIGHT	sdram_iopad0_dq019	pnl_sstl_2classi	
		INOUT	Dq[19] (SDRAM)		
596		RIGHT	sdram_iopad0_dq086	pnl_sstl_2classi	
		INOUT	Dq[86] (SDRAM)		
597		RIGHT	vss_gcs_r21	pnl_sstl_gcs	
			Core digital VSS(0V)		
598		RIGHT	sdram_iopad0_dq022	pnl_sstl_2classi	
		INOUT	Dq[22] (SDRAM)		
599		RIGHT	vdd_vq_r11	pnl_sstl_vq	
			SSTL IO digital VDD(2.5V)		
600		RIGHT	sdram_iopad0_dq082	pnl_sstl_2classi	
		INOUT	Dq[82] (SDRAM)		
601		RIGHT	vss_go_r10	pnl_sstl_go	
			IO digital VSS(0V)		
602		RIGHT	sdram_iopad0_dq018	pnl_sstl_2classi	
		INOUT	Dq[18] (SDRAM)		
603		RIGHT	vdd_sstlr11	pnl_sstl_vc	
			Core digital VDD(1.0V)		
604		RIGHT	sdram_iopad0_dqm10	pnl_sstl_2classi	
		OUTPUT	Dm[10] (SDRAM)		
605		RIGHT	vss_gcs_r20	pnl_sstl_gcs	
			Core digital VSS(0V)		
606		RIGHT	sdram_iopad0_dqm02	pnl_sstl_2classi	
		OUTPUT	Dm[2] (SDRAM)		
607		RIGHT	sdram_iopad0_dqs010	pnl_sstl_2classi	
		INOUT	Dqs[10] (SDRAM)		
608		RIGHT	vdd_vq_r10	pnl_sstll_vq	
			SSTL IO digital VDD(2.5V)		
609		RIGHT	sdram_iopad0_dqs002	pnl_sstl_2classi	
		INOUT	Dqs[2] (SDRAM)		
610		RIGHT	sdram_iopad0_dq085	pnl_sstl_2classi	
		INOUT	Dq[85] (SDRAM)		
611		RIGHT	vss_gcs_r19	pnl_sstl_gcs	
			Core digital VSS(0V)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
612		RIGHT	sdram_iopad0_dq021	pnl_sstl_2classi	
	INOUT	Dq[21] (SDRAM)			
613		RIGHT	vdd_sstlr10	pnl_sstl_vc	
		Core digital VDD(1.0V)			
614		RIGHT	sdram_iopad0_clk0_	pnl_sstl_2classi	
	OUTPUT	Clk0_ (SDRAM)			
615		RIGHT	vdd_vq_r9	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
616		RIGHT	sdram_iopad0_dq081	pnl_sstl_2classi	
	INOUT	Dq[81] (SDRAM)			
617		RIGHT	vss_go_r9	pnl_sstl_go	
		IO digital VSS(0V)			
618		RIGHT	sdram_iopad0_clk0	pnl_sstl_2classi	
	OUTPUT	Clk0 (SDRAM)			
619		RIGHT	vss_gcs_r18	pnl_sstl_gcs	
		Core digital VSS(0V)			
620		RIGHT	sdram_iopad0_dq017	pnl_sstl_2classi	
	INOUT	Dq[17] (SDRAM)			
621		RIGHT	sdram_iopad0_dq084	pnl_sstl_2classi	
	INOUT	Dq[84] (SDRAM)			
622		RIGHT	sdram_iopad0_dq020	pnl_sstl_2classi	
	INOUT	Dq[20] (SDRAM)			
623		RIGHT	vss_gcs_r17	pnl_sstl_gcs	
		Core digital VSS(0V)			
624		RIGHT	sdram_iopad0_dq080	pnl_sstl_2classi	
	INOUT	Dq[80] (SDRAM)			
625		RIGHT	sdram_iopad0_dq016	pnl_sstl_2classi	
	INOUT	Dq[16] (SDRAM)			
626		RIGHT	sdram_iopad0_dq079	pnl_sstl_2classi	
	INOUT	Dq[79] (SDRAM)			
627		RIGHT	vdd_sstlr9	pnl_sstl_vc	
		Core digital VDD(1.0V)			
628		RIGHT	sdram_iopad0_clk1_	pnl_sstl_2classi	
	OUTPUT	Clk1_ (SDRAM)			
629		RIGHT	vss_gcs_r16	pnl_sstl_gcs	
		Core digital VSS(0V)			
630		RIGHT	sdram_iopad0_dq015	pnl_sstl_2classi	
	INOUT	Dq[15] (SDRAM)			
631		RIGHT	vdd_vq_r8	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
632		RIGHT	s dram_iopad0_clk1	pnl_sstl_2classi	
	OUTPUT	Clk1 (SDRAM)			
633		RIGHT	vss_go_r8	pnl_sstl_go	
		IO digital VSS(0V)			
634		RIGHT	s dram_iopad0_dq075	pnl_sstl_2classi	
	INOUT	Dq[75] (SDRAM)			
635		RIGHT	s dram_iopad0_dq011	pnl_sstl_2classi	
	INOUT	Dq[11] (SDRAM)			
636		RIGHT	s dram_iopad0_dq078	pnl_sstl_2classi	
	INOUT	Dq[78] (SDRAM)			
637		RIGHT	vss_gcs_r15	pnl_sstl_gcs	
		Core digital VSS(0V)			
638		RIGHT	s dram_iopad0_dq014	pnl_sstl_2classi	
	INOUT	Dq[14] (SDRAM)			
639		RIGHT	vdd_sstl_r8	pnl_sstl_vc	
		Core digital VDD(1.0V)			
640		RIGHT	s dram_iopad0_dq074	pnl_sstl_2classi	
	INOUT	Dq[74] (SDRAM)			
641		RIGHT	s dram_iopad0_dq010	pnl_sstl_2classi	
	INOUT	Dq[10] (SDRAM)			
642		RIGHT	s dram_iopad0_dqm09	pnl_sstl_2classi	
	OUTPUT	Dm[9] (SDRAM)			
643		RIGHT	vss_gcs_r14	pnl_sstl_gcs	
		Core digital VSS(0V)			
644		RIGHT	s dram_iopad0_dqm01	pnl_sstl_2classi	
	OUTPUT	Dm[1] (SDRAM)			
645		RIGHT	vdd_vq_r7	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
646		RIGHT	s dram_iopad0_dqs009	pnl_sstl_2classi	
	INOUT	Dqs[9] (SDRAM)			
647		RIGHT	vss_go_r7	pnl_sstl_go	
		IO digital VSS(0V)			
648		RIGHT	s dram_iopad0_dqs001	pnl_sstl_2classi	
	INOUT	Dqs[1] (SDRAM)			
649		RIGHT	vdd_sstl_r7	pnl_sstl_vc	
		Core digital VDD(1.0V)			
650		RIGHT	s dram_iopad0_dq077	pnl_sstl_2classi	
	INOUT	Dq[77] (SDRAM)			
651		RIGHT	vss_gcs_r13	pnl_sstl_gcs	
		Core digital VSS(0V)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
652		RIGHT	s dram_iopad0_dq013	pnl_sstl_2classi	
	INOUT	Dq[13] (SDRAM)			
653		RIGHT	s dram_iopad0_dq073	pnl_sstl_2classi	
	INOUT	Dq[73] (SDRAM)			
654		RIGHT	sstl_vref_r1	pnl_sstl_vref	
		SSTL VREF(1.25V)			
655		RIGHT	s dram_iopad0_dq009	pnl_sstl_2classi	
	INOUT	Dq[9] (SDRAM)			
656		RIGHT	s dram_iopad0_dq076	pnl_sstl_2classi	
	INOUT	Dq[76] (SDRAM)			
657		RIGHT	vss_gcs_r12	pnl_sstl_gcs	
		Core digital VSS(0V)			
658		RIGHT	s dram_iopad0_dq012	pnl_sstl_2classi	
	INOUT	Dq[12] (SDRAM)			
659		RIGHT	vdd_sstl_r6	pnl_sstl_vc	
		Core digital VDD(1.0V)			
660		RIGHT	s dram_iopad0_dq072	pnl_sstl_2classi	
	INOUT	Dq[72] (SDRAM)			
661		RIGHT	vdd_vq_r6	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
662		RIGHT	s dram_iopad0_dq008	pnl_sstl_2classi	
	INOUT	Dq[8] (SDRAM)			
663		RIGHT	vss_go_r6	pnl_sstl_go	
		IO digital VSS(0V)			
664		RIGHT	s dram_iopad0_dq071	pnl_sstl_2classi	
	INOUT	Dq[71] (SDRAM)			
665		RIGHT	vss_gcs_r11	pnl_sstl_gcs	
		Core digital VSS(0V)			
666		RIGHT	s dram_iopad0_dq007	pnl_sstl_2classi	
	INOUT	Dq[7] (SDRAM)			
667		RIGHT	s dram_iopad0_dq067	pnl_sstl_2classi	
	INOUT	Dq[67] (SDRAM)			
668		RIGHT	s dram_iopad0_dq003	pnl_sstl_2classi	
	INOUT	Dq[3] (SDRAM)			
669		RIGHT	vss_gcs_r10	pnl_sstl_gcs	
		Core digital VSS(0V)			
670		RIGHT	s dram_iopad0_dq070	pnl_sstl_2classi	
	INOUT	Dq[70] (SDRAM)			
671		RIGHT	vdd_sstl_r5	pnl_sstl_vc	
		Core digital VDD(1.0V)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
672		RIGHT	s dram_iopad0_dq006	pnl_sstl_2classi	
		INOUT	Dq[6] (SDRAM)		
673		RIGHT	s dram_iopad0_dq066	pnl_sstl_2classi	
		INOUT	Dq[66] (SDRAM)		
674		RIGHT	s dram_iopad0_dq002	pnl_sstl_2classi	
		INOUT	Dq[2] (SDRAM)		
675		RIGHT	vss_gcs_r9	pnl_sstl_gcs	
			Core digital VSS(0V)		
676		RIGHT	s dram_iopad0_dqm08	pnl_sstl_2classi	
		OUTPUT	Dm[8] (SDRAM)		
677		RIGHT	vdd_vq_r5	pnl_sstl_vq	
			SSTL IO digital VDD(2.5V)		
678		RIGHT	s dram_iopad0_dqm00	pnl_sstl_2classi	
		OUTPUT	Dm[0] (SDRAM)		
679		RIGHT	vss_go_r5	pnl_sstl_go	
			IO digital VSS(0V)		
680		RIGHT	s dram_iopad0_dqs008	pnl_sstl_2classi	
		INOUT	Dqs[8] (SDRAM)		
681		RIGHT	vdd_sstl_r4	pnl_sstl_vc	
			Core digital VDD(1.0V)		
682		RIGHT	s dram_iopad0_dqs000	pnl_sstl_2classi	
		INOUT	Dqs[0] (SDRAM)		
683		RIGHT	vss_gcs_r8	pnl_sstl_gcs	
			Core digital VSS(0V)		
684		RIGHT	s dram_iopad0_dq069	pnl_sstl_2classi	
		INOUT	Dq[69] (SDRAM)		
685		RIGHT	s dram_iopad0_dq005	pnl_sstl_2classi	
		INOUT	Dq[5] (SDRAM)		
686		RIGHT	s dram_iopad0_dq065	pnl_sstl_2classi	
		INOUT	Dq[65] (SDRAM)		
687		RIGHT	vss_go_r4	pnl_sstl_go	
			IO digital VSS(0V)		
688		RIGHT	s dram_iopad0_dq001	pnl_sstl_2classi	
		INOUT	Dq[1] (SDRAM)		
689		RIGHT	vss_gcs_r7	pnl_sstl_gcs	
			Core digital VSS(0V)		
690		RIGHT	s dram_iopad0_dq068	pnl_sstl_2classi	
		INOUT	Dq[68] (SDRAM)		
691		RIGHT	vdd_sstl_r3	pnl_sstl_vc	
			Core digital VDD(1.0V)		

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
692		RIGHT	s dram.iopad0_dq004	pnl_sstl_2classi	
		INOUT	Dq[4] (SDRAM)		
693		RIGHT	vss_go.r3	pnl_sstl_go	
			IO digital VSS(0V)		
694		RIGHT	s dram.iopad0_dq064	pnl_sstl_2classi	
		INOUT	Dq[64] (SDRAM)		
695		RIGHT	vdd_vq.r4	pnl_sstl_vq	
			SSTL IO digital VDD(2.5V)		
696		RIGHT	s dram.iopad0_dq000	pnl_sstl_2classi	
		INOUT	Dq[0] (SDRAM)		
697		RIGHT	vss_gcs.r6	pnl_sstl_gcs	
			Core digital VSS(0V)		
698		RIGHT	link_s dram.iopad0_oe	pnl_sstl_2classi	
		OUTPUT	Output Enable (action:0 stop:1) (LINK SDRAM)		
699		RIGHT	link_s dram.iopad0_dir	pnl_sstl_2classi	
		OUTPUT	Direction (read:1 write:0) (LINK SDRAM)		
700		RIGHT	link_s dram.iopad0_addr04	pnl_sstl_2classi	
		OUTPUT	A[4] (LINK SDRAM)		
701		RIGHT	link_s dram.iopad0_addr03	pnl_sstl_2classi	
		OUTPUT	A[3] (LINK SDRAM)		
702		RIGHT	link_s dram.iopad0_addr05	pnl_sstl_2classi	
		OUTPUT	A[5] (LINK SDRAM)		
703		RIGHT	vss_gcs.r5	pnl_sstl_gcs	
			Core digital VSS(0V)		
704		RIGHT	link_s dram.iopad0_addr02	pnl_sstl_2classi	
		OUTPUT	A[2] (LINK SDRAM)		
705		RIGHT	vdd_sstl.r2	pnl_sstl_vc	
			Core digital VDD(1.0V)		
706		RIGHT	link_s dram.iopad0_addr06	pnl_sstl_2classi	
		OUTPUT	A[6] (LINK SDRAM)		
707		RIGHT	vss_go.r2	pnl_sstl_go	
			IO digital VSS(0V)		
708		RIGHT	link_s dram.iopad0_addr01	pnl_sstl_2classi	
		OUTPUT	A[1] (LINK SDRAM)		
709		RIGHT	vdd_vq.r2	pnl_sstl_vq	
			SSTL IO digital VDD(2.5V)		
710		RIGHT	link_s dram.iopad0_addr07	pnl_sstl_2classi	
		OUTPUT	A[7] (LINK SDRAM)		
711		RIGHT	vss_gcs.r4	pnl_sstl_gcs	
			Core digital VSS(0V)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
712		RIGHT	link_sdram_iopad0_addr00	pnl_sstl_2classi	
	OUTPUT	A[0] (LINK SDRAM)			
713		RIGHT	link_sdram_iopad0_addr08	pnl_sstl_2classi	
	OUTPUT	A[8] (LINK SDRAM)			
714		RIGHT	link_sdram_iopad0_addr10	pnl_sstl_2classi	
	OUTPUT	A[10] (LINK SDRAM)			
715		RIGHT	vdd_sstlr1	pnl_sstl_vc	
		Core digital VDD(1.0V)			
716		RIGHT	link_sdram_iopad0_addr09	pnl_sstl_2classi	
	OUTPUT	A[9] (LINK SDRAM)			
717		RIGHT	vss_gcs_r3	pnl_sstl_gcs	
		Core digital VSS(0V)			
718		RIGHT	link_sdram_iopad0_bank1	pnl_sstl_2classi	
	OUTPUT	Ba[1] (LINK SDRAM)			
719		RIGHT	vss_go_r1	pnl_sstl_go	
		IO digital VSS(0V)			
720		RIGHT	link_sdram_iopad0_addr11	pnl_sstl_2classi	
	OUTPUT	A[11] (LINK SDRAM)			
721		RIGHT	vdd_vq_r1	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
722		RIGHT	link_sdram_iopad0_bank0	pnl_sstl_2classi	
	OUTPUT	Ba[0] (LINK SDRAM)			
723		RIGHT	vss_gcs_r2	pnl_sstl_gcs	
		Core digital VSS(0V)			
724		RIGHT	link_sdram_iopad0_addr12	pnl_sstl_2classi	
	OUTPUT	A[12] (LINK SDRAM)			
725		RIGHT	link_sdram_iopad0_cs1	pnl_sstl_2classi	
	OUTPUT	CS[1] (LINK SDRAM)			
726		RIGHT	link_sdram_iopad0_cs0	pnl_sstl_2classi	
	OUTPUT	CS[0] (LINK SDRAM)			
727		RIGHT	link_sdram_iopad0_cke	pnl_sstl_2classi	
	OUTPUT	CKE (LINK SDRAM)			
728		RIGHT	lvds_vref_r0	pnl_vref_lvds	
		LVDS VREF(1.2V)			
729		RIGHT	vdd_vc_r0	pnl_vc_lvds	
		Core digital VDD(1.0V)			
730		RIGHT	link_iopad0_data_s_out_iopad1_p	pnl_lvds85_out_gcs	
	OUTPUT	Serial data output1+ (LINK)			
731		RIGHT	link_iopad0_data_s_out_iopad1	pnl_lvds85_out_gcs	
		Core digital VSS(0V)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
732		RIGHT	link_iopad0_data_s_out_iopad1_n	pnl_lvds85_out_gcs	
		OUTPUT	Serial data output1- (LINK)		
733		RIGHT	vss_go_r0	pnl_go_lvds	
			IO digital VSS(0V)		
734		RIGHT	link_iopad0_data_s_out_iopad2_p	pnl_lvds85_out_vop	
		OUTPUT	Serial data output2+ (LINK)		
735		RIGHT	link_iopad0_data_s_out_iopad2	pnl_lvds85_out_vop	
			LVDS IO digital VDD(3.3V)		
736		RIGHT	link_iopad0_data_s_out_iopad2_n	pnl_lvds85_out_vop	
		OUTPUT	Serial data output2- (LINK)		
737		RIGHT	vss_gcs_r1	pnl_gcs_lvds	
			Core digital VSS(0V)		
738		RIGHT	link_iopad0_data_s_in_iopad1_p	pnl_lvds85_se_in	
		INPUT	Serial data input1+ (LINK)		
739		RIGHT	link_iopad0_data_s_in_iopad1_n	pnl_lvds85_se_in	
		INPUT	Serial data input1- (LINK)		
740		RIGHT	link_iopad0_data_s_out_iopad3_p	pnl_lvds85_out_yc	
		OUTPUT	Serial data output3+ (LINK)		
741		RIGHT	link_iopad0_data_s_out_iopad3	pnl_lvds85_out_yc	
			Core digital VDD(1.0V)		
742		RIGHT	link_iopad0_data_s_out_iopad3_n	pnl_lvds85_out_yc	
		OUTPUT	Serial data output3- (LINK)		
743		RIGHT	vss_gcs_r0	pnl_gcs_lvds	
			Core digital VSS(0V)		
744		RIGHT	link_iopad0_data_s_out_iopad4_p	pnl_lvds85_out_go	
		OUTPUT	Serial data output4+ (LINK)		
745		RIGHT	link_iopad0_data_s_out_iopad4	pnl_lvds85_out_go	
			IO digital VSS(0V)		
746		RIGHT	link_iopad0_data_s_out_iopad4_n	pnl_lvds85_out_go	
		OUTPUT	Serial data output4- (LINK)		
747		RIGHT	vdd_vop_r0	pnl_vop_lvds	
			LVDS IO digital VDD(3.3V)		
748		RIGHT	link_iopad0_data_s_in_iopad2_p	pnl_lvds85_se_in	
		INPUT	Serial data input2+ (LINK)		
749		RIGHT	link_iopad0_data_s_in_iopad2_n	pnl_lvds85_se_in	
		INPUT	Serial data input2- (LINK)		
750		RIGHT	link_iopad0_data_s_in_iopad3_p	pnl_lvds85_se_in	
		INPUT	Serial data input3+ (LINK)		
751		RIGHT	link_iopad0_data_s_in_iopad3_n	pnl_lvds85_se_in	
		INPUT	Serial data input3- (LINK)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
752		RIGHT	link_iopad0_data_s_in_iopad4_p	pnl_lvds85_se.in	
	INPUT	Serial data input4+ (LINK)			
753		RIGHT	link_iopad0_data_s_in_iopad4_n	pnl_lvds85_se.in	
	INPUT	Serial data input4- (LINK)			
754		TOP	link_iopad0_event_s_in_iopad4_p	pnl_lvds85_se.in	
	INPUT	Serial event input4+ (LINK)			
755		TOP	link_iopad0_event_s_in_iopad4_n	pnl_lvds85_se.in	
	INPUT	Serial event input4- (LINK)			
756		TOP	vss_go.t13	pnl_go_lvds	
		IO digital VSS(0V)			
757		TOP	link_iopad0_event_s_out_pad4_p	pnl_lvds85_out_vop_p	
	OUTPUT	Serial event output4+ (LINK)			
758		TOP	link_iopad0_event_s_out_pad4	pnl_lvds85_out_vop	
		LVDS IO digital VDD(3.3V)			
759		TOP	link_iopad0_event_s_out_pad4_n	pnl_lvds85_out_vop_n	
	OUTPUT	Serial event output4- (LINK)			
760		TOP	link_iopad0_event_s_in_iopad3_p	pnl_lvds85_se.in_p	
	INPUT	Serial event input3+ (LINK)			
761		TOP	link_iopad0_event_s_in_iopad3_n	pnl_lvds85_se.in_n	
	INPUT	Serial event input3- (LINK)			
762		TOP	vss_gcs.t44	pnl_gcs_lvds	
		Core digital VSS(0V)			
763		TOP	link_iopad0_event_s_in_iopad2_p	pnl_lvds85_se.in_p	
	INPUT	Serial event input2+ (LINK)			
764		TOP	link_iopad0_event_s_in_iopad2_n	pnl_lvds85_se.in_n	
	INPUT	Serial event input2- (LINK)			
765		TOP	link_iopad0_event_s_out_pad3_p	pnl_lvds85_out_vc_p	
	OUTPUT	Serial event output3+ (LINK)			
766		TOP	link_iopad0_event_s_out_pad3	pnl_lvds85_out_vc	
		Core digital VDD(1.0V)			
767		TOP	link_iopad0_event_s_out_pad3_n	pnl_lvds85_out_vc_n	
	OUTPUT	Serial event output3- (LINK)			
768		TOP	vss_gcs.t43	pnl_gcs_lvds	
		Core digital VSS(0V)			
769		TOP	link_iopad0_event_s_out_pad2_p	pnl_lvds85_out_go_p	
	OUTPUT	Serial event output2+ (LINK)			
770		TOP	link_iopad0_event_s_out_pad2	pnl_lvds85_out_go	
		IO digital VSS(0V)			
771		TOP	link_iopad0_event_s_out_pad2_n	pnl_lvds85_out_go_n	
	OUTPUT	Serial event output2- (LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
772		TOP	vdd_vop.t14	pnl_vop_lvds	
			LVDS IO digital VDD(3.3V)		
773		TOP	link_iopad0_event_s_in_iopad1_p	pnl_lvds85_se.in.p	
		INPUT	Serial event input1+ (LINK)		
774		TOP	link_iopad0_event_s_in_iopad1_n	pnl_lvds85_se.in.n	
		INPUT	Serial event input1- (LINK)		
775		TOP	link_iopad0_event_s_out_pad1_p	pnl_lvds85_out_gcs.p	
		OUTPUT	Serial event output1+ (LINK)		
776		TOP	link_iopad0_event_s_out_pad1	pnl_lvds85_out_gcs	
			Core digital VSS(0V)		
777		TOP	link_iopad0_event_s_out_pad1_n	pnl_lvds85_out_gcs.n	
		OUTPUT	Serial event output1- (LINK)		
778		TOP	vdd_sstl_vc.t25	pnl_vc_lvds	
			Core digital VDD(1.0V)		
779		TOP	link_sdram_iopad0_ras	pnl_sstl_2classi	
		OUTPUT	RAS (LINK SDRAM)		
780		TOP	vss_go.t12	pnl_sstl_go	
			IO digital VSS(0V)		
781		TOP	link_sdram_iopad0_cas	pnl_sstl_2classi	
		OUTPUT	CAS (LINK SDRAM)		
782		TOP	vdd_vq.t13	pnl_sstl_vq	
			SSTL IO digital VDD(2.5V)		
783		TOP	link_sdram_iopad0_we	pnl_sstl_2classi	
		OUTPUT	WE (LINK SDRAM)		
784		TOP	vss_gcs.t42	pnl_sstl_gcs	
			Core digital VSS(0V)		
785		TOP	link_sdram_iopad0_dqm3	pnl_sstl_2classi	
		OUTPUT	Dm[3] (LINK SDRAM)		
786		TOP	link_sdram_iopad0_dqm1	pnl_sstl_2classi	
		OUTPUT	Dm[1] (LINK SDRAM)		
787		TOP	link_sdram_iopad0_dqm2	pnl_sstl_2classi	
		OUTPUT	Dm[2] (LINK SDRAM)		
788		TOP	vdd_vc.t24	pnl_sstl_vc	
			Core digital VDD(1.0V)		
789		TOP	link_sdram_iopad0_dqm0	pnl_sstl_2classi	
		OUTPUT	Dm[0] (LINK SDRAM)		
790		TOP	vss_gcs.t41	pnl_sstl_gcs	
			Core digital VSS(0V)		
791		TOP	link_sdram_iopad0_clk	pnl_sstl_2classi	
		OUTPUT	Clk (LINK SDRAM)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
792		TOP	vss_go.t11	pnl_sstl_go	
		IO digital VSS(0V)			
793		TOP	link_sdram_iopad0_dqs3	pnl_sstl_2classi	
	INOUT	Dqs[3] (LINK SDRAM)			
794		TOP	vdd_vp.t12	pnl_sstl_vp	
		SSTL IO digital VDD(2.5V)			
795		TOP	link_sdram_iopad0_dqs1	pnl_sstl_2classi	
	INOUT	Dqs[1] (LINK SDRAM)			
796		TOP	link_sdram_iopad0_dqs2	pnl_sstl_2classi	
	INOUT	Dqs[2] (LINK SDRAM)			
797		TOP	link_sdram_iopad0_dqs0	pnl_sstl_2classi	
	INOUT	Dqs[0] (LINK SDRAM)			
798		TOP	vss_gcs.t40	pnl_sstl_gcs	
		Core digital VSS(0V)			
799		TOP	link_sdram_iopad0_dq24	pnl_sstl_2classi	
	INOUT	Dq[24] (LINK SDRAM)			
800		TOP	vdd_vc.t23	pnl_sstl_vc	
		Core digital VDD(1.0V)			
801		TOP	link_sdram_iopad0_dq08	pnl_sstl_2classi	
	INOUT	Dq[8] (LINK SDRAM)			
802		TOP	link_sdram_iopad0_dq23	pnl_sstl_2classi	
	INOUT	Dq[23] (LINK SDRAM)			
803		TOP	link_sdram_iopad0_dq07	pnl_sstl_2classi	
	INOUT	Dq[7] (LINK SDRAM)			
804		TOP	vss_gcs.t39	pnl_sstl_gcs	
		Core digital VSS(0V)			
805		TOP	link_sdram_iopad0_clk_	pnl_sstl_2classi	
	OUTPUT	Clk_ (LINK SDRAM)			
806		TOP	vss_go.t10	pnl_sstl_go	
		IO digital VSS(0V)			
807		TOP	link_sdram_iopad0_dq25	pnl_sstl_2classi	
	INOUT	Dq[25] (LINK SDRAM)			
808		TOP	vdd_vq.t11	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
809		TOP	link_sdram_iopad0_dq09	pnl_sstl_2classi	
	INOUT	Dq[9] (LINK SDRAM)			
810		TOP	vss_gcs.t38	pnl_sstl_gcs	
		Core digital VSS(0V)			
811		TOP	link_sdram_iopad0_dq22	pnl_sstl_2classi	
	INOUT	Dq[22] (LINK SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
812		TOP	vdd_vc.t22	pnl_sstl_vc	
		Core digital VDD(1.0V)			
813		TOP	link_sdram_iopad0_dq06	pnl_sstl_2classi	
	INOUT	Dq[6] (LINK SDRAM)			
814		TOP	sstl_vref.t0	pnl_sstl_vref	
		SSTL VREF(1.25V)			
815		TOP	link_sdram_iopad0_dq26	pnl_sstl_2classi	
	INOUT	Dq[26] (LINK SDRAM)			
816		TOP	vss_gcs.t37	pnl_sstl_gcs	
		Core digital VSS(0V)			
817		TOP	link_sdram_iopad0_dq10	pnl_sstl_2classi	
	INOUT	Dq[10] (LINK SDRAM)			
818		TOP	vss_go.t9	pnl_sstl_go	
		IO digital VSS(0V)			
819		TOP	link_sdram_iopad0_dq21	pnl_sstl_2classi	
	INOUT	Dq[21] (LINK SDRAM)			
820		TOP	vdd_vq.t10	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
821		TOP	link_sdram_iopad0_dq05	pnl_sstl_2classi	
	INOUT	Dq[5] (LINK SDRAM)			
822		TOP	vss_gcs.t36	pnl_sstl_gcs	
		Core digital VSS(0V)			
823		TOP	link_sdram_iopad0_dq27	pnl_sstl_2classi	
	INOUT	Dq[27] (LINK SDRAM)			
824		TOP	vdd_vc.t21	pnl_sstl_vc	
		Core digital VDD(1.0V)			
825		TOP	link_sdram_iopad0_dq11	pnl_sstl_2classi	
	INOUT	Dq[11] (LINK SDRAM)			
826		TOP	link_sdram_iopad0_dq20	pnl_sstl_2classi	
	INOUT	Dq[20] (LINK SDRAM)			
827		TOP	link_sdram_iopad0_dq04	pnl_sstl_2classi	
	INOUT	Dq[4] (LINK SDRAM)			
828		TOP	vss_gcs.t35	pnl_sstl_gcs	
		Core digital VSS(0V)			
829		TOP	link_sdram_iopad0_dq28	pnl_sstl_2classi	
	INOUT	Dq[28] (LINK SDRAM)			
830		TOP	link_sdram_iopad0_dq12	pnl_sstl_2classi	
	INOUT	Dq[12] (LINK SDRAM)			
831		TOP	link_sdram_iopad0_dq19	pnl_sstl_2classi	
	INOUT	Dq[19] (LINK SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
832		TOP	vddl_vc.t20	pnl_sstl_vc	
		Core digital VDD(1.0V)			
833		TOP	link_sdram_iopad0_dq03	pnl_sstl_2classi	
	INOUT	Dq[3] (LINK SDRAM)			
834		TOP	vss_gcs.t34	pnl_sstl_gcs	
		Core digital VSS(0V)			
835		TOP	link_sdram_iopad0_dq29	pnl_sstl_2classi	
	INOUT	Dq[29] (LINK SDRAM)			
836		TOP	vss_go.t8	pnl_sstl_go	
		IO digital VSS(0V)			
837		TOP	link_sdram_iopad0_dq13	pnl_sstl_2classi	
	INOUT	Dq[13] (LINK SDRAM)			
838		TOP	vdd_vq.t8	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
839		TOP	link_sdram_iopad0_dq18	pnl_sstl_2classi	
	INOUT	Dq[18] (LINK SDRAM)			
840		TOP	link_sdram_iopad0_dq02	pnl_sstl_2classi	
	INOUT	Dq[2] (LINK SDRAM)			
841		TOP	link_sdram_iopad0_dq30	pnl_sstl_2classi	
	INOUT	Dq[30] (LINK SDRAM)			
842		TOP	vss_gcs.t33	pnl_sstl_gcs	
		Core digital VSS(0V)			
843		TOP	link_sdram_iopad0_dq14	pnl_sstl_2classi	
	INOUT	Dq[14] (LINK SDRAM)			
844		TOP	vdd_vc.t19	pnl_sstl_vc	
		Core digital VDD(1.0V)			
845		TOP	link_sdram_iopad0_dq17	pnl_sstl_2classi	
	INOUT	Dq[17] (LINK SDRAM)			
846		TOP	vss_go.t7	pnl_sstl_go	
		IO digital VSS(0V)			
847		TOP	link_sdram_iopad0_dq01	pnl_sstl_2classi	
	INOUT	Dq[1] (LINK SDRAM)			
848		TOP	vdd_vq.t7	pnl_sstl_vq	
		SSTL IO digital VDD(2.5V)			
849		TOP	link_sdram_iopad0_dq31	pnl_sstl_2classi	
	INOUT	Dq[31] (LINK SDRAM)			
850		TOP	vss_sstl_gcs.t32	pnl_sstl_gcs	
		Core digital VSS(0V)			
851		TOP	link_sdram_iopad0_dq15	pnl_sstl_2classi	
	INOUT	Dq[15] (LINK SDRAM)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
852		TOP	link_sdram_iopad0_dq16	pnl_sstl2classi	
	INOUT	Dq[16] (LINK SDRAM)			
853		TOP	link_sdram_iopad0_dq00	pnl_sstl2classi	
	INOUT	Dq[0] (LINK SDRAM)			
854		TOP	link_iopad0_data_p_in4_iopad3	pnl_it2nm8	
	INPUT	Parallel data input4.3 (LINK)			
855		TOP	link_iopad0_data_p_in4_iopad2	pnl_it2nm8	
	INPUT	Parallel data input4.2 (LINK)			
856		TOP	vss_gcs_t31	pnl_gcs	135
		Core digital VSS(0V)			
857		TOP	link_iopad0_data_p_in4_iopad1	pnl_it2nm8	
	INPUT	Parallel data input4.1 (LINK)			
858		TOP	vss_go_t6	pnl_go	135
		IO digital VSS(0V)			
859		TOP	link_iopad0_data_p_out4_iopad3	pnl_tf12it0nm8	
	OUTPUT	Parallel data output4.3 (LINK)			
860		TOP	vdd_vop_t6	pnl_vop	135
		IO digital VDD(3.3V)			
861		TOP	link_iopad0_data_p_out4_iopad2	pnl_tf12it0nm8	
	OUTPUT	Parallel data output4.2 (LINK)			
862		TOP	vdd_vc_t18	pnl_vc	135
		Core digital VDD(1.0V)			
863		TOP	link_iopad0_data_p_out4_iopad1	pnl_tf12it0nm8	
	OUTPUT	Parallel data output4.1 (LINK)			
864		TOP	vss_gcs_t30	pnl_gcs	135
		Core digital VSS(0V)			
865		TOP	link_iopad0_data_p_in3_iopad3	pnl_it2nm8	
	INPUT	Parallel data input3.3 (LINK)			
866		TOP	link_iopad0_data_p_in3_iopad2	pnl_it2nm8	
	INPUT	Parallel data input3.2 (LINK)			
867		TOP	link_iopad0_data_p_in3_iopad1	pnl_it2nm8	
	INPUT	Parallel data input3.1 (LINK)			
868		TOP	vdd_vc_t17	pnl_vc	135
		Core digital VDD(1.0V)			
869		TOP	link_iopad0_data_p_out3_iopad3	pnl_tf12it0nm8	
	OUTPUT	Parallel data output3.3 (LINK)			
870		TOP	vss_gcs_t29	pnl_gcs	135
		Core digital VSS(0V)			
871		TOP	link_iopad0_data_p_out3_iopad2	pnl_tf12it0nm8	
	OUTPUT	Parallel data output3.2 (LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
872		TOP	vss_go.t5	pnl_go	135
		IO digital VSS(0V)			
873		TOP	link_iopad0_data_p_out3_iopad1	pnl_tf12it0nn8	
	OUTPUT	Parallel data output3_1 (LINK)			
874		TOP	vdd_vop.t5	pnl_vop	135
		IO digital VDD(3.3V)			
875		TOP	link_iopad0_data_p_in2_iopad3	pnl_it2nn8	
	INPUT	Parallel data input2_3 (LINK)			
876		TOP	vss_gcs.t28	pnl_gcs	135
		Core digital VSS(0V)			
877		TOP	link_iopad0_data_p_in2_iopad2	pnl_it2nn8	
	INPUT	Parallel data input2_2 (LINK)			
878		TOP	link_iopad0_data_p_in2_iopad1	pnl_it2nn8	
	INPUT	Parallel data input2_1 (LINK)			
879		TOP	link_iopad0_data_p_out2_iopad3	pnl_tf12it0nn8	
	OUTPUT	Parallel data output2_3 (LINK)			
880		TOP	vdd_vc.t16	pnl_vc	135
		Core digital VDD(1.0V)			
881		TOP	link_iopad0_data_p_out2_iopad2	pnl_tf12it0nn8	
	OUTPUT	Parallel data output2_2 (LINK)			
882		TOP	vss_gcs.t27	pnl_gcs	135
		Core digital VSS(0V)			
883		TOP	link_iopad0_data_p_out2_iopad1	pnl_tf12it0nn8	
	OUTPUT	Parallel data output2_1 (LINK)			
884		TOP	vss_go.t4	pnl_go	135
		IO digital VSS(0V)			
885		TOP	link_iopad0_data_p_in1_iopad3	pnl_it2nn8	
	INPUT	Parallel data input1_3 (LINK)			
886		TOP	vdd_vop.t4	pnl_vop	135
		IO digital VDD(3.3V)			
887		TOP	link_iopad0_data_p_in1_iopad2	pnl_it2nn8	
	INPUT	Parallel data input1_2 (LINK)			
888		TOP	link_iopad0_data_p_in1_iopad1	pnl_it2nn8	
	INPUT	Parallel data input1_1 (LINK)			
889		TOP	link_iopad0_data_p_out1_iopad3	pnl_tf12it0nn8	
	OUTPUT	Parallel data output1_3 (LINK)			
890		TOP	vss_gcs.t26	pnl_gcs	135
		Core digital VSS(0V)			
891		TOP	link_iopad0_data_p_out1_iopad2	pnl_tf12it0nn8	
	OUTPUT	Parallel data output1_2 (LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
892		TOP	vdd_vc.t15	pnl_vc	135
			Core digital VDD(1.0V)		
893		TOP	link_iopad0_data_p_out1_iopad1	pnl_tf12it0nn8	
		OUTPUT	Parallel data output1.1 (LINK)		
894		TOP	link_iopad0_event_p_in4_iopad3	pnl_it2nn8	
		INPUT	Parallel event input4.3 (LINK)		
895		TOP	link_iopad0_event_p_in4_iopad2	pnl_it2nn8	
		INPUT	Parallel event input4.2 (LINK)		
896		TOP	vss_gcs.t25	pnl_gcs	135
			Core digital VSS(0V)		
897		TOP	link_iopad0_event_p_in4_iopad1	pnl_it2nn8	
		INPUT	Parallel event input4.1 (LINK)		
898		TOP	link_iopad0_event_p_out4_iopad3	pnl_tf12it0nn8	
		OUTPUT	Parallel event output4.3 (LINK)		
899		TOP	link_iopad0_event_p_out4_iopad2	pnl_tf12it0nn8	
		OUTPUT	Parallel event output4.2 (LINK)		
900		TOP	vdd_vc.t14	pnl_vc	135
			Core digital VDD(1.0V)		
901		TOP	link_iopad0_event_p_out4_iopad1	pnl_tf12it0nn8	
		OUTPUT	Parallel event output4.1 (LINK)		
902		TOP	vss_gcs.t24	pnl_gcs	135
			Core digital VSS(0V)		
903		TOP	link_iopad0_event_p_in3_iopad3	pnl_it2nn8	
		INPUT	Parallel event input3.3 (LINK)		
904		TOP	vss_go.t3	pnl_go	135
			IO digital VSS(0V)		
905		TOP	link_iopad0_event_p_in3_iopad2	pnl_it2nn8	
		INPUT	Parallel event input3.2 (LINK)		
906		TOP	vdd_vop.t3	pnl_vop	135
			IO digital VDD(3.3V)		
907		TOP	link_iopad0_event_p_in3_iopad1	pnl_it2nn8	
		INPUT	Parallel event input3.1 (LINK)		
908		TOP	vss_gcs.t23	pnl_gcs	135
			Core digital VSS(0V)		
909		TOP	link_iopad0_event_p_out3_iopad3	pnl_tf12it0nn8	
		OUTPUT	Parallel event output3.3 (LINK)		
910		TOP	vdd_vc.t13	pnl_vc	135
			Core digital VDD(1.0V)		
911		TOP	link_iopad0_event_p_out3_iopad2	pnl_tf12it0nn8	
		OUTPUT	Parallel event output3.2 (LINK)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
912		TOP	vss_gcs.t22	pnl_gcs	135
		Core digital VSS(0V)			
913		TOP	link_iopad0_event_p_out3_iopad1	pnl_tf12it0nn8	
	OUTPUT	Parallel event output3_1 (LINK)			
914		TOP	link_iopad0_event_p_in2_iopad3	pnl_it2nn8	
	INPUT	Parallel event input2_3 (LINK)			
915		TOP	link_iopad0_event_p_in2_iopad2	pnl_it2nn8	
	INPUT	Parallel event input2_2 (LINK)			
916		TOP	vss_gcs.t21	pnl_gcs	135
		Core digital VSS(0V)			
917		TOP	link_iopad0_event_p_in2_iopad1	pnl_it2nn8	
	INPUT	Parallel event input2_1 (LINK)			
918		TOP	vdd_vc.t12	pnl_vc	135
		Core digital VDD(1.0V)			
919		TOP	link_iopad0_event_p_out2_iopad3	pnl_tf12it0nn8	
	OUTPUT	Parallel event output2_3 (LINK)			
920		TOP	vss_gcs.t20	pnl_gcs	135
		Core digital VSS(0V)			
921		TOP	link_iopad0_event_p_out2_iopad2	pnl_tf12it0nn8	
	OUTPUT	Parallel event output2_2 (LINK)			
922		TOP	vss_gcs.t19	pnl_gcs	135
		Core digital VSS(0V)			
923		TOP	link_iopad0_event_p_out2_iopad1	pnl_tf12it0nn8	
	OUTPUT	Parallel event output2_1 (LINK)			
924		TOP	vdd_vc.t11	pnl_vc	135
		Core digital VDD(1.0V)			
925		TOP	link_iopad0_event_p_in1_iopad3	pnl_it2nn8	
	INPUT	Parallel event input1_3 (LINK)			
926		TOP	vss_gcs.t18	pnl_gcs	135
		Core digital VSS(0V)			
927		TOP	link_iopad0_event_p_in1_iopad2	pnl_it2nn8	
	INPUT	Parallel event input1_2 (LINK)			
928		TOP	vss_gcs.t17	pnl_gcs	135
		Core digital VSS(0V)			
929		TOP	link_iopad0_event_p_in1_iopad1	pnl_it2nn8	
	INPUT	Parallel event input1_1 (LINK)			
930		TOP	vss_go.t2	pnl_go	135
		IO digital VSS(0V)			
931		TOP	link_iopad0_event_p_out1_iopad3	pnl_tf12it0nn8	
	OUTPUT	Parallel event output1_3 (LINK)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
932		TOP	vdd_vc_t10	pnl_vc	135
		Core digital VDD(1.0V)			
933		TOP	vdd_vop_t2	pnl_vop	135
		IO digital VDD(3.3V)			
934		TOP	vss_gcs_t16	pnl_gcs	135
		Core digital VSS(0V)			
935		TOP	link_iopad0_event_p_out1_iopad2	pnl_tf12it0nn8	
	OUTPUT	Parallel event output1_2 (LINK)			
936		TOP	link_iopad0_event_p_out1_iopad1	pnl_tf12it0nn8	
	OUTPUT	Parallel event output1_1 (LINK)			
937		TOP	uart_iopad0_uart1_stx_pad	pnl_tf04it0nn2	
	OUTPUT	TxD channel1 (UART)			
938		TOP	vdd_vc_t9	pnl_vc	135
		Core digital VDD(1.0V)			
939		TOP	uart_iopad0_uart1_srx_pad	pnl_it2nn8	
	INPUT	Rx channel1 (UART)			
940		TOP	vss_gcs_t15	pnl_gcs	135
		Core digital VSS(0V)			
941		TOP	uart_iopad0_uart0_dtr_pad	pnl_tf04it0nn2	
	OUTPUT	DTR channel1 (UART)			
942		TOP	vss_gcs_t14	pnl_gcs	135
		Core digital VSS(0V)			
943		TOP	uart_iopad0_uart0_rts_pad	pnl_tf04it0nn2	
	OUTPUT	RTS channel1 (UART)			
944		TOP	vdd_vc_t8	pnl_vc	135
		Core digital VDD(1.0V)			
945		TOP	uart_iopad0_uart0_stx_pad	pnl_tf04it0nn2	
	OUTPUT	TxD channel0 (UART)			
946		TOP	vss_gcs_t13	pnl_gcs	135
		Core digital VSS(0V)			
947		TOP	uart_iopad0_uart0_dcd_pad	pnl_it2nn8	
	INPUT	DCD channel0 (UART)			
948		TOP	vss_gcs_t12	pnl_gcs	135
		Core digital VSS(0V)			
949		TOP	uart_iopad0_uart0_ri_pad	pnl_it2nn8	
	INPUT	RI channel0 (UART)			
950		TOP	vdd_vc_t7	pnl_vc	135
		Core digital VDD(1.0V)			
951		TOP	uart_iopad0_uart0_dsr_pad	pnl_it2nn8	
	INPUT	DSR channel0 (UART)			

Pin No.	Package	辺	名前	Master Cell	mA
		入出力	備考		
952		TOP	vss_gcs.t11	pnl_gcs	135
			Core digital VSS(0V)		
953		TOP	uart_iopad0_uart0_srx_pad	pnl_lit2nn8	
		INPUT	RxD channel0 (UART)		
954		TOP	uart_iopad0_uart0_cts_pad	pnl_lit2nn8	
		INPUT	CTS channel0 (UART)		
955		TOP	iee1394_iopad0_ieee_d7	pnl_tf04it0nn2	
		INOUT	Data[7] (IEEE1394)		
956		TOP	vdd_vc.t6	pnl_vc	135
			Core digital VDD(1.0V)		
957		TOP	iee1394_iopad0_ieee_d6	pnl_tf04it0nn2	
		INOUT	Data[6] (IEEE1394)		
958		TOP	vss_gcs.t10	pnl_gcs	135
			Core digital VSS(0V)		
959		TOP	iee1394_iopad0_ieee_d5	pnl_tf04it0nn2	
		INOUT	Data[5] (IEEE1394)		
960		TOP	vss_gcs.t9	pnl_gcs	135
			Core digital VSS(0V)		
961		TOP	iee1394_iopad0_ieee_d4	pnl_tf04it0nn2	
		INOUT	Data[4] (IEEE1394)		
962		TOP	vdd_vc.t5	pnl_vc	135
			Core digital VDD(1.0V)		
963		TOP	iee1394_iopad0_ieee_d3	pnl_tf04it0nn2	
		INOUT	Data[3] (IEEE1394)		
964		TOP	vss_gcs.t8	pnl_gcs	135
			Core digital VSS(0V)		
965		TOP	iee1394_iopad0_ieee_d2	pnl_tf04it0nn2	
		INOUT	Data[2] (IEEE1394)		
966		TOP	vss_gcs.t7	pnl_gcs	135
			Core digital VSS(0V)		
967		TOP	iee1394_iopad0_ieee_d1	pnl_tf04it0nn2	
		INOUT	Data[1] (IEEE1394)		
968		TOP	vdd_vc.t4	pnl_vc	135
			Core digital VDD(1.0V)		
969		TOP	iee1394_iopad0_ieee_d0	pnl_tf04it0nn2	
		INOUT	Data[0] (IEEE1394)		
970		TOP	vss_go.t1	pnl_go	135
			IO digital VSS(0V)		
971		TOP	iee1394_iopad0_ieee_isox	pnl_tf04it0nn2	
		OUTPUT	Isolation Control (IEEE1394)		

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
972		TOP	vss_gcs_t6	pnl_gcs	135
		Core digital VSS(0V)			
973		TOP	ieee1394_iopad0_ieee_pc2	pnl_tf04it0nn2	
	OUTPUT	Power Class[2] (IEEE1394)			
974		TOP	vdd_vop_t1	pnl_vop	135
		IO digital VDD(3.3V)			
975		TOP	ieee1394_iopad0_ieee_pc1	pnl_tf04it0nn2	
	OUTPUT	Power Class[1] (IEEE1394)			
976		TOP	vss_gcs_t5	pnl_gcs	135
		Core digital VSS(0V)			
977		TOP	ieee1394_iopad0_ieee_pc0	pnl_tf04it0nn2	
	OUTPUT	Power Class[0] (IEEE1394)			
978		TOP	vdd_vc_t3	pnl_vc	135
		Core digital VDD(1.0V)			
979		TOP	ieee1394_iopad0_ieee_pd	pnl_tf04it0nn2	
	OUTPUT	PD (IEEE1394)			
980		TOP	vss_gcs_t4	pnl_gcs	135
		Core digital VSS(0V)			
981		TOP	ieee1394_iopad0_ieee_lps	pnl_tf04it0nn2	
	OUTPUT	LPS (IEEE1394)			
982		TOP	ieee1394_iopad0_ieee_lreq	pnl_tf04it0nn2	
	OUTPUT	LREQ (IEEE1394)			
983		TOP	ieee1394_iopad0_ieee_cna	pnl_it2nn8	
	INPUT	Cable Not Active (IEEE1394)			
984		TOP	vdd_vc_t2	pnl_vc	135
		Core digital VDD(1.0V)			
985		TOP	ieee1394_iopad0_ieee_lkon	pnl_it2nn8	
	INPUT	Link On (IEEE1394)			
986		TOP	vss_gcs_t3	pnl_gcs	135
		Core digital VSS(0V)			
987		TOP	ieee1394_iopad0_ieee_ctl1	pnl_tf04it0nn2	
	INOUT	CTL[1] (IEEE1394)			
988		TOP	vss_gcs_t2	pnl_gcs	135
		Core digital VSS(0V)			
989		TOP	ieee1394_iopad0_ieee_ctl0	pnl_tf04it0nn2	
	INOUT	CTL[0] (IEEE1394)			
990		TOP	vdd_vc_t1	pnl_vc	135
		Core digital VDD(1.0V)			
991		TOP	ieee1394_iopad0_ieee_sclk	pnl_it2nn8	
	INPUT	Syclk (IEEE1394)			

Pin No.	Package	辺	名前	Master Cell	mA
	入出力	備考			
992		TOP	vss_gcs_t1	pnl_gcs	135
		Core digital VSS(0V)			
993		TOP	ieee1394_iopad0_ieee_rstx	pnl_tf04it0nn2	
	OUTPUT	RST (IEEE1394)			
994		TOP	vdd_vc_t0	pnl_vc	135
		Core digital VDD(1.0V)			
995		TOP	ext_iopad0_data31	pnl_tf12it0nn8	
	INOUT	外部バスデータ [31]			
996		TOP	vss_gcs_t0	pnl_gcs	135
		Core digital VSS(0V)			
997		TOP	ext_iopad0_data30	pnl_tf12it0nn8	
	INOUT	外部バスデータ [30]			
998		TOP	ext_iopad0_data29	pnl_tf12it0nn8	
	INOUT	外部バスデータ [29]			
999		TOP	ext_iopad0_data28	pnl_tf12it0nn8	
	INOUT	外部バスデータ [28]			
1000		TOP	vss_go_t0	pnl_go	135
		IO digital VSS(0V)			
1001		TOP	ext_iopad0_data27	pnl_tf12it0nn8	
	INOUT	外部バスデータ [27]			
1002		TOP	vdd_vop_t0	pnl_vop	135
		IO digital VDD(3.3V)			
1003		TOP	ext_iopad0_data26	pnl_tf12it0nn8	
	INOUT	外部バスデータ [26]			
1004		TOP	ext_iopad0_data25	pnl_tf12it0nn8	
	INOUT	外部バスデータ [25]			



# 3

## 命令セット

### 3.1 MIPS 互換の命令

*Responsive Multithreaded Processor* は MIPS 互換の命令をサポートしている。以下に MIPS 互換の命令を示す。

#### 3.1.1 Load / Store 命令

LB				Load Byte
8bit ロード				MIPS I
31	26 25	21 20	16 15	0
100000	base	rt	offset	

LB

ニーモニック:

LB rt, offset(base)

機能:

$GPR[rt] \leftarrow \text{sign\_extend}(\text{MEM.BYTE}[GPR[\text{base}] + \text{sign\_extend}(\text{offset})])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

指定された番地から 1byte ロードする。ロードされた値は 32bit に符号拡張される。

<b>LBU</b>				Load Byte Unsigned
8bit 符号なしロード				MIPS I
31	26 25	21 20	16 15	0
100100	base	rt	offset	
<b>LBU</b>				

ニーモニック:

LBU rt, offset(base)

機能:

$GPR[rt] \leftarrow \text{zero\_extend}(\text{MEM.BYTE}[GPR[\text{base}] + \text{sign\_extend}(\text{offset})])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

指定された番地から 1byte をロードする。ロードされた値は 32bit にゼロ拡張される。

<b>SB</b>				Store Byte
8bit ストア				MIPS I
31	26 25	21 20	16 15	0
101000	base	rt	offset	
<b>SB</b>				

ニーモニック:

SB rt, offset(base)

機能:

$\text{MEM.BYTE}[GPR[\text{base}] + \text{sign\_extend}(\text{offset})] \leftarrow GPR[rt]$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

指定されたアドレスに 1byte ストアする。

<b>LH</b>		Load Halfword	
16bit ロード		MIPS I	
31	26 25	21 20	16 15 0
100001	base	rt	offset

**LH**

ニーモニック:

LH rt, offset(base)

機能:

$GPR[rt] \leftarrow \text{sign\_extend}(\text{MEM.HWORD}[GPR[\text{base}] + \text{sign\_extend}(\text{offset})])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

概要:

指定されたアドレスから Half word ロードする . ロードされた値は 32bit に符号拡張される .

<b>LHU</b>		Load Halfword Unsigned	
16bit 符号なしロード		MIPS I	
31	26 25	21 20	16 15 0
100101	base	rt	offset
<b>LHU</b>			

ニーモニック:

LHU rt, offset(base)

機能:

$GPR[rt] \leftarrow \text{zero\_extend}(\text{MEM.HWORD}[GPR[\text{base}] + \text{sign\_extend}(\text{offset})])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

概要:

指定されたアドレスから Half word ロードする . ロードされた値は 32bit にゼロ拡張される .

<b>SH</b>		Store Halfword	
16bit ストア		MIPS I	
31	26 25	21 20	16 15 0
101001	base	rt	offset
<b>SH</b>			

ニーモニック:

SH rt, offset(base)

機能:

$\text{MEM.HWORD}[\text{GPR}[\text{base}] + \text{sign\_extend}(\text{offset})] \leftarrow \text{GPR}[\text{rt}]$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Store) :

概要:

指定されたアドレスに Half word ストアする .

<b>LW</b>				Load Word
32bit ロード				MIPS I
31	26 25	21 20	16 15	0
100011	base	rt	offset	
<b>LW</b>				

ニーモニック:

LW rt, offset(base)

機能:

$GPR[rt] \leftarrow MEM.WORD[GPR[base] + sign\_extend(offset)]$

例外:

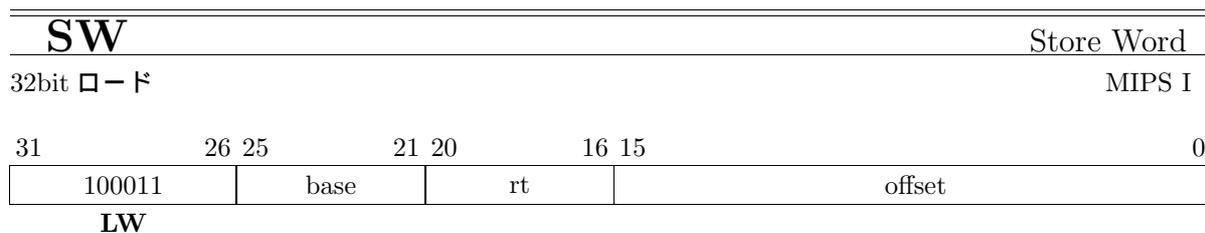
D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

概要:

指定されたアドレスから 1Word ロードする .



ニーモニック:

SW rt, offset(base)

機能:

MEM.WORD[GPR[base] + sign\_extend(offset)] ← GPR[rt]

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Store) :

概要:

指定されたアドレスに 1Word ストアする .

<b>LWL</b>		Load Word Left	
32bit unaligned ロード		MIPS I	
31	26 25	21 20	16 15 0
100010	base	rt	offset

**LWL**

ニーモニック:

LWL rt, offset(base)

機能:

$GPR[rt] \leftarrow \text{merge}(GPR[rt], MEM[GPR[base] + \text{sign\_extend}(\text{offset})])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

アラインされていないアドレスに対してロードを行う。LWR 命令との組み合わせにより、アラインされていないアドレスから 1Word ロードすることができる。

<b>LWR</b>		Load Word Right	
32bit unaligned ロード		MIPS I	
31	26 25	21 20	16 15 0
100110	base	rt	offset
<b>LWR</b>			

ニーモニック:

LWR rt, offset(base)

機能:

$GPR[rt] \leftarrow \text{merge}(\text{MEM}[GPR[\text{base}] + \text{sign\_extend}(\text{offset})], GPR[rt])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

アラインされていないアドレスに対してロードを行う。LWL 命令との組み合わせにより、アラインされていないアドレスから 1Word ロードすることができる。

<b>SWL</b>		Store Word Left	
32bit unaligned ストア		MIPS I	
31	26 25	21 20	16 15 0
101010	base	rt	offset
<b>SWL</b>			

ニーモニック:

SWL rt, offset(base)

機能:

$\text{MEM}[\text{GPR}[\text{base}] + \text{sign\_extend}(\text{offset})] \leftarrow \text{GPR}[\text{rt}]$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

アラインされていないアドレスに対してストアを行う。SWR 命令との組み合わせにより、アラインされていないアドレスに 1Word ストアすることができる。

<b>SWR</b>		Store Word Right	
32bit unaligned ストア		MIPS I	
31	26 25	21 20	16 15 0
101110	base	rt	offset
<b>SWR</b>			

ニーモニック:

SWR rt, offset(base)

機能:

$\text{MEM}[\text{GPR}[\text{base}] + \text{sign\_extend}(\text{offset})] \leftarrow \text{GPR}[\text{rt}]$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

アラインされていないアドレスに対してストアを行う。SWL 命令との組み合わせにより、アラインされていないアドレスに 1Word ストアすることができる。

<b>LL</b>		Load Linked Word	
atomic read-modify-write 用 32bit ロード		MIPS II	
31	26 25	21 20	16 15 0
110000	base	rt	offset

**LL**

ニーモニック:

LL rt, offset(base)

機能:

$GPR[rt] \leftarrow MEM.WORD[GPR[base] + sign\_extend(offset)]$

LL bit  $\leftarrow 1$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

概要:

Atomic Read-Modify-Write のロードを行う。

<b>SC</b>		Store Conditional Word	
atomic read-modify-write 用 32bit ストア		MIPS II	
31	26 25	21 20	16 15 0
111000	base	rt	offset
<b>SC</b>			

ニーモニック:

SC rt, offset(base)

機能:

```

if LL Bit = 1 then
    MEM.WORD[GPR[base] + sign_extend(offset)] ← GPR[rt]
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif

```

例外:

D-TLB No Entry Matched :  
D-TLB Protection Error :  
Data Address Miss Align (Load) :

概要:

Atomic Read-Modify-Write のストアを行う。Atomic Read-Modify-Write が成功すると 1 が返り、失敗すると 0 が返る。

<b>LWC1</b>		Load Word to Floating Point	
浮動小数点レジスタ用 32bit ロード		MIPS I	

31	26 25	21 20	16 15	0
110001	base	rt	offset	

**LWC1**

ニーモニック:

LWC1 ft, offset(base)

機能:

$FPR[ft] \leftarrow MEM.WORD[GPR[base] + sign\_extend(offset)]$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

概要:

指定されたアドレスから浮動小数点レジスタへ 1Word ロードする .

<b>SWC1</b>		Store Word from Floating Point	
浮動小数点レジスタ用 32bit ストア		MIPS I	
31	26 25	21 20	16 15 0
111001	base	rt	offset

**SWC1**

**ニーモニック:**

SWC1 ft, offset(base)

**機能:**

MEM.WORD[GPR[base] + sign\_extend(offset)] ← FPR[ft]

**例外:**

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Store) :

**概要:**

浮動小数点レジスタから指定されたアドレスへ 1Word ストアする .

<b>LDC1</b>		Load Doubleword to Floating Point	
浮動小数点レジスタ用 64bit ロード		MIPS I	

31	26 25	21 20	16 15	0
110101	base	rt	offset	

**LDC1**

ニーモニック:

LDC1 ft, offset(base)

機能:

$FPR[ft] \leftarrow MEM.DWORD[GPR[base] + sign\_extend(offset)]$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

概要:

指定されたアドレスから浮動小数点レジスタへ Double word ロードする .

<b>SDC1</b>		Store Doubleword from Floating Point	
浮動小数点レジスタ用 64bit ストア		MIPS I	
31	26 25	21 20	16 15 0
111101	base	rt	offset
<b>SDC1</b>			

ニーモニック:

SDC1 ft, offset(base)

機能:

$\text{MEM.DWORD}[\text{GPR}[\text{base}] + \text{sign\_extend}(\text{offset})] \leftarrow \text{FPR}[\text{ft}]$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Store) :

概要:

浮動小数点レジスタから指定されたアドレスへ Double word ストアする .

## 3.1.2 演算命令

<b>ADDI</b>				Add Immediate Word
即値加算				MIPS I
31	26 25	21 20	16 15	0
001000	rs	rt	immediate	

**ADDI**

ニーモニック:

ADDI rt, rs, immediate

機能:

$GPR[rt] \leftarrow GPR[rs] + \text{sign\_extend}(\text{immediate})$

例外:

Overflow:

概要:

レジスタと即値を加算する。

<b>ADDIU</b>				Add Immediate Unsigned Word
符号無し即値加算				MIPS I
31	26 25	21 20	16 15	0
001001	rs	rt	immediate	

**ADDIU**

ニーモニック:

ADDIU rt, rs, immediate

機能:

$GPR[rt] \leftarrow GPR[rs] + \text{sign\_extend}(\text{immediate})$

例外:

None

概要:

レジスタと即値を加算する。オーバフローが発生しても例外を起さない。

<b>SLTI</b>		Set on Less Than Immediate	
符号付き即値比較		MIPS I	
31	26 25	21 20	16 15 0
001010	rs	rt	immediate
<b>SLTI</b>			

ニーモニック:

SLTI rt, rs, immediate

機能:

```

if GPR[rs] < sign_extend(immediate) then
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif

```

例外:

None

概要:

レジスタと即値を比較する。

<b>SLTIU</b>		Set on Less Than Immediate Unsigned	
符号無し即値比較		MIPS I	

31	26 25	21 20	16 15	0
001011	rs	rt	immediate	

**SLTIU**

ニーモニック:

SLTIU rt, rs, immediate

機能:

```

if GPR[rs] < sign_extend(immediate) then
    GPR[rt] ← 1
else
    GPR[rt] ← 0
endif

```

例外:

None

概要:

レジスタと即値を符号無し値として比較する。

<b>ANDI</b>		And Immediate	
即値論理積		MIPS I	

31	26 25	21 20	16 15	0
001100	rs	rt	immediate	

**ANDI**

ニーモニック:

ANDI rt, rs, immediate

機能:

GPR[rt] ← GPR[rs] **and** zero\_extend(immediate)

例外:

None

概要:

レジスタと即値の論理積をとる。

<b>ORI</b>				Or Immediate
即値論理和				MIPS I
31	26 25	21 20	16 15	0
001101	rs	rt	immediate	

**ORI**

ニーモニック:

ORI rt, rs, immediate

機能:

GPR[rt] ← GPR[rs] **or** zero\_extend(immediate)

例外:

None

概要:

レジスタと即値の論理和をとる。

<b>XORI</b>				Exclusive Or Immediate
即値排他的論理和				MIPS I
31	26 25	21 20	16 15	0
001110	rs	rt	immediate	

**XORI**

ニーモニック:

XORI rt, rs, immediate

機能:

GPR[rt] ← GPR[rs] **xor** zero\_extend(immediate)

例外:

None

概要:

レジスタと即値の排他的論理和をとる。

<b>LUI</b>				Load Upper Immediate			
上位即値ロード				MIPS I			
31	26	25	21	20	16	15	0
001111	00000	rt	immediate				
<b>LUI</b>		<b>0</b>					

ニーモニック:

LUI rt, immediate

機能:

$GPR[rt] \leftarrow \{ \text{immediate}, 0000000000000000 \}$

例外:

None

概要:

即値をレジスタの上位 16bit にロードする。

<b>ADD</b>						Add Word					
加算						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000	rs	rt	rd	00000	100000						
<b>SPECIAL</b>				<b>0</b>		<b>ADD</b>					

ニーモニック:

ADD rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] + GPR[rt]$

例外:

Overflow:

概要:

レジスタの値を加算する。

<b>ADDU</b>						Add Unsigned Word					
符号無し加算						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00000			100001		
SPECIAL						0			ADDU		

ニーモニック:

ADDU rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] + GPR[rt]$

例外:

None

概要:

レジスタの値を加算する．オーバフローを起しても例外を発生させない．

<b>SUB</b>						Subtract Word					
減算						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00000			100010		
SPECIAL						0			SUB		

ニーモニック:

SUB rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$

例外:

Overflow :

概要:

レジスタの値を減算する．

<b>SUBU</b>						Subtract Unsigned Word					
符号無し減算						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00000			100011		
SPECIAL						0			SUBU		

ニーモニック:

SUBU rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$

例外:

None

概要:

レジスタの値を減算する．オーバフローを起しても例外を発生させない

<b>SLT</b>						Set on Less Than					
符号付き比較						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00000			101010		
SPECIAL						0			SLT		

ニーモニック:

SLT rd, rs, rt

機能:

```
if GPR[rs] < GPR[rt] then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

例外:

None

概要:

レジスタの値を比較する．

<b>SLTU</b>						Set on Less Than Unsigned
符号無し比較						MIPS I
31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	101011	
SPECIAL				0	SLTU	

ニーモニック:

SLTU rd, rs, rt

機能:

```
if GPR[rs] < GPR[rt] then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

例外:

None

概要:

レジスタの値を符号無しで比較する。

<b>AND</b>						And
論理積						MIPS I
31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	100100	
SPECIAL				0	AND	

ニーモニック:

AND rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \text{ and } GPR[rt]$

例外:

None

概要:

レジスタの値の論理積をとる。

<b>OR</b>						Or
論理和						MIPS I
31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	100101	
<b>SPECIAL</b>				<b>0</b>	<b>OR</b>	

ニーモニック:

OR rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \text{ or } GPR[rt]$

例外:

None

概要:

レジスタの値の論理和をとる。

<b>XOR</b>						Exclusive Or
排他的論理和						MIPS I
31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	100110	
<b>SPECIAL</b>				<b>0</b>	<b>XOR</b>	

ニーモニック:

XOR rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \text{ xor } GPR[rt]$

例外:

None

概要:

レジスタの値の排他的論理和をとる。

<b>NOR</b>						Not Or					
否定論理和						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00000		100111	
<b>SPECIAL</b>						<b>0</b>	<b>NOR</b>				

ニーモニック:

NOR rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \text{ nor } GPR[rt]$

例外:

None

概要:

レジスタの値の否定論理和をとる。

<b>SLL</b>						Shift Word Left Logical						
左論理シフト						MIPS I						
31	26	25	21	20	16	15	11	10	6	5	0	
000000		00000		rt		rd		sa		000000		
<b>SPECIAL</b>		<b>0</b>							<b>SLL</b>			

ニーモニック:

SLL rd, rt, sa

機能:

$GPR[rd] \leftarrow GPR[rt] \ll sa$

例外:

None

概要:

レジスタの値を左論理シフトする。

<b>SRL</b>							Shift Word Right Logical				
右論理シフト							MIPS I				
31	26	25	21	20	16	15	11	10	6	5	0
000000			00000		rt	rd	sa		000010		
SPECIAL			0		SRL						

ニーモニック:

SRL rd, rt, sa

機能:

$GPR[rd] \leftarrow GPR[rt] \gg sa$

例外:

None

概要:

レジスタの値を右論理シフトする。

<b>SRA</b>							Shift Word Right Arithmetic				
右算術シフト							MIPS I				
31	26	25	21	20	16	15	11	10	6	5	0
000000			00000		rt	rd	sa		000011		
SPECIAL			0		SRA						

ニーモニック:

SRA rd, rt, sa

機能:

$GPR[rd] \leftarrow GPR[rt] \gg sa$

例外:

None

概要:

レジスタの値を右算術シフトする。

<b>SLLV</b>						Shift Word Left Logical Variable					
左論理シフト						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00000			000100		
SPECIAL						0			SLLV		

ニーモニック:

SLLV rd, rt, rs

機能:

$GPR[rd] \leftarrow GPR[rt] \ll GPR[rs]$

例外:

None

概要:

レジスタの値を左論理シフトする。

<b>SRLV</b>						Shift Word Right Logical Variable					
右論理シフト						MIPS I					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00000			000110		
SPECIAL						0			SRLV		

ニーモニック:

SRLV rd, rt, rs

機能:

$GPR[rd] \leftarrow GPR[rt] \gg GPR[rs]$

例外:

None

概要:

レジスタの値を右論理シフトする。

<b>SRAV</b>		Shift Word Right Arithmetic Variable			
右算術シフト		MIPS I			
31	26 25	21 20	16 15	11 10	6 5 0
000000	rs	rt	rd	00000	000111
<b>SPECIAL</b>				<b>0</b>	<b>SRAV</b>

ニーモニック:

SRAV rd, rt, rs

機能:

$GPR[rd] \leftarrow GPR[rt] \gg GPR[rs]$

例外:

None

概要:

レジスタの値を右算術シフトする。

### 3.1.3 Jump / 分岐命令

<b>J</b>		Jump
ジャンプ		MIPS I
31	26 25	0
000010	instr_index	
<b>J</b>		

ニーモニック:

J target

機能:

$pc \leftarrow \{ pc[31:28], instr\_index, 00 \}$

例外:

None

概要:

指定したアドレスに無条件ジャンプする。

<b>JAL</b>		Jump and Link
プロシージャコール		MIPS I
31	26 25	0
000011	instr_index	
<b>JAL</b>		

ニーモニック:

JAL target

機能:

$pc \leftarrow \{ pc[31:28], instr\_index, 00 \}$

$GPR[31] \leftarrow pc + 8$

例外:

None

概要:

指定したアドレスに無条件ジャンプする。リターンアドレスを R31 に保存する。

<b>JR</b>		Jump Register
レジスタ間接ジャンプ		MIPS I
31	26 25	21 20
000000	rs	0000000000000000
<b>SPECIAL</b>		<b>JR</b>

ニーモニック:

JR rs

機能:

$pc \leftarrow GPR[rs]$

例外:

None

概要:

レジスタで指定したアドレスに無条件ジャンプする。

<b>JALR</b>				Jump and Link Register							
レジスタ間接プロシージャコール				MIPS I							
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs		00000		rd		00000		001001
<b>SPECIAL</b>			<b>0</b>		<b>0</b>		<b>JALR</b>				

ニーモニック:

JALR rs (rd = 31 implied)  
 JALR rd, rs

機能:

$pc \leftarrow GPR[rs]$   
 $GPR[rd] \leftarrow pc + 8$

例外:

None

概要:

指定したアドレスに無条件ジャンプする。リターンアドレスを rd に保存する。

<b>BEQ</b>				Branch on Equal							
条件分岐				MIPS I							
31	26	25	21	20	16	15	0				
000100			rs		rt		offset				
<b>BEQ</b>											

ニーモニック:

BEQ rs, rt, offset

機能:

if  $GPR[rs] = GPR[rt]$  then  
 branch

例外:

None

概要:

rs と rt が等しい場合、指定したアドレスに分岐する。

<b>BNE</b>				Branch on Not Equal
条件分岐				MIPS I
31	26 25	21 20	16 15	0
000101	rs	rt	offset	
<b>BNE</b>				

ニーモニック:

BNE rs, rt, offset

機能:

if GPR[rs]  $\neq$  GPR[rt] then  
branch

例外:

None

概要:

rs と rt が等しくない場合, 指定したアドレスに分岐する.

<b>BLEZ</b>				Branch on Less Than or Equal to Zero
条件分岐				MIPS I
31	26 25	21 20	16 15	0
000110	rs	00000	offset	
<b>BLEZ</b>				

ニーモニック:

BLEZ rs, offset

機能:

if GPR[rs]  $\leq$  0 then  
branch

例外:

None

概要:

rs が 0 以下の場合, 指定したアドレスに分岐する.

**BGTZ**

Branch on Greater Than Zero

条件分岐

MIPS I

31	26 25	21 20	16 15	0
000111	rs	00000	offset	
<b>BGTZ</b>		<b>0</b>		

ニーモニック:

BGTZ rs, offset

機能:

if GPR[rs] > 0 then  
branch

例外:

None

概要:

rs が 0 より大きい場合, 指定したアドレスに分岐する.

**BEQL**

Branch on Equal Likely

条件分岐

MIPS II

31	26 25	21 20	16 15	0
010100	rs	rt	offset	
<b>BEQL</b>				

ニーモニック:

BEQL rs, rt, offset

機能:

if GPR[rs] = GPR[rt] then  
branch\_likely

例外:

None

概要:

rs と rt が等しい場合, 指定したアドレスに分岐する.

<b>BNEL</b>		Branch on Not Equal Likely	
条件分岐		MIPS II	
31	26 25	21 20	16 15 0
010101	rs	rt	offset
<b>BNEL</b>			

ニーモニック:

BNEL rs, rt, offset

機能:

if GPR[rs]  $\neq$  GPR[rt] then  
branch\_likely

例外:

None

概要:

rs と rt が等しくない場合, 指定したアドレスに分岐する.

<b>BLEZL</b>		Branch on Less Than or Equal to Zero Likely	
条件分岐		MIPS II	
31	26 25	21 20	16 15 0
010110	rs	00000	offset
<b>BLEZL</b>		<b>0</b>	

ニーモニック:

BLEZL rs, offset

機能:

if GPR[rs]  $\leq$  0 then  
branch\_likely

例外:

None

概要:

rs が 0 以下の場合, 指定したアドレスに分岐する.

<b>BGTZL</b>		Branch on Greater Than to Zero Likely	
条件分岐		MIPS II	
31	26 25	21 20	16 15 0
010111	rs	00000	offset
<b>BGTZL</b>		<b>0</b>	

ニーモニック:

BGTZL rs, offset

機能:

if GPR[rs] > 0 then  
branch\_likely

例外:

None

概要:

rs0 が 0 より大きい場合, 指定したアドレスに分岐する.

<b>BLTZ</b>		Branch on Less Than Zero	
条件分岐		MIPS I	
31	26 25	21 20	16 15 0
000001	rs	00000	offset
<b>REGIMM</b>		<b>BLTZ</b>	

ニーモニック:

BLTZ rs, offset

機能:

if GPR[rs] < 0 then  
branch

例外:

None

概要:

rs が 0 より小さい場合, 指定したアドレスに分岐する.

<b>BGEZ</b>		Branch on Greater Than or Equal to Zero	
条件分岐		MIPS I	
31	26 25	21 20	16 15 0
000001	rs	00001	offset
<b>REGIMM</b>		<b>BGEZ</b>	

ニーモニック:

BGEZ rs, offset

機能:

if GPR[rs]  $\geq$  0 then  
branch

例外:

None

概要:

rs が 0 以上の場合, 指定したアドレスに分岐する.

<b>BLTZAL</b>		Branch on Less Than Zero and Link	
条件プロシージャコール		MIPS I	
31	26 25	21 20	16 15 0
000001	rs	10000	offset
<b>REGIMM</b>		<b>BLTZAL</b>	

ニーモニック:

BGEZ rs, offset

機能:

if GPR[rs]  $<$  0 then  
branch  
GPR[31]  $\leftarrow$  pc + 8

例外:

None

概要:

rs が 0 より小さい場合, 指定したアドレスに分岐する. リターンアドレスを R31 にセーブする.

<b>BGEZAL</b>		Branch on Greater Than or Equal to Zero and Link	
条件プロシージャコール		MIPS I	
31	26 25	21 20	16 15 0
000001	rs	10001	offset
REGIMM		BGEZAL	

ニーモニック:

BGEZAL rs, offset

機能:

```
if GPR[rs] ≥ 0 then
    branch
    GPR[31] ← pc + 8
```

例外:

None

概要:

rs が 0 以上の場合、指定したアドレスに分岐する。リターンアドレスを R31 にセーブする。

<b>BLTZL</b>		Branch on Less Than Zero Likely	
条件分岐		MIPS II	
31	26 25	21 20	16 15 0
000001	rs	00010	offset
REGIMM		BLTZL	

ニーモニック:

BLTZL rs, offset

機能:

```
if GPR[rs] < 0 then
    branch_likely
```

例外:

None

概要:

rs が 0 より小さい場合、指定したアドレスに分岐する。

<b>BGEZL</b>		Branch on Greater Than or Equal to Zero Likely	
条件分岐		MIPS II	
31	26 25	21 20	16 15
000001	rs	00011	offset
<b>REGIMM</b>		<b>BGEZL</b>	

ニーモニック:

BGEZL rs, offset

機能:

if GPR[rs]  $\geq$  0 then  
branch\_likely

例外:

None

概要:

rs が 0 以上の場合, 指定したアドレスに分岐する.

<b>BLTZALL</b>		Branch on Less Than Zero and Link Likely	
条件プロシージャコール		MIPS II	
31	26 25	21 20	16 15
000001	rs	10010	offset
<b>REGIMM</b>		<b>BLTZALL</b>	

ニーモニック:

BLTZALL rs, offset

機能:

if GPR[rs]  $<$  0 then  
branch\_likely  
GPR[31]  $\leftarrow$  pc + 8

例外:

None

概要:

rs が 0 より小さい場合, 指定したアドレスに分岐する. R31 にリターンアドレスをセーブする.

<b>BGEZALL</b>		Branch on Greater Than or Equal to Zero and Link Likely		
条件プロシージャコール		MIPS II		
31	26 25	21 20	16 15	0
000001	rs	10011	offset	
REGIMM		BGEZALL		

ニーモニック:

BGEZALL rs, offset

機能:

if GPR[rs]  $\geq$  0 then  
 branch\_likely  
 GPR[31]  $\leftarrow$  pc + 8

例外:

None

概要:

rs が 0 以上の場合、指定したアドレスに分岐する。リターンアドレスを R31 にセーブする。

### 3.1.4 浮動小数点命令

<b>MTC1</b>		Move Word to Floating Point			
レジスタ間転送		MIPS I			
31	26 25	21 20	16 15	11 10	0
010001	00100	rt	fs	0000000000	
COP1		MT		0	

ニーモニック:

MTC1 rt, fs

機能:

FPR[fs]  $\leftarrow$  GPR[rt]

例外:

None

概要:

汎用レジスタの値を浮動小数点レジスタに入れる。

<b>MFC1</b>		Move Word from Floating Point	
レジスタ間転送		MIPS I	
31	26 25	21 20	16 15
11 10			0
010001	00000	rt	fs
<b>COP1</b>	<b>MF</b>	<b>0</b>	

ニーモニック:

MFC1 rt, fs

機能:

GPR[rt] ← FPR[fs]

例外:

None

概要:

浮動小数点レジスタの値を汎用レジスタに入れる。

<b>ADD.fmt</b>		Floating Point Add	
浮動小数点加算		MIPS I	
31	26 25	21 20	16 15
11 10	6 5	0	
010001	fmt	ft	fs
fd	000000		
<b>COP1</b>	<b>ADD</b>		

ニーモニック:

ADD.S fd, fs, ft (fmt = 10000)

ADD.D fd, fs, ft (fmt = 10001)

機能:

$FPR[fd] \leftarrow FPR[fs] + FPR[ft]$

例外:

Floating Point Invalid Operation :

Floating Point Inexact :

Floating Point Overflow :

Floating Point Underflow :

概要:

レジスタの値を加算する。

<b>SUB.fmt</b>		Floating Point Subtract	
浮動小数点減算		MIPS I	
31	26 25	21 20	16 15
11 10	6 5	0	
010001	fmt	ft	fs
fd	000001		
<b>COP1</b>		<b>SUB</b>	

## ニーモニック:

SUB.S fd, fs, ft (fmt = 10000)

SUB.D fd, fs, ft (fmt = 10001)

## 機能:

$FPR[fd] \leftarrow FPR[fs] - FPR[ft]$

## 例外:

Floating Point Invalid Operation :

Floating Point Inexact :

Floating Point Overflow :

Floating Point Underflow :

## 概要:

レジスタの値を減算する。

<b>MUL.fmt</b>		Floating Point Multiply				
浮動小数点乗算		MIPS I				
31	26 25	21 20	16 15	11 10	6 5	0
010001	fmt	ft	fs	fd	000010	
<b>COP1</b>					<b>MUL</b>	

## ニーモニック:

MUL.S fd, fs, ft (fmt = 10000)

MUL.D fd, fs, ft (fmt = 10001)

## 機能:

$FPR[fd] \leftarrow FPR[fs] \times FPR[ft]$

## 例外:

Floating Point Invalid Operation :

Floating Point Inexact :

Floating Point Overflow :

Floating Point Underflow :

## 概要:

レジスタの値を乗算する。

<b>DIV.fmt</b>		Floating Point Divide	
浮動小数点除算		MIPS I	
31	26 25	21 20	16 15
11 10	6 5	0	
010001	fmt	ft	fs
fd	000011		
<b>COP1</b>		<b>DIV</b>	

## ニーモニック:

DIV.S fd, fs, ft (fmt = 10000)  
 DIV.D fd, fs, ft (fmt = 10001)

## 機能:

$FPR[fd] \leftarrow FPR[fs] / FPR[ft]$

## 例外:

Floating Point Invalid Operation :  
 Floating Point Inexact :  
 Floating Point Overflow :  
 Floating Point Underflow :  
 Floating Point Divide By 0 :

## 概要:

レジスタの値を除算する。

<b>ABS.fmt</b>		Floating Point Absolute Value				
浮動小数点絶対値		MIPS I				
31	26 25	21 20	16 15	11 10	6 5	0
010001	fmt	00000	fs	fd	000101	
<b>COP1</b>		<b>0</b>			<b>ABS</b>	

ニーモニック:

ABS.S fd, fs (fmt = 10000)  
 ABS.D fd, fs (fmt = 10001)

機能:

$FPR[fd] \leftarrow \text{abs}(FPR[fs])$

例外:

Floating Point Invalid Operation:

概要:

レジスタの値の絶対値をとる。

<b>NEG.fmt</b>						Floating Point Negate
浮動小数点符号反転						MIPS I
31	26 25	21 20	16 15	11 10	6 5	0
010001	fmt	00000	fs	fd	000111	
<b>COP1</b>		<b>0</b>			<b>NEG</b>	

ニーモニック:

NEG.S fd, fs (fmt = 10000)

NEG.D fd, fs (fmt = 10001)

機能:

$FPR[fd] \leftarrow -(FPR[fs])$

例外:

Floating Point Invalid Operation :

概要:

レジスタの値を符号反転する .

<b>MOV.fmt</b>						Floating Point Move
浮動小数点移動						MIPS I
31	26 25	21 20	16 15	11 10	6 5	0
010001	fmt	00000	fs	fd	000110	
<b>COP1</b>		<b>0</b>			<b>MOV</b>	

ニーモニック:

MOV.S fd, fs (fmt = 10000)

MOV.D fd, fs (fmt = 10001)

機能:

$FPR[fd] \leftarrow FPR[fs]$

例外:

None

概要:

レジスタの値をコピーする .

<b>CVT.S.fmt</b>		Floating Point Convert to Single Floating Point			
浮動小数点フォーマット変換				MIPS I	
31	26 25	21 20	16 15	11 10	6 5 0
010001	fmt	00000	fs	fd	100000
<b>COP1</b>		<b>0</b>			<b>CVT.S</b>

ニーモニック:

CVT.S.D fd, fs (fmt = 10001)  
 CVT.S.W fd, fs (fmt = 10100)

機能:

$FPR[fd] \leftarrow \text{convert\_and\_round}( FPR[fs] )$

例外:

Floating Point Invalid Operation :  
 Floating Point Inexact :  
 Floating Point Overflow :  
 Floating Point Underflow :

概要:

レジスタの値を単精度浮動小数点フォーマットに変換する。

<b>CVT.D.fmt</b>		Floating Point Convert to Double Floating Point			
浮動小数点フォーマット変換				MIPS I	
31	26 25	21 20	16 15	11 10	6 5 0
010001	fmt	00000	fs	fd	100001
<b>COP1</b>		<b>0</b>			<b>CVT.D</b>

ニーモニック:

CVT.D.S fd, fs (fmt = 10000)

CVT.D.W fd, fs (fmt = 10100)

機能:

$FPR[fd] \leftarrow \text{convert\_and\_round}( FPR[fs] )$

例外:

Floating Point Invalid Operation :

Floating Point Inexact :

概要:

レジスタの値を倍精度浮動小数点フォーマットに変換する。

<b>ROUND.W.fmt</b>		Floating Point Round to Word Fixed Point			
浮動小数点フォーマット変換		MIPS II			
31	26 25	21 20	16 15	11 10	6 5 0
010001	fmt	00000	fs	fd	001100
<b>COP1</b>		<b>0</b>			<b>ROUND.W</b>

ニーモニック:

ROUND.W.S fd, fs (fmt = 10000)  
 ROUND.W.D fd, fs (fmt = 10001)

機能:

$FPR[fd] \leftarrow \text{convert\_and\_round}( FPR[fs] )$

例外:

Floating Point Invalid Operation :  
 Floating Point Inexact :  
 Floating Point Overflow :

概要:

レジスタノ値を整数フォーマットに変換する。

<b>TRUNC.W.fmt</b>		Floating Point Truncate to Word Fixed Point				
浮動小数点フォーマット変換		MIPS II				
31	26 25	21 20	16 15	11 10	6 5	0
010001	fmt	00000	fs	fd	001101	
<b>COP1</b>		<b>0</b>				<b>TRUNC.W</b>

ニーモニック:

TRUNC.W.S fd, fs (fmt = 10000)  
 TRUNC.W.D fd, fs (fmt = 10001)

機能:

$FPR[fd] \leftarrow \text{convert\_and\_round}( FPR[fs] )$

例外:

Floating Point Invalid Operation :  
 Floating Point Inexact :  
 Floating Point Overflow :

概要:

レジスタの値を整数フォーマットに変換する。

<b>CEIL.W.fmt</b>		Floating Point Ceiling to Word Fixed Point				
浮動小数点フォーマット変換		MIPS II				
31	26 25	21 20	16 15	11 10	6 5	0
010001	fmt	00000	fs	fd	001110	
<b>COP1</b>		<b>0</b>			<b>CEIL.W</b>	

ニーモニック:

CEIL.W.S fd, fs (fmt = 10000)  
 CEIL.W.D fd, fs (fmt = 10001)

機能:

$FPR[fd] \leftarrow \text{convert\_and\_round}(FPR[fs])$

例外:

Floating Point Invalid Operation :  
 Floating Point Inexact :  
 Floating Point Overflow :

概要:

レジスタの値を整数フォーマットに変換する。

<b>FLOOR.W.fmt</b>		Floating Point Floor Convert to Word Fixed Point			
浮動小数点フォーマット変換		MIPS II			
31	26 25	21 20	16 15	11 10	6 5 0
010001	fmt	00000	fs	fd	001111
<b>COP1</b>		<b>0</b>			<b>FLOOR.W</b>

ニーモニック:

FLOOR.W.S fd, fs (fmt = 10000)  
 FLOOR.W.D fd, fs (fmt = 10001)

機能:

$FPR[fd] \leftarrow \text{convert\_and\_round}( FPR[fs] )$

例外:

Floating Point Invalid Operation :  
 Floating Point Inexact :  
 Floating Point Overflow :

概要:

レジスタの値を整数フォーマットに変換する。

## 3.1.5 その他の命令

<b>SYSCALL</b>		System Call
システムコール		MIPS I
31	26 25	6 5 0
000000	000000000000000000000000	001100
<b>SPECIAL</b>	<b>0</b>	<b>SYSCALL</b>

ニーモニック:

SYSCALL

機能:

exception(system\_call)

例外:

System Call :

概要:

システムコール例外を発生する .

<b>BREAK</b>		Breakpoint
ブレイクポイント		MIPS I
31	26 25	6 5 0
000000	000000000000000000000000	001101
<b>SPECIAL</b>	<b>0</b>	<b>BREAK</b>

ニーモニック:

BREAK

機能:

exception(breakpoint)

例外:

Break Point :

概要:

ブレイクポイント例外を発生する .

<b>TGE</b>					Trap if Greater or Equal				
条件トラップ					MIPS II				
31	26	25	21	20	16	15	6	5	0
000000	rs		rt		0000000000			110000	
<b>SPECIAL</b>					<b>0</b>			<b>TGE</b>	

ニーモニック:

TGE rs, rt

機能:

if GPR[rs]  $\geq$  GPR[rt] then  
exception(trap)

例外:

Trap:

概要:

rs が rt 以上の場合, トラップが発生する.

<b>TGEU</b>					Trap if Greater or Equal Unsigned				
条件トラップ					MIPS II				
31	26	25	21	20	16	15	6	5	0
000000	rs		rt		0000000000			110001	
<b>SPECIAL</b>					<b>0</b>			<b>TGEU</b>	

ニーモニック:

TGEU rs, rt

機能:

if GPR[rs]  $\geq$  GPR[rt] then  
exception(trap)

例外:

Trap:

概要:

rs が rt 以上の場合, トラップが発生する. 値は符号無しとして比較する.

<b>TLT</b>				Trap if Less Than	
条件トラップ				MIPS II	
31	26 25	21 20	16 15	6 5	0
000000	rs	rt	0000000000	110010	
<b>SPECIAL</b>			<b>0</b>	<b>TLT</b>	

ニーモニック:

TLT rs, rt

機能:

if GPR[rs] < GPR[rt] then  
exception(trap)

例外:

Trap:

概要:

rs が rt より小さい場合、トラップが発生する。

<b>TLTU</b>				Trap if Less Than Unsigned	
条件トラップ				MIPS II	
31	26 25	21 20	16 15	6 5	0
000000	rs	rt	0000000000	110011	
<b>SPECIAL</b>			<b>0</b>	<b>TLTU</b>	

ニーモニック:

TLTU rs, rt

機能:

if GPR[rs] < GPR[rt] then  
exception(trap)

例外:

Trap:

概要:

rs が rt より小さい場合、トラップが発生する。値は符号無しとして比較する。

<b>TEQ</b>				Trap if Equal	
条件トラップ				MIPS II	
31	26 25	21 20	16 15	6 5	0
000000	rs	rt	0000000000	110100	
<b>SPECIAL</b>			<b>0</b>	<b>TEQ</b>	

ニーモニック:

TEQ rs, rt

機能:

if GPR[rs] = GPR[rt] then  
exception(trap)

例外:

Trap:

概要:

rs が rt と等しい場合、トラップが発生する。

<b>TNE</b>				Trap if Not Equal	
条件トラップ				MIPS II	
31	26 25	21 20	16 15	6 5	0
000000	rs	rt	0000000000	110110	
<b>SPECIAL</b>			<b>0</b>	<b>TNE</b>	

ニーモニック:

TEQ rs, rt

機能:

if GPR[rs]  $\neq$  GPR[rt] then  
exception(trap)

例外:

Trap:

概要:

rs が rt と等しくない場合、トラップが発生する。

<b>TGEI</b>				Trap if Greater or Equal Immediate
条件トラップ				MIPS II
31	26 25	21 20	16 15	0
000001	rs	01000	immediate	
REGIMM		TGEI		

ニーモニック:

TGEI rs, immediate

機能:

if GPR[rs]  $\geq$  sign\_extend( immediate ) then  
exception(trap)

例外:

Trap:

概要:

rs が immediate 以上の場合，トラップが発生する．

<b>TGEIU</b>				Trap if Greater or Equal Immediate Unsigned
条件トラップ				MIPS II
31	26 25	21 20	16 15	0
000001	rs	01001	immediate	
REGIMM		TGEIU		

ニーモニック:

TGEIU rs, immediate

機能:

if GPR[rs]  $\geq$  sign\_extend( immediate ) then  
exception(trap)

例外:

Trap:

概要:

rs が immediate 以上の場合，トラップが発生する．値は符号無しとして比較する．

<b>TLTI</b>				Trap if Less Than Immediate
条件トラップ				MIPS II
31	26 25	21 20	16 15	0
000001	rs	01010	immediate	
<b>REGIMM</b>		<b>TLTI</b>		

ニーモニック:

TLTI rs, immediate

機能:

```
if GPR[rs] < sign_extend( immediate ) then
    exception(trap)
```

例外:

Trap:

概要:

rs が immediate より小さい場合、トラップが発生する。

<b>TLTIU</b>				Trap if Less Than Immediate Unsigned
条件トラップ				MIPS II
31	26 25	21 20	16 15	0
000001	rs	01010	immediate	
<b>REGIMM</b>		<b>TLTIU</b>		

ニーモニック:

TLTIU rs, immediate

機能:

```
if GPR[rs] < sign_extend( immediate ) then
    exception(trap)
```

例外:

Trap:

概要:

rs が immediate より小さい場合、トラップが発生する。値は符号無しとして比較する。

<b>TEQI</b>				Trap if Equal Immediate
条件トラップ				MIPS II
31	26 25	21 20	16 15	0
000001	rs	01100	immediate	
<b>REGIMM</b>		<b>TEQI</b>		

ニーモニック:

TEQI rs, immediate

機能:

if GPR[rs] = sign\_extend( immediate ) then  
exception(trap)

例外:

Trap:

概要:

rs と immediate が等しい場合, トラップが発生する.

<b>TNEI</b>				Trap if Not Equal Immediate
条件トラップ				MIPS II
31	26 25	21 20	16 15	0
000001	rs	01110	immediate	
<b>REGIMM</b>		<b>TNEI</b>		

ニーモニック:

TNEI rs, immediate

機能:

if GPR[rs]  $\neq$  sign\_extend( immediate ) then  
exception(trap)

例外:

Trap:

概要:

rs と immediate が等しくない場合, トラップが発生する.

## 3.2 MIPS 命令と動作の異なる命令

以下に *Responsive Multithreaded Processor* の中で MIPS 命令と動作の異なる命令を示す。

### 3.2.1 演算命令

<b>DADDI</b>		Doubleword Add Immediate	
64bit 即値加算		MIPS III 動作改	
31	26 25	21 20	16 15 0
011000	rs	rt	immediate

**DADDI**

ニーモニック:

DADDI rt, rs, immediate

機能:

$FPR[rt] \leftarrow FPR[rs] + \text{sign\_extension}(\text{immediate})$

例外:

Overflow:

概要:

加算を行う。*Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DADDIU</b>				Doubleword Add Immediate Unsigned			
64bit 即値加算				MIPS III 動作改			
31	26 25	21 20	16 15	0			
011001	rs	rt	immediate				

**DADDIU**

ニーモニック:

DADDIU rt, rs, immediate

機能:

$FPR[rt] \leftarrow FPR[rs] + \text{sign\_extension}(\text{immediate})$

例外:

None

概要:

加算を行う。オーバーフローが発生しても例外を発生させない。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DADD</b>				Doubleword Add			
64bit 加算				MIPS III 動作改			
31	26 25	21 20	16 15	11 10	6 5	0	
000000	rs	rt	rd	00000	101100		

**SPECIAL**

**0**

**DADD**

ニーモニック:

DADD rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] + FPR[rt]$

例外:

Overflow :

概要:

加算を行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DADDU</b>						Doubleword Add Unsigned					
64bit 加算						MIPS III 動作改					
31	26	25	21	20	16	15	11	10	6	5	0
000000	rs		rt		rd		00000		101101		
<b>SPECIAL</b>						<b>0</b>		<b>DADDU</b>			

ニーモニック:

DADDU rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] + FPR[rt]$

例外:

None

概要:

加算を行う。オーバーフローが発生しても例外を発生させない。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSUB</b>						Doubleword Subtract					
64bit 減算						MIPS III 動作改					
31	26	25	21	20	16	15	11	10	6	5	0
000000	rs		rt		rd		00000		101110		
<b>SPECIAL</b>						<b>0</b>		<b>DSUB</b>			

ニーモニック:

DSUB rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] - FPR[rt]$

例外:

Overflow :

概要:

減算を行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSUBU</b>						Doubleword Subtract Unsigned	
64bit 減算						MIPS III 動作改	
31	26 25	21 20	16 15	11 10	6 5	0	
000000	rs	rt	rd	00000	101111		
<b>SPECIAL</b>				<b>0</b>	<b>DSUBU</b>		

ニーモニック:

DSUBU rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] - FPR[rt]$

例外:

None

概要:

減算を行う。オーバーフローが発生しても例外を発生させない。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSL</b>						Doubleword Shift Left Logical	
64bit 左論理シフト						MIPS III 動作改	
31	26 25	21 20	16 15	11 10	6 5	0	
000000	00000	rt	rd	sa	111000		
<b>SPECIAL</b>		<b>0</b>				<b>DSL</b>	

ニーモニック:

DSL rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \ll sa$

例外:

None

概要:

左論理シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSRL</b>		Doubleword Shift Right Logical				
64bit 右論理シフト		MIPS III 動作改				
31	26 25	21 20	16 15	11 10	6 5	0
000000	00000	rt	rd	sa	111010	
<b>SPECIAL</b>		<b>0</b>		<b>DSRL</b>		

ニーモニック:

DSRL rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \gg sa$

例外:

None

概要:

右論理シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSRA</b>		Doubleword Shift Right Arithmetic				
64bit 右算術シフト		MIPS III 動作改				
31	26 25	21 20	16 15	11 10	6 5	0
000000	00000	rt	rd	sa	111011	
<b>SPECIAL</b>		<b>0</b>		<b>DSRA</b>		

ニーモニック:

DSRA rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \gg sa$

例外:

None

概要:

右算術シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSLL32</b>		Doubleword Shift Left Logical plus 32	
64bit 左論理シフト		MIPS III 動作改	
31	26 25	21 20	16 15
11 10	6 5	0	
000000	00000	rt	rd
sa	111100		
<b>SPECIAL</b>	<b>0</b>	<b>DSLL32</b>	

ニーモニック:

DSLL32 rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \ll (sa + 32)$

例外:

None

概要:

左論理シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSRL32</b>		Doubleword Shift Right Logical plus 32	
64bit 右論理シフト		MIPS III 動作改	
31	26 25	21 20	16 15
11 10	6 5	0	
000000	00000	rt	rd
sa	111110		
<b>SPECIAL</b>	<b>0</b>	<b>DSRL32</b>	

ニーモニック:

DSRL32 rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \gg (sa + 32)$

例外:

None

概要:

右論理シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSRA32</b>		Doubleword Shift Right Arithmetic plus 32				
64bit 右算術シフト		MIPS III 動作改				
31	26 25	21 20	16 15	11 10	6 5	0
000000	00000	rt	rd	sa	111111	
SPECIAL		0		DSRA32		

ニーモニック:

DSRA32 rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \gg (sa + 32)$

例外:

None

概要:

右算術シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSLIV</b>		Doubleword Shift Left Logical Variable				
64bit 左論理シフト		MIPS III 動作改				
31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	010100	
SPECIAL		0		DSLIV		

ニーモニック:

DSLIV rd, rt, rs

機能:

$FPR[rd] \leftarrow FPR[rt] \ll FPR[rs]$

例外:

None

概要:

左論理シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSRLV</b>						Doubleword Shift Right Logical Variable					
64bit 右論理シフト						MIPS III 動作改					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00000		010110	
<b>SPECIAL</b>						<b>0 DSRLV</b>					

ニーモニック:

DSRLV rd, rt, rs

機能:

$FPR[rd] \leftarrow FPR[rt] \gg FPR[rs]$

例外:

None

概要:

右論理シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>DSRAV</b>						Doubleword Shift Right Arithmetic Variable					
64bit 右算術シフト						MIPS III 動作改					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00000		010111	
<b>SPECIAL</b>						<b>0 DSRAV</b>					

ニーモニック:

DSRAV rd, rt, rs

機能:

$FPR[rd] \leftarrow FPR[rt] \ggg FPR[rs]$

例外:

None

概要:

右算術シフトを行う。 *Responsive Multithreaded Processor* では浮動小数点レジスタを用いて演算を行う。

<b>MULT</b>						Multiply Word
符号付き乗算						MIPS I 動作改
31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	011000	
SPECIAL				0	MULT	

ニーモニック:

MULT rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \times GPR[rt]$

例外:

None

概要:

乗算を行う。Responsive Multithreaded Processor では 3 オペランド命令で演算結果の下位 32bit をデスティネーションレジスタに格納する。

<b>MULTU</b>						Multiply Word Unsigned
符号無し乗算						MIPS I 動作改
31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	011001	
SPECIAL				0	MULTU	

ニーモニック:

MULTU rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \times GPR[rt]$

例外:

None

概要:

乗算を行う。値は符号無し整数として扱う。Responsive Multithreaded Processor では 3 オペランド命令で演算結果の下位 32bit をデスティネーションレジスタに格納する。

<b>DIV</b>						Divide Word					
符号付き除算						MIPS I 動作改					
31	26	25	21	20	16	15	11	10	6	5	0
000000	rs	rt	rd	00000	011010						
<b>SPECIAL</b>						<b>0</b>	<b>DIV</b>				

ニーモニック:

DIV rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \div GPR[rt]$

例外:

Divide by Zero :

概要:

除算を行う。 *Responsive Multithreaded Processor* では3オペランド命令で、商をデスティネーションレジスタに格納する。

<b>DIVU</b>						Divide Word Unsigned					
符号無し除算						MIPS I 動作改					
31	26	25	21	20	16	15	11	10	6	5	0
000000	rs	rt	rd	00000	011011						
<b>SPECIAL</b>						<b>0</b>	<b>DIVU</b>				

ニーモニック:

DIVU rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \div GPR[rt]$

例外:

Divide by Zero :

概要:

除算を行う。値は符号無し整数として扱う。商をデスティネーションレジスタに格納する。 *Responsive Multithreaded Processor* では3オペランド命令になる。

<b>DMULT</b>						Doubleword Multiply	
符号付き 64bit 乗算						MIPS III 動作改	
31	26 25	21 20	16 15	11 10	6 5	0	
000000	rs	rt	rd	00000	011100		
<b>SPECIAL</b>				<b>0</b>	<b>DMULT</b>		

ニーモニック:

DMULT rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \times FPR[rt]$

例外:

None

概要:

乗算を行う。Responsive Multithreaded Processor では 3 オペランド命令で、浮動小数点レジスタを用いて演算を行う。演算結果の下位 64bit をデスティネーションレジスタに格納する。

<b>DMULTU</b>						Doubleword Multiply Unsigned	
符号無し 64bit 乗算						MIPS III 動作改	
31	26 25	21 20	16 15	11 10	6 5	0	
000000	rs	rt	rd	00000	011101		
<b>SPECIAL</b>				<b>0</b>	<b>DMULTU</b>		

ニーモニック:

DMULTU rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \times FPR[rt]$

例外:

None

概要:

乗算を行う。値は符号無し整数として扱う。Responsive Multithreaded Processor では 3 オペランド命令で、浮動小数点レジスタを用いて演算を行う。演算結果の下位 64bit をデスティネーションレジスタに格納する。

## 3.2.2 浮動小数点命令

<b>C.cond.fmt</b>							Floating-Point Compare						
浮動小数点比較							MIPS I 動作改						
31	26	25	21	20	16	15	11	10	6	5	4	3	0
010001	fmt		ft		fs		00000		11		cond		
<b>COP1</b>							<b>0</b>		<b>FC</b>				

ニーモニック:

C.cond.S fs, ft (fmt = 10000)

C.cond.D fs, ft (fmt = 10001)

機能:

$FPR[7] \leftarrow FPR[fs] \text{ compare\_cond } FPR[ft]$

例外:

Floating Point Invalid:

概要:

浮動小数点の比較を行う。 *Responsive Multithreaded Processor* ではステータスレジスタではなく、浮動小数点レジスタに結果が格納される。

<b>BC1T</b>										Branch on FP True
浮動小数点分岐										MIPS I 動作改
31	26	25	21	20	18	17	16	15	0	
010001		01000		000		0	1	offset		
<b>COP1</b>		<b>BC</b>		<b>9</b>		<b>nd tf</b>				

ニーモニック:

BC1T offset

機能:

if FPR[7] = 1 then  
branch

例外:

None

概要:

FPR[7] が 1 の場合, 指定されたアドレスに分岐する. *Responsive Multithreaded Processor* ではステータスレジスタではなく, 浮動小数点レジスタの内容により分岐を判断する.

<b>BC1F</b>										Branch on FP False
浮動小数点分岐										MIPS I 動作改
31	26	25	21	20	18	17	16	15	0	
010001		01000		000		0	0	offset		
<b>COP1</b>		<b>BC</b>		<b>9</b>		<b>nd tf</b>				

ニーモニック:

BC1F offset

機能:

if FPR[7] = 0 then  
branch

例外:

None

概要:

FPR[7] が 0 の場合, 指定されたアドレスに分岐する. *Responsive Multithreaded Processor* ではステータスレジスタではなく, 浮動小数点レジスタの内容により分岐を判断する.

<b>BC1TL</b>		Branch on FP True Likely	
浮動小数点分岐		MIPS II 動作改	

31	26 25	21 20	18 17 16 15	0
010001	01000	000	1 1	offset
<b>COP1</b>	<b>BC</b>	<b>9</b>	<b>nd tf</b>	

ニーモニック:

BC1TL offset

機能:

if FPR[7] = 1 then  
branch\_likely

例外:

None

概要:

FPR[7] が 1 の場合, 指定されたアドレスに分岐する. *Responsive Multithreaded Processor* ではステータスレジスタではなく, 浮動小数点レジスタの内容により分岐を判断する.

<b>BC1FL</b>		Branch on FP False Likely	
浮動小数点分岐		MIPS II 動作	

31	26 25	21 20	18 17 16 15	0
010001	01000	000	1 0	offset
<b>COP1</b>	<b>BC</b>	<b>9</b>	<b>nd tf</b>	

ニーモニック:

BC1FL offset

機能:

if FPR[7] = 0 then  
branch\_likely

例外:

None

概要:

FPR[7] が 0 の場合, 指定されたアドレスに分岐する. *Responsive Multithreaded Processor* ではステータスレジスタではなく, 浮動小数点レジスタの内容により分岐を判断する.

## 3.2.3 その他の命令

<b>SYNC</b>		Synchronize Operation
命令実行順序制御		MIPS II 動作改
31	26 25	6 5 0
000000	0000000000000000	001111
<b>SPECIAL</b>	<b>0</b>	<b>SYNC</b>

ニーモニック:

SYNC

機能:

synchronize\_operation\_order()

例外:

None

概要:

*Responsive Multithreaded Processor* ではアウトオブオーダーで命令が実行されるが、sync 命令の前後での実行順序が保証される。つまり sync 命令より後の命令は sync 命令より前の命令より先に実行されることはない。また、sync 命令は投機実行されないため、投機実行の制御を行うことができる。

## 3.2.4 サポートしていない MIPS II 命令

*Responsive Multithreaded Processor* は基本的に MIPS II 命令セット互換であるが、いくつかの命令をサポートしていない。以下に MIPS II 命令で *Responsive Multithreaded Processor* がサポートしていない命令を示す。

ニーモニック	概要	
LWC2	Load Word to Coprocessor-2	MIPS I
LWC3	Load Word to Coprocessor-3	MIPS I
SWC2	Store Word to Coprocessor-2	MIPS I
SWC3	Store Word to Coprocessor-3	MIPS I
LDC2	Load Doubleword to Coprocessor-2	MIPS II
LDC3	Load Doubleword to Coprocessor-3	MIPS II
SDC2	Store Doubleword to Coprocessor-2	MIPS II
SDC3	Store Doubleword to Coprocessor-3	MIPS II
MFHI	Move From HI	MIPS I
MTHI	Move To HI	MIPS I
MFLO	Move From LO	MIPS I
MTLO	Move To LO	MIPS I
CTC1	Move Control Word To Floating-Point	MIPS I
CFC1	Move Control Word From Floating-Point	MIPS I
SQRT.fmt	Floating-Point Square Root	MIPS II

### 3.3 Responsive Multithreaded Processor 固有の命令

以下に *Responsive Multithreaded Processor* 固有の命令を示す。

## 3.3.1 Load / Store 命令

<b>IOLB</b>										Load Byte of I/O
I/O 用ロード命令										RESPII
31	26	25	21	20	16	15	6	5	0	
010000		rs		rt		0000000000			110000	
<b>COP0</b>			<b>0</b>				<b>IOLB</b>			

ニーモニック:

IOLB rt, rs

機能:

$GPR[rt] \leftarrow \text{sign\_extend}(\text{MEM.BYTE}[GPR[rs]])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

概要:

指定されたアドレスから 1byte ロードする。通常の Load 命令は投機実行されるが、この命令は投機実行されないため、I/O のように 1 度ロードすると状態が変わるものに用いる。

<b>IOLH</b>				Load Half Word of I/O					
I/O用ロード命令				RESPII					
31	26	25	21	20	16	15	6	5	0
010000		rs		rt		0000000000		110001	
<b>COP0</b>				<b>0</b>				<b>IOLH</b>	

ニーモニック:

IOLH rt, rs

機能:

$GPR[rt] \leftarrow \text{sign\_extend}(\text{MEM.HWORD}[GPR[rs]])$

例外:

D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

概要:

指定されたアドレスから half word ロードする。通常の Load 命令は投機実行されるが、この命令は投機実行されないため、I/O のように 1 度ロードすると状態が変わるものに用いる。

<b>IOLW</b>				Load Word of I/O	
I/O用ロード命令				RESPII	
31	26 25	21 20	16 15	6 5	0
010000	rs	rt	0000000000	110010	
<b>COP0</b>			<b>0</b>	<b>IOLW</b>	

ニーモニック:

IOLW rt, rs

機能:

$GPR[rt] \leftarrow \text{sign\_extend}(\text{MEM.WORD}[GPR[rs]])$

例外:

D-TLB No Entry Matched:

D-TLB Protection Error:

Data Address Miss Align (Load):

概要:

指定されたアドレスから 1word ロードする。通常の Load 命令は投機実行されるが、この命令は投機実行されないため、I/O のように 1 度ロードすると状態が変わるものに用いる。

## 3.3.2 演算命令

<b>DAND</b>						Doubleword And	
64bit 論理積						RESPII	
31	26 25	21 20	16 15	11 10	6 5	0	
000000	rs	rt	rd	00001	100100		
<b>SPECIAL</b>				<b>1</b>	<b>AND</b>		

ニーモニック:

DAND rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \text{ and } FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて論理積を計算する。

<b>DOR</b>						Doubleword Or	
64bit 論理和						RESPII	
31	26 25	21 20	16 15	11 10	6 5	0	
000000	rs	rt	rd	00001	100101		
<b>SPECIAL</b>				<b>1</b>	<b>OR</b>		

ニーモニック:

DOR rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \text{ or } FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて論理和を計算する。

<b>DXOR</b>						Doubleword Exclusive Or					
64bit 排他的論理和						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00001			100110		
SPECIAL						1		XOR			

ニーモニック:

DXOR rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \text{ xor } FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて排他的論理和を計算する。

<b>DNOR</b>						Doubleword Not Or					
64bit 否定論理和						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00001			100111		
SPECIAL						1		NOR			

ニーモニック:

DNOR rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \text{ nor } FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて論理和の否定を計算する。

<b>MULTH</b>						Multiply Word on High Bit					
符号付き乗算						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00001		011000	
SPECIAL						1		MULT			

ニーモニック:

MULTH rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \times GPR[rt]$

例外:

None

概要:

乗算を行う。乗算結果の上位 32bit をデスティネーションレジスタに格納する。

<b>MULTUH</b>						Multiply Word Unsigned on High Bit					
符号無し乗算						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00001		011001	
SPECIAL						1		MULTU			

ニーモニック:

MULTUH rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \times GPR[rt]$

例外:

None

概要:

乗算を行う。値は符号無しとして計算する。乗算結果の上位 32bit をデスティネーションレジスタに格納する。

<b>DMULTH</b>						Doubleword Multiply on High Bit					
符号付き 64bit 乗算						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00001			011100		
SPECIAL						1			DMULT		

ニーモニック:

DMULTH rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \times FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて乗算を行う。乗算結果の上位 64bit をデスティネーションレジスタに格納する。

<b>DMULTUH</b>						Doubleword Multiply Unsigned on High Bit					
符号無し 64bit 乗算						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs	rt	rd	00001			011101		
SPECIAL						1			DMULTU		

ニーモニック:

DMULTUH rd, rs, rt

機能:

$FPR[rd] \leftarrow FPR[rs] \times FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて乗算を行う。値は符号無しとして計算する。乗算結果の上位 64bit をデスティネーションレジスタに格納する。

<b>REM</b>						Reminder Word					
符号付き剰余						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00001		011010	
<b>SPECIAL</b>						<b>1</b>		<b>DIV</b>			

ニーモニック:

REM rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \div GPR[rt]$

例外:

Divide by Zero :

概要:

除算を行う。剰余をデスティネーションレジスタに格納する。

<b>REMU</b>						Reminder Word Unsigned					
符号無し剰余						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00001		011011	
<b>SPECIAL</b>						<b>1</b>		<b>DIVU</b>			

ニーモニック:

REMU rd, rs, rt

機能:

$GPR[rd] \leftarrow GPR[rs] \div GPR[rt]$

例外:

Divide by Zero :

概要:

除算を行う。値は符号無しとして計算する。剰余をデスティネーションレジスタに格納する。

<b>DSL</b>				Doubleword Set on Less Than							
符号付き 64bit 比較				RESPII							
31	26	25	21	20	16	15	11	10	6	5	0
000000				rs	rt	rd	00001		101010		
<b>SPECIAL</b>						<b>1</b>		<b>SLT</b>			

ニーモニック:

DSL rd, rs, rt

機能:

```

if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif

```

例外:

None

概要:

浮動小数点レジスタを用いて比較を行う。

<b>DSL TU</b>						Doubleword Set on Less Than Unsigned					
符号無し 64bit 比較						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs		rt		rd		00001		101011
<b>SPECIAL</b>						<b>1</b>		<b>SLTU</b>			

ニーモニック:

DSL TU rd, rs, rt

機能:

```

if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif

```

例外:

None

概要:

浮動小数点レジスタを用いて比較を行う。値は符号無しとして比較を行う。

<b>RTL</b>						Rotate Left					
左ローテーション						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000			00001		rt		rd		sa		000000
<b>SPECIAL</b>						<b>1</b>		<b>SLL</b>			

ニーモニック:

RTL rd, rt, sa

機能:

```

GPR[rd] ← GPR[rt] <<< sa

```

例外:

None

概要:

左ローテーション演算を行う。

<b>RTR</b>						Rotate Right					
右ローテーション						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000	00001	rt	rd	sa	000010						
<b>SPECIAL</b>			<b>1</b>					<b>SRL</b>			

ニーモニック:

RTR rd, rt, sa

機能:

$GPR[rd] \leftarrow GPR[rt] \ggg sa$

例外:

None

概要:

右ローテーション演算を行う。

<b>RTL</b>						Rotate Left Variable					
左ローテーション						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000	rs	rt	rd	00001	000100						
<b>SPECIAL</b>							<b>1</b>	<b>SLLV</b>			

ニーモニック:

RTL rd, rt, rs

機能:

$GPR[rd] \leftarrow GPR[rt] \lll GPR[rs]$

例外:

None

概要:

左ローテーション演算を行う。

<b>RTRV</b>						Rotate Right Variable					
右ローテーション						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00001		000110	
SPECIAL						1		SRLV			

ニーモニック:

RTRV rd, rt, rs

機能:

$GPR[rd] \leftarrow GPR[rt] \ggg GPR[rs]$

例外:

None

概要:

右ローテーション演算を行う。

<b>DRTL</b>						Doubleword Rotate Left					
64bit 左ローテーション						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000		00001		rt		rd		sa		111000	
SPECIAL		1		DSL							

ニーモニック:

DRTL rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \lll sa$

例外:

None

概要:

浮動小数点レジスタを用いて左ローテーション演算を行う。

<b>DRTR</b>							Doubleword Rotate Right				
64bit 右ローテーション							RESPII				
31	26	25	21	20	16	15	11	10	6	5	0
000000		00001		rt		rd		sa		111010	
SPECIAL		1								DSRL	

ニーモニック:

DRTR rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \ggg sa$

例外:

None

概要:

浮動小数点レジスタを用いて右ローテーション演算を行う。

<b>DRTL32</b>							Doubleword Rotate Left plus 32				
64bit 左ローテーション							RESPII				
31	26	25	21	20	16	15	11	10	6	5	0
000000		00001		rt		rd		sa		111100	
SPECIAL		1								DSLL32	

ニーモニック:

DRTL rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \lll (sa + 32)$

例外:

None

概要:

浮動小数点レジスタを用いて左ローテーション演算を行う。

<b>DRTR32</b>							Doubleword Rotate Right plus 32				
64bit 右ローテーション							RESPII				
31	26	25	21	20	16	15	11	10	6	5	0
000000			00001		rt		rd		sa		111110
SPECIAL			1						DSRL32		

ニーモニック:

DRTR32 rd, rt, sa

機能:

$FPR[rd] \leftarrow FPR[rt] \ggg (sa + 32)$

例外:

None

概要:

浮動小数点レジスタを用いて右ローテーション演算を行う。

<b>DRTL</b>							Doubleword Rotate Left Variable				
64bit 左ローテーション							RESPII				
31	26	25	21	20	16	15	11	10	6	5	0
000000			rs		rt		rd		00001		010100
SPECIAL									1		DSLIV

ニーモニック:

DRTL rd, rt, rs

機能:

$FPR[rd] \leftarrow FPR[rt] \lll FPR[rs]$

例外:

None

概要:

浮動小数点レジスタを用いて左ローテーション演算を行う。

<b>DRTRV</b>						Doubleword Rotate Right Variable					
64bit 右ローテーション						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
000000		rs		rt		rd		00001		010110	
SPECIAL						1		DSRLV			

ニーモニック:

DRTRV rd, rt, rs

機能:

$FPR[rd] \leftarrow FPR[rt] \ggg FPR[rs]$

例外:

None

概要:

浮動小数点レジスタを用いて右ローテーション演算を行う。

### 3.3.3 転送命令

<b>MTC1H</b>						Move Word to Floating Point on High bit					
レジスタ間転送						RESPII					
31	26	25	21	20	16	15	11	10	6	5	0
010001		00100		rt		fs		00001		000000	
COP1		MT						1		0	

ニーモニック:

MTC1H rt, fs

機能:

$FPR[fs] \leftarrow \{GPR[rt], 0^{32}\}$

例外:

None

概要:

浮動小数点レジスタの上位 32 ビットに汎用レジスタの値を転送する。

<b>MFC1H</b>		Move Word from Floating Point on High bit					
レジスタ間転送						RESPII	
31	26 25	21 20	16 15	11 10	6 5	0	
010001	00000	rt	fs	00001	000000		
<b>COP1</b>	<b>MF</b>			<b>1</b>	<b>0</b>		

ニーモニック:

MFC1H rt, fs

機能:

GPR[rt] ← high\_32bit(FPR[fs])

例外:

None

概要:

汎用レジスタに浮動小数点レジスタの上位 32 ビットを転送する。

### 3.3.4 システム制御命令

<b>MFC0</b>		Move from System Control Register					
システムレジスタリード命令						SYSTEM (特権命令)	
31	26 25	21 20	16 15	11 10	6 5	0	
010000	00000	rt	rd	00000	000000		
<b>COP0</b>	<b>MF</b>			<b>0</b>	<b>CTRL</b>		

ニーモニック:

MFC0 rt, rd

機能:

GPR[rt] ← SYSTEM[GPR[rd]]

例外:

Coprocessor Unusable:

概要:

システムレジスタから値を読み込む。システムレジスタのアドレスは GPR[rd] で指定する。

<b>MTC0</b>		Move to System Control Register			
システムレジスタライト命令				SYSTEM (特権命令)	
31	26 25	21 20	16 15	11 10	6 5 0
010000	00100	rt	rd	00000	000000
<b>COP0</b>	<b>MT</b>			<b>0</b>	<b>CTRL</b>

ニーモニック:

MTC0 rt, rd

機能:

SYSTEM[GPR[rd]] ← GPR[rt]

例外:

Coprocessor Unusable:

概要:

システムレジスタに値を書き込む。システムレジスタのアドレスは GPR[rd] で指定する。

<b>MFIMM</b>		Move from Instruction MMU Control Register			
命令 MMU 制御レジスタリード命令				SYSTEM (特権命令)	
31	26 25	21 20	16 15	11 10	6 5 0
010000	00000	rt	rd	00000	000010
<b>COP0</b>	<b>MF</b>			<b>0</b>	<b>IMMU</b>

ニーモニック:

MFIMM rt, rd

機能:

GPR[rt] ← IMMU[GPR[rd]]

例外:

Coprocessor Unusable:

概要:

命令 MMU 制御レジスタから値を読み込む。制御レジスタのアドレスは GPR[rd] で指定する。

<b>MTIMM</b>		Move to Instruction MMU Control Register									
命令 MMU 制御レジスタライト命令						SYSTEM (特権命令)					
31	26	25	21	20	16	15	11	10	6	5	0
010000		00100		rt	rd	00000		000010			
<b>COP0</b>		<b>MT</b>				<b>0</b>		<b>IMMU</b>			

ニーモニック:

MTIMM rt, rd

機能:

IMMU[GPR[rd]] ← GPR[rt]

例外:

Coprocessor Unusable:

概要:

命令 MMU 制御レジスタに値を書き込む。制御レジスタのアドレスは GPR[rd] で指定する。

<b>MFDMM</b>		Move from Data MMU Control Register									
データ MMU 制御レジスタリード命令						SYSTEM					
31	26	25	21	20	16	15	11	10	6	5	0
010000		00000		rt	rd	00000		000011			
<b>COP0</b>		<b>MF</b>				<b>0</b>		<b>DMMU</b>			

ニーモニック:

MFDMM rt, rd

機能:

GPR[rt] ← DMMU[GPR[rd]]

例外:

Coprocessor Unusable:

概要:

データ MMU 制御レジスタから値を読み込む。制御レジスタのアドレスは GPR[rd] で指定する。

<b>MTDMM</b>		Move to Data MMU Control Register			
データ MMU 制御レジスタライト命令				SYSTEM (特権命令)	
31	26 25	21 20	16 15	11 10	6 5 0
010000	00100	rt	rd	00000	000011
<b>COP0</b>	<b>MT</b>			<b>0</b>	<b>DMMU</b>

ニーモニック:

MTDMM rt, rd

機能:

DMMU[GPR[rd]] ← GPR[rt]

例外:

Coprocessor Unusable:

概要:

データ MMU 制御レジスタに値を書き込む。制御レジスタのアドレスは GPR[rd] で指定する。

<b>ERET</b>		Exception Return			
例外復帰命令				SYSTEM	
31	26 25			6 5	0
010000		00000000000000000000			011000
<b>COP0</b>		<b>0</b>			<b>ERET</b>

ニーモニック:

ERET

機能:

*Exception Return*

例外:

None

概要:

例外処理から復帰する。

## 3.3.5 スレッド制御命令

<b>MKTH</b>						Make Thread
スレッド生成						THREAD
31	26 25	21 20	16 15	11 10	6 5	0
011101	rs	rt	rd	00000	000001	
<b>THREAD</b>				<b>0</b>	<b>MKTH</b>	

ニーモニック:

MKTH rd, rs, rt

機能:

```

make_thread(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

スレッドの生成を行う。GPR[rs] に作成するスレッドの ID, GPR[rt] にスタートアドレスを指定する。スレッドの生成に成功すると GPR[rd] に 1 が, 失敗すると 0 が返る。

<b>DELTH</b>				Delete Thread							
スレッド削除				THREAD							
31	26	25	21	20	16	15	11	10	6	5	0
011101				rs		00000		rd		00000	
<b>THREAD</b>				<b>0</b>		<b>0</b>		<b>DELTH</b>			

ニーモニック:

DELTH rd, rs

機能:

```

delete_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

スレッドの削除を行う。GPR[rs] に削除するスレッドの ID を指定する。スレッドの削除に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>CHGPR</b>				Change Priority							
優先度の変更				THREAD							
31	26	25	21	20	16	15	11	10	6	5	0
011101			rs		rt		rd		00000		000011
<b>THREAD</b>				<b>0</b>				<b>CHGPR</b>			

ニーモニック:

CHGPR rd, rs, rt

機能:

```

change_priority(GPR[rs], GPR[rd])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

優先度の変更を行う。GPR[rs] に変更するスレッドの ID, GPR[rt] に新しい優先度を指定する。  
優先度の変更に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>CHGST</b>						Change Status					
状態の変更						THREAD					
31	26	25	21	20	16	15	11	10	6	5	0
011101		rs		rt		rd		00000		000100	
<b>THREAD</b>						<b>0</b>		<b>CHGST</b>			

ニーモニック:

CHGST rd, rs, rt

機能:

```

change_status(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

スレッドの状態を変更する。GPR[rs] に変更するスレッド ID の、GPR[rt] に新しい状態を指定する。状態の変更に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>RUNTH</b>				Run Thread							
スレッドの実行				THREAD							
31	26	25	21	20	16	15	11	10	6	5	0
011101				rs		00000		rd		00000	
<b>THREAD</b>				<b>0</b>		<b>0</b>		<b>RUNTH</b>			

ニーモニック:

RUNTH rd, rs

機能:

```
run_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

例外:

概要:

スレッドを実行状態する。GPR[rs] に実行するスレッドの ID を指定する。スレッドの実行に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>STOPTH</b>				Stop Thread			
スレッドの停止				THREAD			
31	26 25	21 20	16 15	11 10	6 5	0	
011101	rs	00000	rd	00000	000110		
<b>THREAD</b>			<b>0</b>	<b>0</b>		<b>STOPTH</b>	

ニーモニック:

STOPTH rd, rs

機能:

```

stop_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

スレッドを停止状態にする。GPR[rs] に停止するスレッド ID を指定する。スレッドの停止に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>STOPLF</b>					Stop Myself
スレッドの停止					THREAD
31	26 25	16 15	11 10	6 5	0
011101	0000000000	rd	00000	000111	
<b>THREAD</b>	<b>0</b>		<b>0</b>	<b>STOPLF</b>	

ニーモニック:

STOPLF rd

機能:

```

stop_myself()
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

自分自身のスレッドを停止する。スレッドの停止に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>BKUPTH</b>						Backup Thread
スレッドの退避						THREAD
31	26 25	21 20	16 15	11 10	6 5	0
011101	rs	00000	rd	00000	001000	
<b>THREAD</b>		<b>0</b>		<b>0</b>	<b>BKUPTH</b>	

ニーモニク:

BKUPTH rd, rs

機能:

```

backup_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

アクティブスレッドをコンテキストキャッシュに退避する。GPR[rs] に退避するスレッドの ID を指定する。スレッドの退避に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>BKUPSLF</b>		Backup Myself			
スレッドの退避		THREAD			
31	26 25	16 15	11 10	6 5	0
011101	0000000000	rd	00000	001001	
<b>THREAD</b>	<b>0</b>		<b>0</b>	<b>BKUPSLF</b>	

ニーモニック:

BKUPSLF rd

機能:

```

backup_myself()
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

自分自身をコンテキストキャッシュに退避する。スレッドの退避に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>RSTRTH</b>						Restore Thread					
スレッドの復帰						THREAD					
31	26	25	21	20	16	15	11	10	6	5	0
011101	rs		00000		rd		00000		001010		
<b>THREAD</b>			<b>0</b>		<b>0</b>		<b>RSTRTH</b>				

ニーモニック:

RSTRTH rd, rs

機能:

```

restore_thread(GPR[rs])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

キャッシュスレッドをコンテキストキャッシュから復帰する。GPR[rs] に復帰するスレッドの ID を指定する。スレッドの復帰に成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>SWAPTH</b>						Swap Thread
スレッドの入れ換え						THREAD
31	26 25	21 20	16 15	11 10	6 5	0
011101	rs	rt	rd	00000	001011	
<b>THREAD</b>				<b>0</b>	<b>SWAPTH</b>	

ニーモニック:

SWAPTH rd, rs, rt

機能:

```

swap_thread(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

アクティブスレッドとキャッシュスレッドを入れ換える。GPR[rs] に退避するアクティブスレッドの ID, GPR[rt] に復帰するキャッシュスレッドを指定する。スレッドの入れ換えに成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>SWAPSLF</b>				Swap Myself			
スレッドの入れ換え				THREAD			
31	26 25	21 20	16 15	11 10	6 5	0	
011101	00000	rt	rd	00000	001100		
<b>THREAD</b>	<b>0</b>			<b>0</b>	<b>SWAPSLF</b>		

ニーモニック:

```
SWAPSLF rd, rt
```

機能:

```
swap_myself(GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif
```

例外:

概要:

自分自身とキャッシュスレッドを入れ換える。GPR[rt] に復帰するキャッシュスレッドを指定する。スレッドの入れ換えに成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>CPTHTOA</b>						Copy Thread to Active Thread					
スレッドのコピー					THREAD						
31	26	25	21	20	16	15	11	10	6	5	0
011101	rs		rt		rd		00000		001101		
<b>THREAD</b>						<b>0</b>		<b>CPTHTOA</b>			

ニーモニック:

CPTHTOA rd, rs, rt

機能:

```

copy_to_active(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

アクティブスレッドを別のアクティブスレッドとしてコピーする。GPR[rs] にコピー元のアクティブスレッドの ID, GPR[rt] にコピー先のアクティブスレッドの ID を指定する。スレッドのコピーに成功すると GPR[rd] に 1 が、失敗すると 0 が返る。

<b>CPTHTOM</b>						Copy Thread to Cache Thread (Memory)					
スレッドのコピー						THREAD					
31	26	25	21	20	16	15	11	10	6	5	0
011101	rs		rt		rd		00000		001110		
<b>THREAD</b>						<b>0</b>		<b>CPTHTOM</b>			

ニーモニック:

CPTHTOM rd, rs, rt

機能:

```

copy_to_meory(GPR[rs], GPR[rt])
if success_thread_operation then
    GPR[rd] ← 1
else
    GPR[rd] ← 0
endif

```

例外:

概要:

アクティブスレッドを別のキャッシュスレッドとしてコピーする。GPR[rs] にコピー元のアクティブスレッドの ID, GPR[rt] にコピー先のキャッシュスレッドの ID を指定する。スレッドのコピーに成功すると GPR[rd] に 1 が, 失敗すると 0 が返る。

<b>GETTT</b>						Get Thread Table					
スレッドテーブルの参照						THREAD					
31	26	25	21	20	16	15	11	10	6	5	0
011101		rs		00000		rd		00000		001111	
<b>THREAD</b>			<b>0</b>	<b>0</b>			<b>0</b>			<b>GETTT</b>	

ニーモニック:

GETTT rd, rs

機能:

GPR[rd] ← ThreadTable of GPR[rs]

例外:

概要:

スレッドテーブルから値を読み込む。GPR[rs] に読み込むスレッドの ID を指定する。GPR[rd] にスレッドテーブルの内容が返る。

<b>GETTID</b>						Get Thread ID					
スレッド ID の参照						THREAD					
31	26	25	21	20	16	15	11	10	6	5	0
011101		rs		00000		rd		00000		010000	
<b>THREAD</b>			<b>0</b>	<b>0</b>			<b>0</b>			<b>GETTID</b>	

ニーモニック:

GETTID rd, rs

機能:

GPR[rd] ← ThreadID of GPR[rs]

例外:

概要:

コンテキスト ID からスレッド ID を調べる。GPR[rs] に調べるスレッドのコンテキスト ID を指定する。GPR[rd] にスレッド ID が返る。

<b>GETOTID</b>					Get Own Thread ID				
スレッド ID の参照					THREAD				
31	26	25	16	15	11	10	6	5	0
011101	0000000000			rd	00000		010001		
<b>THREAD</b>			<b>0</b>		<b>0</b>		<b>GETOTID</b>		

ニーモニック:

GETOTID rd

機能:

GPR[rd] ← Own ThreadID

例外:

概要:

自分自身のスレッド ID を調べる . GPR[rd] にスレッド ID が返る .

<b>GETMTID</b>						Get Cache Thread ID (Get Memory Thread ID)					
スレッド ID の参照						THREAD					
31	26	25	21	20	16	15	11	10	6	5	0
011101	rs		00000		rd	00000		010010			
<b>THREAD</b>			<b>0</b>			<b>0</b>		<b>GETMTID</b>			

ニーモニック:

GETMTID rd, rs

機能:

GPR[rd] ← ThreadID of GPR[rs]

例外:

概要:

コンテキスト ID からスレッド ID を調べる . GPR[rs] に調べるキャッシュスレッドのコンテキスト ID を指定する . GPR[rd] にスレッド ID が返る .

<b>GETCNUM</b>		Get Context ID Number	
コンテキスト ID の参照		THREAD	
31	26 25	21 20	16 15
11 10	6 5	0	
011101	rs	00000	rd
00000	0		010011
<b>THREAD</b>	<b>0</b>		<b>GETCNUM</b>

ニーモニック:

GETCNUM rd, rs

機能:

GPR[rd] ← ContextID of GPR[rs]

例外:

概要:

スレッド ID からコンテキスト ID を調べる。GPR[rs] に調べるスレッドの ID を指定する。GPR[rd] にコンテキスト ID が返る。8bit 目が 1 の場合、スレッドはアクティブスレッドにあり、2bit 目から 0bit 目にコンテキスト ID が返る。6bit 目が 1 の場合、スレッドはキャッシュスレッドにあり、4bit 目から 0bit 目にコンテキストキャッシュにおけるコンテキスト ID が返る。

## 3.3.6 SIMD 演算命令

<b>SADD.size</b>										SIMD Add			
SIMD 加算										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100		rs		rt		rd		000	size			100000	
<b>SIMD</b>										<b>0</b>		<b>ADD</b>	

ニーモニック:

SADD.8 rd, rs, rt	(size = 01)
SADD.16 rd, rs, rt	(size = 10)
SADD.32 rd, rs, rt	(size = 11)

機能:

$$\text{FPR}[\text{rd}] \leftarrow \text{FPR}[\text{rs}] + \text{FPR}[\text{rt}]$$

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 加算を行う。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

<b>SADD.size.sc</b>										SIMD Add Scalar			
SIMD 加算										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100	rs		rt		rd		000		size	110000			
<b>SIMD</b>							<b>0</b>		<b>ADD.sc</b>				

ニーモニック:

SADD.8.sc rd, rs, rt (size = 01)

SADD.16.sc rd, rs, rt (size = 10)

SADD.32.sc rd, rs, rt (size = 11)

機能:

$FPR[rd] \leftarrow FPR[rs] + FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 加算を行う。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SSUB.size</b>										SIMD Subtract			
SIMD 減算										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100		rs		rt		rd		000		size		100010	
SIMD								0		SUB			

ニーモニック:

SSUB.8 rd, rs, rt (size = 01)

SSUB.16 rd, rs, rt (size = 10)

SSUB.32 rd, rs, rt (size = 11)

機能:

$FPR[rd] \leftarrow FPR[rs] - FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 減算を行う。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

<b>SSUB.size.sc</b>										SIMD Subtract Scalar			
SIMD 減算										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100		rs		rt		rd		000		size		110010	
SIMD								0		SUB.sc			

ニーモニック:

SSUB.8.sc rd, rs, rt (size = 01)

SSUB.16.sc rd, rs, rt (size = 10)

SSUB.32.sc rd, rs, rt (size = 11)

機能:

$FPR[rd] \leftarrow FPR[rs] - FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 減算を行う。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SMULT.size</b>										SIMD Multiply			
符号付き SIMD 乗算										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100	rs		rt		rd		000		size		011000		
<b>SIMD</b>							<b>0</b>		<b>MULT</b>				

## ニーモニック:

SMULT.8 rd, rs, rt (size = 01)

SMULT.16 rd, rs, rt (size = 10)

SMULT.32 rd, rs, rt (size = 11)

## 機能:

$$\text{FPR}[\text{rd}] \leftarrow \text{FPR}[\text{rs}] \times \text{FPR}[\text{rt}]$$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 乗算を行う。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

<b>SMULT.size.sc</b>				SIMD Multiply Scalar			
符号付き SIMD 乗算				SIMD			
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	101000	
SIMD				0		MULT.sc	

## ニーモニック:

SMULT.8.sc rd, rs, rt (size = 01)

SMULT.16.sc rd, rs, rt (size = 10)

SMULT.32.sc rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow FPR[rs] \times FPR[rt]$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 乗算を行う。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SMULTU.size</b>				SIMD Multiply Unsigned			
符号無し SIMD 乗算				SIMD			
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	011001	
SIMD				0		MULTU	

## ニーモニック:

SMULTU.8 rd, rs, rt (size = 01)

SMULTU.16 rd, rs, rt (size = 10)

SMULTU.32 rd, rs, rt (size = 11)

## 機能:

$$\text{FPR}[\text{rd}] \leftarrow \text{FPR}[\text{rs}] \times \text{FPR}[\text{rt}]$$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 乗算を行う。値は符号無し整数として計算する。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。

<b>SMULTU.size.sc</b>				SIMD Multiply Unsigned Scalar			
符号無し SIMD 乗算				SIMD			
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	101001	
SIMD				0		MULTU.sc	

## ニーモニック:

SMULTU.8.sc rd, rs, rt (size = 01)

SMULTU.16.sc rd, rs, rt (size = 10)

SMULTU.32.sc rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow FPR[rs] \times FPR[rt]$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 乗算を行う。値は符号無し整数として計算する。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SAND.size.sc</b>										SIMD And Scalar			
SIMD 論理積										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100	rs		rt		rd		000		size	100100			
<b>SIMD</b>								<b>0</b>		<b>AND.sc</b>			

ニーモニック:

SAND.8.sc rd, rs, rt (size = 01)

SAND.16.sc rd, rs, rt (size = 10)

SAND.32.sc rd, rs, rt (size = 11)

機能:

$FPR[rd] \leftarrow FPR[rs] \text{ and } FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 論理積を計算する。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SOR.size.sc</b>										SIMD Or Scalar			
SIMD 論理和										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100		rs		rt		rd		000		size		100101	
SIMD								0		OR.sc			

## ニーモニック:

SOR.8.sc rd, rs, rt (size = 01)

SOR.16.sc rd, rs, rt (size = 01)

SOR.32.sc rd, rs, rt (size = 01)

## 機能:

$FPR[rd] \leftarrow FPR[rs] \text{ or } FPR[rt]$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 論理和を計算する。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SXOR.size.sc</b>				SIMD Exclusive Or Scalar			
SIMD 排他的論理和				SIMD			
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	100110	
SIMD				0		XOR.sc	

ニーモニック:

SXOR.8.sc rd, rs, rt (size = 01)

SXOR.16.sc rd, rs, rt (size = 10)

SXOR.32.sc rd, rs, rt (size = 11)

機能:

$FPR[rd] \leftarrow FPR[rs] \text{ xor } FPR[rt]$

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 排他的論理和を計算する。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SNOR.size.sc</b>										SIMD Not Or Scalar				
SIMD 否定論理和										SIMD				
31	26 25		21 20		16 15		11 10		8	7	6	5	0	
011100		rs		rt		rd		000		size		100111		
SIMD								0		NOR.sc				

## ニーモニック:

SNOR.8.sc rd, rs, rt (size = 01)

SNOR.16.sc rd, rs, rt (size = 10)

SNOR.32.sc rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow FPR[rs] \text{ nor } FPR[rt]$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 論理和の否定を計算する。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SSLT.size</b>										SIMD Set Less Than			
SIMD 大小判定										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100		rs		rt		rd		000		size		101010	
<b>SIMD</b>								<b>0</b>		<b>SLT</b>			

ニーモニック:

SSLT.8 rd, rs, rt (size = 01)

SSLT.16 rd, rs, rt (size = 10)

SSLT.32 rd, rs, rt (size = 11)

機能:

```

if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif

```

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 大小判定を行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。

<b>SSLT.size.sc</b>				SIMD Set Less Than Scalar			
SIMD 大小判定				SIMD			
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	011010	
SIMD				0		SLT.sc	

ニーモニック:

SSLT.8.sc rd, rs, rt (size = 01)  
 SSLT.16.sc rd, rs, rt (size = 10)  
 SSLT.32.sc rd, rs, rt (size = 11)

機能:

```
if FPR[rs] < FPR[rt] then
  FPR[rd] ← 1
else
  FPR[rd] ← 0
endif
```

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 大小判定を行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SSLTU.size</b>		SIMD Set Less Than Unsigned				
符号無し SIMD 大小判定		SIMD				
31	26 25	21 20	16 15	11 10	8 7 6 5	0
011100	rs	rt	rd	000	size	101011
<b>SIMD</b>				<b>0</b>		<b>SLTU</b>

ニーモニック:

SSLTU.8 rd, rs, rt (size = 01)  
 SSLTU.16 rd, rs, rt (size = 10)  
 SSLTU.32 rd, rs, rt (size = 11)

機能:

```
if FPR[rs] < FPR[rt] then
  FPR[rd] ← 1
else
  FPR[rd] ← 0
endif
```

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 大小判定を行う。値は符号無し整数として比較を行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。

<b>SSLTU.size.sc</b>				SIMD Set Less Than Unsigned Scalar			
符号無し SIMD 大小判定				SIMD			
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	011011	
SIMD				0		SLTU.sc	

## ニーモニック:

SSLTU.8.sc rd, rs, rt (size = 01)  
 SSLTU.16.sc rd, rs, rt (size = 10)  
 SSLTU.32.sc rd, rs, rt (size = 11)

## 機能:

```

if FPR[rs] < FPR[rt] then
    FPR[rd] ← 1
else
    FPR[rd] ← 0
endif
  
```

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 大小判定を行う。値は符号無し整数として比較を行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。





<b>SSRLV.size</b>										SIMD Shift Right Logical Variable																								
SIMD 右論理シフト															SIMD																			
31	26 25					21 20					16 15					11 10					8	7	6	5	0									
011100					rs					rt					rd					000					size					000110				
<b>SIMD</b>															<b>0</b>					<b>SRLV</b>														

## ニーモニック:

SSRLV.8 rd, rt, rs (size = 01)

SSRLV.16 rd, rt, rs (size = 10)

SSRLV.32 rd, rt, rs (size = 11)

## 機能:

$$\text{FPR}[\text{rd}] \leftarrow \text{FPR}[\text{rt}] \gg \text{FPR}[\text{rs}]$$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 右論理シフトを行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。

<b>SSRLV.size.sc</b>						SIMD Shift Right Logical Variable Scalar							
SIMD 右論理シフト						SIMD							
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100		rs		rt		rd		000		size		010110	
SIMD						0						SRLV.sc	

## ニーモニック:

SSRLV.8.sc rd, rt, rs (size = 01)

SSRLV.16.sc rd, rt, rs (size = 10)

SSRLV.32.sc rd, rt, rs (size = 11)

## 機能:

$FPR[rd] \leftarrow FPR[rt] \gg FPR[rs]$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 右論理シフトを行う。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SSRAV.size</b>										SIMD Shift Right Arithmetic Variable																			
SIMD 右算術シフト										SIMD																			
31					26	25														0									
011100				rs				rt				rd				000		size		000111									
SIMD										0										SRAV									

## ニーモニック:

SSRAV.8 rd, rt, rs (size = 01)

SSRAV.16 rd, rt, rs (size = 10)

SSRAV.32 rd, rt, rs (size = 11)

## 機能:

$FPR[rd] \leftarrow FPR[rt] \gg FPR[rs]$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 右算術シフトを行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。

<b>SSRAV.size.sc</b>							SIMD Shift Right Arithmetic Variable Scalar						
SIMD 右算術シフト							SIMD						
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100		rs		rt		rd		000		size		010111	
SIMD							0		SRAV.sc				

## ニーモニック:

SSRAV.8.sc rd, rt, rs (size = 01)

SSRAV.16.sc rd, rt, rs (size = 10)

SSRAV.32.sc rd, rt, rs (size = 11)

## 機能:

$$\text{FPR}[\text{rd}] \leftarrow \text{FPR}[\text{rt}] \gg \text{FPR}[\text{rs}]$$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 右算術シフトを行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SRTL.V.size</b>										SIMD Rotate Left Variable										
SIMD 左ローテーション										SIMD										
31					26	25														0
011100		rs		rt		rd		000		size		000000								
<b>SIMD</b>										<b>0</b>		<b>RTL</b>								

ニーモニック:

SRTL.V.8 rd, rt, rs (size = 01)

SRTL.V.16 rd, rt, rs (size = 10)

SRTL.V.32 rd, rt, rs (size = 11)

機能:

$FPR[rd] \leftarrow FPR[rt] \lll FPR[rs]$

例外:

None

概要:

浮動小数点レジスタを用いて SIMD 左ローテーションを行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。

<b>SRTL.V.size.sc</b>										SIMD Rotate Left Variable Scalar										
SIMD 左ローテーション										SIMD										
31						26	25													0
011100		rs		rt		rd		000		size		010000								
<b>SIMD</b>										<b>0</b>		<b>RTL.sc</b>								

## ニーモニック:

SRTL.V.8.sc rd, rt, rs (size = 01)

SRTL.V.16.sc rd, rt, rs (size = 10)

SRTL.V.32.sc rd, rt, rs (size = 11)

## 機能:

$FPR[rd] \leftarrow FPR[rt] \lll FPR[rs]$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 左ローテーションを行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>SRTRV.size</b>		SIMD Rotate Right Variable					
SIMD 右ローテーション						SIMD	
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	000010	
<b>SIMD</b>				<b>0</b>		<b>RTR</b>	

## ニーモニック:

SRTRV.8 rd, rt, rs (size = 01)

SRTRV.16 rd, rt, rs (size = 10)

SRTRV.32 rd, rt, rs (size = 11)

## 機能:

$$\text{FPR}[\text{rd}] \leftarrow \text{FPR}[\text{rt}] \ggg \text{FPR}[\text{rs}]$$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 右ローテーションを行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。

<b>SRTRV.size.sc</b>		SIMD Rotate Right Variable Scalar				
SIMD 右ローテーション		SIMD				
31	26 25	21 20	16 15	11 10	8 7 6 5	0
011100	rs	rt	rd	000	size	010010
SIMD				0		RTR.sc

## ニーモニック:

SRTRV.8.sc rd, rt, rs (size = 01)

SRTRV.16.sc rd, rt, rs (size = 10)

SRTRV.32.sc rd, rt, rs (size = 11)

## 機能:

$$\text{FPR}[\text{rd}] \leftarrow \text{FPR}[\text{rt}] \gg \text{FPR}[\text{rs}]$$

## 例外:

None

## 概要:

浮動小数点レジスタを用いて SIMD 右ローテーションを行う。8 は 8bit × 8 演算, 16 は 16bit × 4 演算, 32 は 32bit × 2 演算。FPR[rt] の下位 bit が各フィールドに複製されて演算される。

<b>PCK</b>										Pack Data			
データパッキング										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100	rs		rt		rd		000		size		111000		
<b>SIMD</b>							<b>0</b>		<b>PCK</b>				

## ニーモニック:

PCK.8 rd, rs, rt (size = 01)

PCK.16 rd, rs, rt (size = 10)

PCK.32 rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow \text{pack}(FPR[rs], FPR[rt])$

## 例外:

None

## 概要:

浮動小数点レジスタにデータをパッキングする。rs, rt に指定された浮動小数点レジスタを size の大きさに分割し、区切ったフィールドの下位半分を集めて 1 つにまとめる。

<b>PCKH</b>										Pack Data on High Bit																			
データパッキング										SIMD																			
31	26 25				21 20				16 15				11 10				8	7	6	5	0								
011100	rs				rt				rd				000				size				111001								
<b>SIMD</b>										<b>0</b>										<b>PCKH</b>									

ニーモニック:

PCKH.8 rd, rs, rt (size = 01)

PCKH.16 rd, rs, rt (size = 10)

PCKH.32 rd, rs, rt (size = 11)

機能:

$FPR[rd] \leftarrow packh(FPR[rs], FPR[rt])$

例外:

None

概要:

浮動小数点レジスタにデータをパッキングする。rs, rt に指定された浮動小数点レジスタを size の大きさで分割し、区切ったフィールドの上位半分を集めて1つにまとめる。

<b>CAT1</b>										Concatenate Data Type1			
データ結合										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100	rs		rt		rd		000		size		111010		
<b>SIMD</b>							<b>0</b>		<b>CAT1</b>				

## ニーモニック:

CAT1.8 rd, rs, rt (size = 01)

CAT1.16 rd, rs, rt (size = 10)

CAT1.32 rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow cat1(FPR[rs], FPR[rt])$

## 例外:

None

## 概要:

データを結合して浮動小数点レジスタに入れる。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

<b>CAT1H</b>										Concatenate Data Type1 on High Bit											
データ結合										SIMD											
31	26	25	21	20	16	15	11	10	8	7	6	5								0	
011100				rs				rt				rd				000		size		111011	
SIMD										0 CAT1H											

## ニーモニック:

CAT1H.8 rd, rs, rt (size = 01)

CAT1H.16 rd, rs, rt (size = 10)

CAT1H.32 rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow \text{cat1h}(FPR[rs], FPR[rt])$

## 例外:

None

## 概要:

データを結合して浮動小数点レジスタに入れる。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

<b>CAT2</b>										Concatenate Data Type2											
データ結合										SIMD											
31						26	25													0	
011100				rs				rt				rd				000		size		111100	
SIMD										CAT2											

## ニーモニック:

CAT2.8 rd, rs, rt (size = 01)

CAT2.16 rd, rs, rt (size = 10)

CAT2.32 rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow \text{cat2}(FPR[rs], FPR[rt])$

## 例外:

None

## 概要:

データを結合して浮動小数点レジスタに入れる。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

CAT2H		Concatenate Data Type2 on High Bit					
データ結合						SIMD	
31	26 25	21 20	16 15	11 10	8 7 6 5	0	
011100	rs	rt	rd	000	size	111101	
SIMD				0		CAT2H	

## ニーモニック:

CAT2H.8 rd, rs, rt (size = 01)

CAT2H.16 rd, rs, rt (size = 10)

CAT2H.32 rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow \text{cat2h}(FPR[rs], FPR[rt])$

## 例外:

None

## 概要:

データを結合して浮動小数点レジスタに入れる。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

<b>CAT3</b>										Concatenate Data Type3			
データ結合										SIMD			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011100	rs		rt		rd		000		size		111110		
<b>SIMD</b>							<b>0</b>		<b>CAT3</b>				

## ニーモニック:

CAT3.8 rd, rs, rt (size = 01)

CAT3.16 rd, rs, rt (size = 10)

CAT3.32 rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow \text{cat3}(FPR[rs], FPR[rt])$

## 例外:

None

## 概要:

データを結合して浮動小数点レジスタに入れる。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

<b>CAT3H</b>										Concatenate Data Type3 on High Bit									
データ結合										SIMD									
31	26	25	21	20	16	15	11	10	8	7	6	5	0						
011100				rs		rt		rd		000		size		111111					
SIMD										0		CAT3H							

## ニーモニック:

CAT3H.8 rd, rs, rt (size = 01)

CAT3H.16 rd, rs, rt (size = 10)

CAT3H.32 rd, rs, rt (size = 11)

## 機能:

$FPR[rd] \leftarrow \text{cat3h}(FPR[rs], FPR[rt])$

## 例外:

None

## 概要:

データを結合して浮動小数点レジスタに入れる。8 は 8bit × 8 演算，16 は 16bit × 4 演算，32 は 32bit × 2 演算。

## 3.3.7 同期命令

<b>RGPSH</b>		Read Shared(General Purpose Register)				
同期命令					SYNC	
31	26 25	21 20	16 15	11 10	6 5	0
010000	00000	rt	ss	00000	100000	
<b>COP0</b>		<b>MF</b>		<b>0</b>		<b>GPSHR</b>

ニーモニック:

RGPSH rt, ss

機能:

$GPR[rt] \leftarrow SHARE[ss]$

例外:

None

概要:

共有レジスタから汎用レジスタに値を読み込む。

<b>WGPSH</b>		Write Shared(General Purpose Register)				
同期命令					SYNC	
31	26 25	21 20	16 15	11 10	6 5	0
010000	00100	rt	sd	00000	100000	
<b>COP0</b>		<b>MT</b>		<b>0</b>		<b>GPSHR</b>

ニーモニック:

WGPSH rt, sd

機能:

$SHARE[sd] \leftarrow GPR[rt]$

例外:

None

概要:

汎用レジスタから共有レジスタに値を書き込む。

<b>RFPSH</b>							Read Shared(Floating-Point Register)				
同期命令							SYNC				
31	26	25	21	20	16	15	11	10	6	5	0
010000		00000		rt		ss		00000		100100	
<b>COP0</b>		<b>MF</b>						<b>0</b>		<b>FPSHR</b>	

ニーモニック:

RFPSH rt, ss

機能:

FPR[rt] ← SHARE[ss]

例外:

None

概要:

共有レジスタから浮動小数点レジスタに値を読み込む。

<b>WFPSH</b>							Write Shared(Floating-Point Register)				
同期命令							SYNC				
31	26	25	21	20	16	15	11	10	6	5	0
010000		00100		rt		sd		00000		100100	
<b>COP0</b>		<b>MT</b>						<b>0</b>		<b>FPSHR</b>	

ニーモニック:

WFPSH rt, sd

機能:

SHARE[sd] ← FPR[rt]

例外:

None

概要:

浮動小数点レジスタから共有レジスタに値を書き込む。

<b>RGPEX</b>		Read Exclusive(General Purpose Register)			
同期命令					SYNC
31	26 25	21 20	16 15	11 10	6 5 0
010000	00000	rt	ss	00000	100001
<b>COP0</b>	<b>MF</b>			<b>0</b>	<b>GPLOCK</b>

ニーモニック:

RGPEX rt, ss

機能:

GPR[rt] ← SHARE[ss]

例外:

None

概要:

共有レジスタから汎用レジスタに値を読み込む。

<b>WGPEX</b>		Write Exclusive(General Purpose Register)			
同期命令					SYNC
31	26 25	21 20	16 15	11 10	6 5 0
010000	00100	rt	sd	00000	100001
<b>COP0</b>	<b>MT</b>			<b>0</b>	<b>GPLOCK</b>

ニーモニック:

WGPEX rt, sd

機能:

SHARE[sd] ← GPR[rt]

例外:

None

概要:

汎用レジスタから共有レジスタに値を書き込む。

<b>RFPEX</b>		Read Exclusive(Floating-Point Register)			
同期命令					SYNC
31	26 25	21 20	16 15	11 10	6 5 0
010000	00000	rt	ss	00000	100101
<b>COP0</b>	<b>MF</b>			<b>0</b>	<b>FPLOCK</b>

ニーモニック:

RFPEX rt, ss

機能:

FPR[rt] ← SHARE[ss]

例外:

None

概要:

共有レジスタから浮動小数点レジスタに値を読み込む。

<b>WFPEX</b>		Write Exclusive(Floating-Point Register)			
同期命令					SYNC
31	26 25	21 20	16 15	11 10	6 5 0
010000	00100	rt	sd	00000	100101
<b>COP0</b>	<b>MT</b>			<b>0</b>	<b>FPLOCK</b>

ニーモニック:

WFPEX rt, sd

機能:

SHARE[sd] ← FPR[rt]

例外:

None

概要:

浮動小数点レジスタから共有レジスタに値を書き込む。

<b>GPCO</b>		Read Consumer(General Purpose Register)	
同期命令			SYNC
31	26 25	21 20	16 15
010000	00000	rt	ss
<b>COP0</b>	<b>MF</b>		<b>GPPRCO</b>

**ニーモニック:**

GPCO rt, ss, tid

**機能:**

GPR[rt] ← SHARE[ss]

**例外:**

None

**概要:**

共有レジスタから汎用レジスタに値を読み込む。tid にはスレッドの ID を指定する。

<b>GPPR</b>		Write Producer(General Purpose Register)	
同期命令			SYNC
31	26 25	21 20	16 15
010000	00100	rt	ss
<b>COP0</b>	<b>MT</b>		<b>GPPRCO</b>

**ニーモニック:**

GPPR rt, sd, tid

**機能:**

SHARE[sd] ← GPR[rt]

**例外:**

None

**概要:**

汎用レジスタから共有レジスタに値を書き込む。tid にはスレッドの ID を指定する。

<b>FPCO</b>		Read Consumer(Floating-Point Register)			
同期命令					SYNC
31	26 25	21 20	16 15	11 10	6 5 0
010000	00000	rt	ss	tid	100110
<b>COP0</b>	<b>MF</b>				<b>FPPRCO</b>

ニーモニック:

FPCO rt, ss, tid

機能:

FPR[rt] ← SHARE[ss]

例外:

None

概要:

共有レジスタから浮動小数点レジスタに値を読み込む。tid にはスレッドの ID を指定する。

<b>FPPR</b>		Write Producer(Floating-Point Register)			
同期命令					SYNC
31	26 25	21 20	16 15	11 10	6 5 0
010000	00100	rt	ss	tid	100110
<b>COP0</b>	<b>MT</b>				<b>FPPRCO</b>

ニーモニック:

FPPR rt, sd, tid

機能:

SHARE[sd] ← FPR[rt]

例外:

None

概要:

浮動小数点レジスタから共有レジスタに値を書き込む。tid にはスレッドの ID を指定する。

<b>BAR</b>						Barrier
同期命令						SYNC
31	26 25	21 20	16 15	11 10	6 5	0
010000	00100	rt	sd	00000	100011	
<b>COP0</b>	<b>MT</b>			<b>0</b>	<b>BARRIER</b>	

ニーモニック:

BAR rt, sd

機能:

SHARE[sd] ← GPR[rt] + 1

例外:

None

概要:

—

<b>PBAR</b>						Pre Barrier
同期命令						SYNC
31	26 25	21 20	16 15	11 10	6 5	0
010000	000000	00000	sd	00000	100011	
<b>COP0</b>	<b>MF</b>	<b>0</b>		<b>0</b>	<b>BARRIER</b>	

ニーモニック:

PBAR sd

機能:

—

例外:

None

概要:

—

## 3.3.8 整数ベクトル命令

<b>VADD</b>										Vector Add			
ベクトル加算										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		000	s0	s	100000			
<b>VINT</b>							<b>0</b>		<b>ADD</b>				

ニーモニック:

VADD.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VADD.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VADD.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VADD.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] + VGPR[rt]$

例外:

Vector Integer Exception:

概要:

ベクトル加算 . s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う . sync が 1 の場合、投機実行を抑制する .

<b>VSUB</b>											Vector Subtract			
ベクトル減算											VECTOR			
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
011110	rs		rt		rd		00	s1	s0	s	100010			
<b>VINT</b>							<b>0</b>			<b>SUB</b>				

ニーモニック:

VSUB.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VSUB.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VSUB.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VSUB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VSUB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VSUB.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] - VGPR[rt]$

例外:

Vector Integer Exception:

概要:

ベクトル減算 . s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う . s1 が 1 の場合は  $VGPR[rs]$  の代わりに、スカラレジスタ ( $SGPR[rs]$ ) を用いて演算を行う . sync が 1 の場合は、投機実行を抑制する .

<b>VMULT</b>										Vector Multiply			
ベクトル乗算										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		000	s0	s	011000			
<b>VINT</b>							<b>0</b>	<b>MULT</b>					

**ニーモニック:**

VMULT.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMULT.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMULT.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMULT.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

**機能:**

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

**例外:**

Vector Integer Exception :

**概要:**

符号付きベクトル乗算。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMULTU</b>										Vector Multiply Unsigned			
ベクトル乗算										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		000	s0	s	011001			
<b>VINT</b>							<b>0</b>			<b>MULTU</b>			

ニーモニック:

VMULTU.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMULTU.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMULTU.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMULTU.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

例外:

Vector Integer Exception:

概要:

符号無しベクトル乗算。s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMULTH</b>										Vector Multiply on High Bit									
ベクトル乗算										VECTOR									
31	26 25		21 20		16 15		11 10		8	7	6	5	0						
011110	rs		rt		rd		000		s0	s	010000								
<b>VINT</b>										<b>0</b>		<b>MULTH</b>							

ニーモニック:

VMULTH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMULTH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMULTH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMULTH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

例外:

Vector Integer Exception:

概要:

符号付きベクトル乗算。演算結果の上位 bit(63-32bit) が VGPR[rd] に格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMULTUH</b>										Vector Multiply Unsigned on High Bit									
ベクトル乗算										VECTOR									
31	26 25		21 20		16 15		11 10		8	7	6	5				0			
011110	rs		rt		rd		000		s0	s				010001					
<b>VINT</b>										<b>0</b>			<b>MULTUH</b>						

ニーモニック:

VMULTUH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMULTUH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMULTUH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMULTUH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

例外:

Vector Integer Exception:

概要:

符号無しベクトル乗算。演算結果の上位 bit(63-32bit) が VGPR[rd] に格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VDIV</b>											Vector Divide			
ベクトル除算											VECTOR			
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
011110	rs		rt		rd		00	s1	s0	s	011010			
<b>VINT</b>							<b>0</b>				<b>DIV</b>			

ニーモニック:

VDIV.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIV.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIV.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIV.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIV.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIV.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \div VGPR[rt]$

例外:

Vector Integer Exception:

概要:

符号付きベクトル除算。s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う。s1 が 1 の場合は  $VGPR[rs]$  の代わりに、スカラレジスタ ( $SGPR[rs]$ ) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VDIVU</b>											Vector Divide Unsigned																					
ベクトル除算											VECTOR																					
31						26 25						21 20						16 15						11 10 9	8	7	6	5	0			
011110					rs					rt					rd					00		s1	s0	s	011011							
<b>VINT</b>											<b>0</b>											<b>DIVU</b>										

## ニーモニック:

VDIVU.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIVU.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIVU.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIVU.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIVU.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIVU.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \div \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

符号無しベクトル除算。s0 が 1 の場合は VGPR[rt] の代わりに、スカラーレジスタ (SGPR[rt]) を用いて演算を行う。s1 が 1 の場合は VGPR[rs] の代わりに、スカラーレジスタ (SGPR[rs]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VREM</b>											Vector Reminder			
ベクトル剰余											VECTOR			
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
011110	rs		rt		rd		00	s1	s0	s	010010			
<b>VINT</b>							<b>0</b>			<b>REM</b>				

## ニーモニック:

VREM.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VREM.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VREM.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VREM.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VREM.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VREM.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \div \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

符号付きベクトル剰余。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。s1 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VREMU</b>										Vector Reminder Unsigned				
ベクトル剰余										VECTOR				
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
011110	rs		rt		rd		00	s1	s0	s	010011			
<b>VINT</b>							<b>0</b>				<b>REMU</b>			

## ニーモニック:

VREMU.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VREMU.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VREMU.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VREMU.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VREMU.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VREMU.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \div \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

符号無しベクトル剰余。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。s1 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMADD</b>										Vector Multiply and Add												
ベクトル積和演算										VECTOR												
31	26	25	21	20	16	15	11	10	8	7	6	5	0									
011110			rs			rt			rd			000		s0	s	100001						
VINT										0 MADD												

## ニーモニック:

VMADD.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMADD.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMADD.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMADD.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}] + \text{VGPR}[\text{rd}]$$

## 例外:

Vector Integer Exception:

## 概要:

ベクトル積和演算 . s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う . sync が 1 の場合は、投機実行を抑制する .

<b>VMSUB</b>										Vector Multiply and Subtract													
ベクトル積差演算										VECTOR													
31	26	25	21	20	16	15	11	10	8	7	6	5								0			
011110				rs				rt				rd				000		s0	s	100011			
VINT										0 MSUB													

ニーモニック:

VMSUB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMSUB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMSUB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMSUB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt] - VGPR[rd]$

例外:

Vector Integer Exception:

概要:

ベクトル積差演算。s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VACC</b>										Vector Accumulate									
ベクトル累算										VECTOR									
31	26	25	21	20	16	15	11	10	7	6	5				0				
011110	rs			rt			rd			0000	s	001010							
VINT										0			ACC						

ニーモニック:

VACC rd, rs (sync(s) = 0)

VACC.sy rd, rs (sync(s) = 1)

機能:

$SGPR[rd] \leftarrow \sum VGPR[rs]$

例外:

Vector Integer Exception:

概要:

ベクトル累算。ベクトルの要素を全て加算する。sync が 1 の場合は、投機実行を抑制する。

<b>VMAC</b>										Vector Multiply and Accumulate																			
ベクトル積和演算										VECTOR																			
31						26 25						21 20						16 15						11 10	8	7	6	5	0
011110					rs					rt					rd					000		s0	s	001011					
VINT										0										MAC									

## ニーモニック:

VMAC.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMAC.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMAC.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMAC.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

$$\text{SGPR}[\text{rd}] \leftarrow \sum \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

ベクトル積和演算 . 2つのベクトル要素を乗算し,それを全て加算する .s0が1の場合はVGPR[rt]の代わりに, スカラレジスタ (SGPR[rt]) を用いて演算を行う . sync が 1 の場合は, 投機実行を抑制する .

<b>VAND</b>										Vector And			
ベクトル論理積										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		000	s0	s	100100			
<b>VINT</b>							<b>0</b>	<b>AND</b>					

**ニーモニック:**

VAND.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VAND.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VAND.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VAND.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

**機能:**

$VGPR[rd] \leftarrow VGPR[rs] \text{ and } VGPR[rt]$

**例外:**

Vector Integer Exception :

**概要:**

ベクトル論理積 . s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う . sync が 1 の場合は、投機実行を抑制する .

<b>VOR</b>										Vector Or			
ベクトル論理和										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		000	s0	s	100101			
<b>VINT</b>							<b>0</b>		<b>OR</b>				

ニーモニック:

VOR rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VOR.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VOR.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VOR.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \text{ or } VGPR[rt]$

例外:

Vector Integer Exception:

概要:

ベクトル論理和 . s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う . sync が 1 の場合は、投機実行を抑制する .

<b>VXOR</b>										Vector Exclusive Or			
ベクトル排他的論理和										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		000		s0	s	100110		
<b>VINT</b>							<b>0</b>		<b>XOR</b>				

ニーモニック:

VXOR.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VXOR.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VXOR.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VXOR.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \text{ xor } VGPR[rt]$

例外:

Vector Integer Exception:

概要:

ベクトル排他的論理和。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VNOR</b>										Vector Not Or			
ベクトル否定論理和										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		000	s0	s	100111			
<b>VINT</b>							<b>0</b>		<b>NOR</b>				

ニーモニック:

VNOR.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VNOR.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VNOR.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VNOR.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \text{ nor } VGPR[rt]$

例外:

Vector Integer Exception:

概要:

ベクトル否定論理和。s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VSLLV</b>										Vector Shift Left Logical Variable										
ベクトル左論理シフト										VECTOR										
31	26	25	21	20	16	15	11	10	8	7	6	5								0
011110		rs			rt		rd			000		s0	s	000100						
<b>VINT</b>										<b>0</b>			<b>SLLV</b>							

## ニーモニック:

VSLLV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSLLV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSLLV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSLLV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rt}] \ll \text{VGPR}[\text{rs}]$$

## 例外:

Vector Integer Exception:

## 概要:

ベクトル左論理シフト。s0 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VSRLV</b>										Vector Shift Right Logical Variable										
ベクトル右論理シフト										VECTOR										
31	26	25	21	20	16	15	11	10	8	7	6	5								0
011110				rs		rt		rd		000		s0	s	000110						
<b>VINT</b>										<b>0</b>		<b>SRLV</b>								

ニーモニック:

VSRLV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSRLV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSRLV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSRLV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rt] \gg VGPR[rs]$

例外:

Vector Integer Exception:

概要:

ベクトル右論理シフト。s0 が 1 の場合は  $VGPR[rs]$  の代わりに、スカラレジスタ ( $SGPR[rs]$ ) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。



<b>VRTLV</b>										Vector Rotate Left Variable										
ベクトル左ローテーション										VECTOR										
31	26	25	21	20	16	15	11	10	8	7	6	5								0
011110		rs			rt		rd		000		s0	s	000000							
VINT										0			SRTL							

ニーモニック:

VRTLV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VRTLV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VRTLV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VRTLV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rt] \lll VGPR[rs]$

例外:

Vector Integer Exception:

概要:

ベクトル左ローテーション .s0 が 1 の場合は  $VGPR[rs]$  の代わりに、スカラレジスタ ( $SGPR[rs]$ ) を用いて演算を行う .sync が 1 の場合は、投機実行を抑制する .

<b>VRTRV</b>										Vector Rotate Right Variable									
ベクトル右ローテーション										VECTOR									
31	26 25		21 20		16 15		11 10		8	7	6	5				0			
011110	rs		rt		rd		000		s0	s	000010								
<b>VINT</b>										<b>0</b>			<b>SRTRV</b>						

ニーモニック:

VRTRV.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VRTRV.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VRTRV.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VRTRV.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rt] \ggg VGPR[rs]$

例外:

Vector Integer Exception:

概要:

ベクトル右ローテーション .s0 が 1 の場合は  $VGPR[rs]$  の代わりに、スカラレジスタ ( $SGPR[rs]$ ) を用いて演算を行う .sync が 1 の場合は、投機実行を抑制する .

<b>VCMP</b>										Vector Compare			
ベクトル比較										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011110	rs		rt		rd		cond	s0	s	101000			
VINT										CMP			

## ニーモニック:

VCMP.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

```

## 例外:

Vector Integer Exception :

## 概要:

ベクトル比較命令 . 条件 (cond) により VGPR[rd] に 1 または 0 が入る . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . sync が 1 の場合は , 投機実行を抑制する . 比較条件 (cond) には , eq(=) , gt(>) , lt(<) , ne(≠) , le(≤) , ge(≥) のどれかを指定する .

<b>VCMPU</b>										Vector Compare Unsigned												
ベクトル比較										VECTOR												
31	26	25	21	20	16	15	11	10	8	7	6	5								0		
011110			rs			rt			rd			cond		s0	s	101001						
VINT										CMPU												

## ニーモニック:

VCMPU.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPU.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPU.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPU.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

```

if VGR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VREG[rd] ← 0
endif

```

## 例外:

Vector Integer Exception:

## 概要:

符号無しベクトル比較命令。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VCMPTS</b>										Vector Compare to Scalar Register																		
ベクトル比較										VECTOR																		
31						26 25						21 20						16 15						11 10			8 7 6 5	0
011110					rs					rt					rd					cond		s0	s	101010				
VINT										CMPTS																		

ニーモニック:

VCMPTS.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

ベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VCMPUTS</b>										Vector Compare Unsigned to Scalar Register									
ベクトル比較										VECTOR									
31	26 25		21 20		16 15		11 10		8	7	6	5							0
011110	rs		rt		rd		cond		s0	s							101011		
VINT										CMPUTS									

ニーモニック:

VCMPUTS.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPUTS.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPUTS.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPUTS.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

符号無しベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VIRSH</b>										Vector Register Shift		
ベクトルレジスタシフト										VECTOR (MRMTP2 以降)		
31	26	25	21	20	16	15	11	10	7	6	5	0
011110		rs		rt		rd		0000	s		010100	
<b>VINT</b>								<b>0</b>		<b>RSH</b>		

ニーモニック:

VIRSH rd, rs, rt (sync(s) = 0)  
 VIRSH.sy rd, rs, rt (sync(s) = 1)

機能:

$VREG[rd][i] \leftarrow VREG[rs][i - SREG[rt]]$

例外:

Vector Integer Exception:

概要:

レジスタの要素をスカラレジスタ rt の分だけシフトする。デスティネーションレジスタの要素 i にはスカラレジスタ rt の分だけ減算した要素番号の値が入る。シフト値はマイナスの値も指定することが可能。ソースレジスタの要素番号がマイナスおよびベクトル長よりも大きくなった場合、デスティネーションレジスタには 0 が入る。

<b>VIRROT</b>										Vector Register Rotation		
ベクトルレジスタローテーション										VECTOR (MRMTP2 以降)		
31	26	25	21	20	16	15	11	10	7	6	5	0
011110	rs		rt		rd		0000		s	010101		
<b>VINT</b>							<b>0</b>		<b>RROT</b>			

ニーモニック:

VIRROT rd, rs, rt (sync(s) = 0)  
 VIRROT.sy rd, rs, rt (sync(s) = 1)

機能:

$VREG[rd][i] \leftarrow VREG[rs][(i - SREG[rt]) \bmod LENGTH]$

例外:

Vector Integer Exception:

概要:

レジスタの要素をスカラレジスタ *rt* の分だけローテーションする。デスティネーションレジスタの要素 *i* にはスカラレジスタ *rt* の分だけ減算し、ベクトル長で除算した剰余の要素番号の値が入る。シフト値はマイナスの値も指定することが可能。

<b>VIRPK</b>										Vector Register Packing														
ベクトルレジスタパッキング										VECTOR (MRMTP2 以降)														
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0										
011110	rs				00000	rd				00	si	b	s	010110										
<b>VINT</b>					<b>0</b>					<b>0</b>					<b>RPK</b>									

## ニーモニック:

VIRPK rd, rs	(sync(s) = 0, high(h) = 0, byte(b) = 0)
VIRPK.sy rd, rs	(sync(s) = 1, high(h) = 0, byte(b) = 0)
VIRPK.b rd, rs	(sync(s) = 0, high(h) = 0, byte(b) = 1)
VIRPK.b.sy rd, rs	(sync(s) = 1, high(h) = 0, byte(b) = 1)
VIRPK.h rd, rs	(sync(s) = 0, high(h) = 1, byte(b) = 0)
VIRPK.h.sy rd, rs	(sync(s) = 1, high(h) = 1, byte(b) = 0)
VIRPK.b.h rd, rs	(sync(s) = 0, high(h) = 1, byte(b) = 1)
VIRPK.b.h.sy rd, rs	(sync(s) = 1, high(h) = 1, byte(b) = 1)

## 機能:

$VREG[rd][i/2] \leftarrow \{ VREG[rs][i]_{15...0}, VREG[rs][i+1]_{15...0} \}$  (high = 0, byte = 0)  
 $VREG[rd][i/2] \leftarrow \{ VREG[rs][i]_{31...16}, VREG[rs][i+1]_{31...16} \}$  (high = 1, byte = 0)  
 $VREG[rd][i/2] \leftarrow \{ VREG[rs][i]_{23...16}, VREG[rs][i+1]_{23...16}, VREG[rs][i]_{7...0}, VREG[rs][i+1]_{7...0} \}$   
 (high = 0, byte = 1)  
 $VREG[rd][i/2] \leftarrow \{ VREG[rs][i]_{31...24}, VREG[rs][i+1]_{31...24}, VREG[rs][i]_{15...8}, VREG[rs][i+1]_{15...8} \}$   
 (high = 1, byte = 1)

## 例外:

Vector Integer Exception :

## 概要:

ベクトルレジスタの要素  $i$  と  $i+1$  をパッキングし  $i$  番目のレジスタに格納する。この結果、デスティネーションのベクトルレジスタのベクトル長は元のレジスタのベクトル長の半分になる。sync が 1 の場合は、投機実行を抑制する。

<b>VIRUPK</b>										Vector Register Unpacking				
ベクトルレジスタアンパッキング										VECTOR (MRMTP2 以降)				
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
011110	rs		00000		rd		00	h	b	s	010111			
<b>VINT</b>					<b>0</b>		<b>0</b>			<b>RUPK</b>				

### ニーモニック:

VIRUPK rd, rs	(sync(s) = 0, signed(si) = 0, byte(b) = 0)
VIRUPK.sy rd, rs	(sync(s) = 1, signed(si) = 0, byte(b) = 0)
VIRUPK.b rd, rs	(sync(s) = 0, signed(si) = 0, byte(b) = 1)
VIRUPK.b.sy rd, rs	(sync(s) = 1, signed(si) = 0, byte(b) = 1)
VIRUPK.s rd, rs	(sync(s) = 0, signed(si) = 1, byte(b) = 0)
VIRUPK.s.sy rd, rs	(sync(s) = 1, signed(si) = 1, byte(b) = 0)
VIRUPK.b.s rd, rs	(sync(s) = 0, signed(si) = 1, byte(b) = 1)
VIRUPK.b.s.sy rd, rs	(sync(s) = 1, signed(si) = 1, byte(b) = 1)

### 機能:

$VREG[rd][i*2] \leftarrow zero\_ext( VREG[rs][i]_{31...16} )$ ,  
 $VREG[rd][i*2+1] \leftarrow zero\_ext( VREG[rs][i]_{15...0} )$  (signed = 0, byte = 0)  
 $VREG[rd][i*2] \leftarrow sign\_ext( VREG[rs][i]_{31...16} )$ ,  
 $VREG[rd][i*2+1] \leftarrow sign\_ext( VREG[rs][i]_{15...0} )$  (signed = 1, byte = 0)  
 $VREG[rd][i*2] \leftarrow \{ zero\_ext( VREG[rs][i]_{31...24} ), zero\_ext( VREG[rs][i]_{15...8} ) \}$ ,  
 $VREG[rd][i*2+1] \leftarrow \{ zero\_ext( VREG[rs][i]_{23...16} ), zero\_ext( VREG[rs][i]_{7...0} ) \}$  (signed = 0, byte = 1)  
 $VREG[rd][i*2] \leftarrow \{ sign\_ext( VREG[rs][i]_{31...24} ), sign\_ext( VREG[rs][i]_{15...8} ) \}$ ,  
 $VREG[rd][i*2+1] \leftarrow \{ sign\_ext( VREG[rs][i]_{23...16} ), sign\_ext( VREG[rs][i]_{7...0} ) \}$  (signed = 1, byte = 1)

### 例外:

Vector Integer Exception :

### 概要:

ベクトルレジスタの要素  $i$  をアンパッキングし  $i*2$  番目と  $i*2+1$  番目のレジスタに格納する。この結果、デスティネーションのベクトルレジスタのベクトル長は元のレジスタのベクトル長の倍になる。sync が 1 の場合は、投機実行を抑制する。

<b>VIMFC</b>										Move from Vector Integer Control Register									
制御レジスタリード										VECTOR									
31	26	25	21	20	16	15	11	10	7	6	5				0				
011110	rs			00000			rd			0000			s	110000					
VINT					0					0					MFC				

## ニーモニック:

VIMFC rd, rs (sync(s) = 0)  
 VIMFC.sy rd, rs (sync(s) = 1)

## 機能:

$GPR[rd] \leftarrow VICTRL[rs]$

## 例外:

## 概要:

整数ベクトル制御レジスタリード命令。rs で指定された制御レジスタの値を汎用レジスタに格納する。sync が 1 の場合、投機実行を抑制する。

<b>VIMTC</b>										Move to Vector Integer Control Register									
制御レジスタライト										VECTOR									
31	26	25	21	20	16	15	11	10	7	6	5				0				
011110	rs		00000		rd		0000		s	110001									
VINT					0					0					MTC				

## ニーモニック:

VIMTC rd, rs (sync(s) = 0)

VIMTC.sy rd, rs (sync(s) = 1)

## 機能:

VICTRL[rd] ← GPR[rs]

## 例外:

## 概要:

整数ベクトル制御レジスタライト命令 . rd で指定された制御レジスタに汎用レジスタの値を格納する . sync が 1 の場合 , 投機実行を抑制する .

<b>VIMFS</b>										Move from Vector Integer Scalar Register																													
整数スカラーレジスタリード															VECTOR																								
31	26 25					21 20					16 15					11 10					7 6 5					0													
011110					rs					00000					rd					0000					s					110010									
<b>VINT</b>										<b>0</b>										<b>0</b>										<b>MFS</b>									

ニーモニック:

VIMFS rd, rs (sync(s) = 0)  
 VIMFS.sy rd, rs (sync(s) = 1)

機能:

$GPR[rd] \leftarrow SGPR[rs]$

例外:

Vector Integer Exception:

概要:

整数スカラーレジスタリード命令。rs で指定された整数スカラーレジスタの値を汎用レジスタに格納する。sync が 1 の場合、投機実行を抑制する。

<b>VIMTS</b>							Move to Vector Integer Scalar Register					
整数スカラレジスタライト							VECTOR					
31	26	25	21	20	16	15	11	10	7	6	5	0
011110	rs		00000		rd		0000		s	110011		
VINT			0		0		MTS					

ニーモニック:

VIMTS rd, rs (sync(s) = 0)

VIMTS.sy rd, rs (sync(s) = 1)

機能:

SGPR[rd] ← GPR[rs]

例外:

Vector Integer Exception:

概要:

整数スカラレジスタライト命令。rd で指定された整数スカラレジスタに汎用レジスタの値を格納する。sync が 1 の場合、投機実行を抑制する。

<b>VIMFV</b>										Move from Vector Integer Vector Register																						
整数ベクトルレジスタリード										VECTOR																						
31						26 25						21 20						16 15						11 10				7 6 5				0
011110					rs					rt					rd					0000			s	110100								
<b>VINT</b>										<b>0</b>										<b>MFV</b>												

ニーモニック:

VIMFV rd, rs, rt (sync(s) = 0)  
 VIMFV.sy rd, rs, rt (sync(s) = 1)

機能:

$SGPR[rd] \leftarrow VGPR[rs][rt]$

例外:

Vector Integer Exception:

概要:

整数ベクトルレジスタリード命令 .rs で指定された整数ベクトルレジスタの rt 番目の要素の値を整数スカラレジスタに格納する .sync が 1 の場合 , 投機実行を抑制する .

<b>VIMTV</b>		Move to Vector Integer Vector Register					
整数ベクトルレジスタライト						VECTOR	
31	26 25	21 20	16 15	11 10	7 6 5	0	
011110	rs	rt	rd	0000	s	110101	
VINT				0		MTV	

ニーモニック:

VIMTV rd, rs, rt (sync(s) = 0)  
 VIMTV.sy rd, rs, rt (sync(s) = 1)

機能:

$VGPR[rd][rt] \leftarrow SGPR[rs]$

例外:

Vector Integer Exception:

概要:

整数ベクトルレジスタリード命令。rd で指定された整数ベクトルレジスタの rt 番目の要素に整数スカラレジスタの値を書き込む。sync が 1 の場合、投機実行を抑制する。

<b>VIMTM</b>										Move to Vector Integer Mask Register																			
整数ベクトルマスクレジスタライト										VECTOR																			
31		26		25		21		20		9		8		7		6		5		0									
011110		rs		000000000000		s1		s0		s		011110																	
VINT										0										MTM									

## ニーモニック:

VIMTM.lo rs	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VIMTM.hi rs	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VIMTM.lo.sy rs	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VIMTM.hi.sy rs	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

## 機能:

VICTRL[Mask Register] ← SGPR[rs]

## 例外:

Vector Integer Exception:

## 概要:

整数ベクトルマスクレジスタライト命令。rs で指定した整数スカラレジスタの値をマスクレジスタに格納する。s0 が 1 の場合、マスクレジスタの下位 32bit に値を格納し、s1 が 1 の場合、マスクレジスタの上位 32bit に値を格納する。sync が 1 の場合、投機実行を抑制する。

<b>VIRSV</b>										Vector Integer Register Reserve															
整数ベクトルレジスタ予約										VECTOR															
31	26	25	21	20	16	15	11	10	7	6	5	0													
011110			rs			00000			rd			0000			s	110110									
VINT			0			0			RSV																

ニーモニック:

VIRSV rd, rs (sync(s) = 0)  
 VIRSV.sy rd, rs (sync(s) = 1)

機能:

```
reserve_vector_register(GPR[rs])
if success_reserve_operation then
  GPR[rd] ← 1
else
  GPR[rd] ← 0
endif
```

例外:

None

概要:

整数ベクトルレジスタ予約命令。GPR[rs] に予約するレジスタの構成を指定する。予約に成功した場合は GPR[rd] に 1 が、失敗した場合は 0 が格納される。sync が 1 の場合、投機実行を抑制する。

<b>VIRLS</b>		Vector Integer Register Release				
整数ベクトルレジスタ開放						VECTOR
31	26 25	16 15	11 10	7 6 5	0	
011110	0000000000	rd	0000	s	110111	
<b>VINT</b>		<b>0</b>			<b>RLS</b>	

ニーモニック:

VIRLS rd (sync(s) = 0)  
 VIRLS.sy rd (sync(s) = 1)

機能:

```
release_vector_register()
if success_release_operation then
  GPR[rd] ← 1
else
  GPR[rd] ← 0
endif
```

例外:

None

概要:

整数ベクトルレジスタ開放命令。開放に成功した場合は GPR[rd] に 1 が、失敗した場合は 0 が格納される。sync が 1 の場合、投機実行を抑制する。

<b>VIDCI</b>		Vector Integer Define Compound Instruction					
整数ベクトル複合命令定義						VECTOR	
31	26 25	21 20	16 15	11 10	7 6 5	0	
011110	rs	00000	rd	0000	s	101110	
<b>VINT</b>		<b>0</b>		<b>0</b>		<b>DCI</b>	

ニーモニック:

VIDCI rd, rs (sync(s) = 0)  
 VIDCL.sy rd, rs (sync(s) = 1)

機能:

VICPD[rd] ← GPR[rs]

例外:

Vector Integer Exception:

概要:

整数ベクトル複合命令の定義を行う。GPR[rs] で定義した命令を rd で指定した複合命令バッファのアドレスに格納する。sync が 1 の場合、投機実行を抑制する。

<b>VIECI</b>		Vector Integer Execute Compound Instruction					
整数ベクトル複合命令実行						VECTOR	
31	26 25	21 20	16 15	11 10	6 5	0	
011110	rs	rt	rd	no	101111		
VINT					ECI		

ニーモニック:

VIECI rd, rs, rt, no

機能:

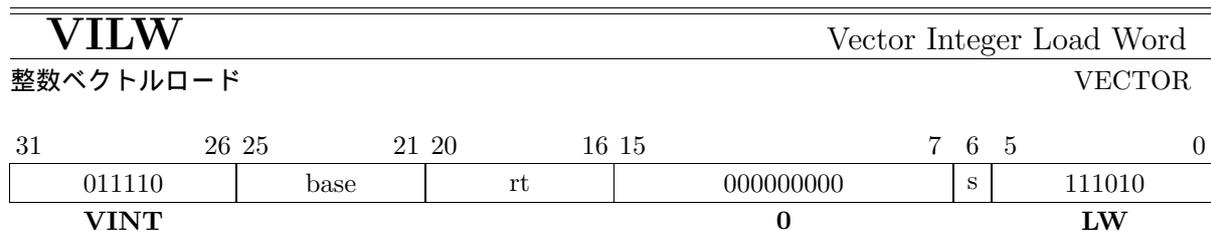
$VGPR[rd] \leftarrow VGPR[rs] \text{ op } VGPR[rt]$

例外:

Vector Integer Exception:

概要:

整数ベクトル複合命令の実行を行う。no で指定した複合命令バッファのアドレスから命令を実行する。

**ニーモニック:**

VILW rt, base (sync(s) = 0)

VILW.sy rt, base (sync(s) = 1)

**機能:**

$VGPR[rt] \leftarrow MEM.WORD[GPR[base]]$

**例外:**

Vector Integer Exception :

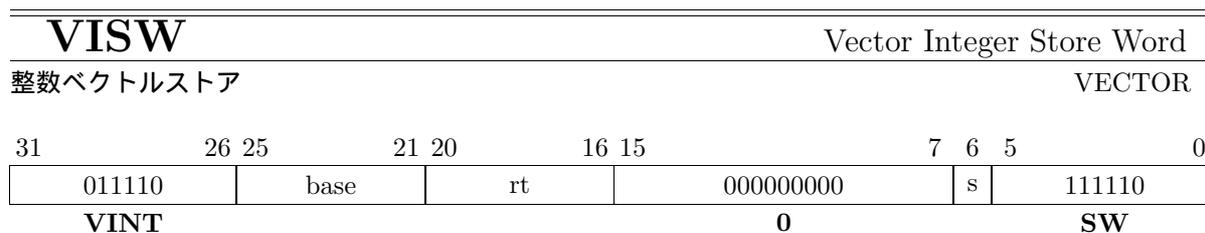
D-TLB No Entry Matched :

D-TLB Protection Error :

Data Address Miss Align (Load) :

**概要:**

メモリから整数ベクトルレジスタにロードする。sync が 1 の場合、投機実行を抑制する。



ニーモニック:

VISW rt, base (sync(s) = 0)

VISW.sy rt, base (sync(s) = 1)

機能:

MEM.WORD[GPR[base]] ← VGPR[rt]

例外:

Vector Integer Exception:

D-TLB No Entry Matched:

D-TLB Protection Error:

Data Address Miss Align (Store):

概要:

整数ベクトルレジスタをメモリにストアする。sync が 1 の場合、投機実行を抑制する。

<b>VADD.QB</b>											Vector Add Quad Byte			
ベクトル加算										VECTOR				
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
111110		rs		rt		rd		0	s2	0	s0	s	100000	
<b>VINT.QB</b>										<b>0</b>	<b>0</b>	<b>ADD</b>		

## ニーモニック:

VADD.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VADD.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VADD.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VADD.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VADD.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VADD.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VADD.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VADD.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] + \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception:

## 概要:

8 bit × 4 ベクトル加算 . s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合、投機実行を抑制する .

<b>VSUB.QB</b>										Vector Subtract Quad Byte					
ベクトル減算										VECTOR					
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0	
111110		rs		rt		rd		0	s2	s1	s0	s	100010		
<b>VINT.QB</b>										<b>0</b>				<b>SUB</b>	

## ニーモニック:

VSUB.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.QB.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 0)
VSUB.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.QB.sv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 0)
VSUB.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.QB.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 1)
VSUB.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 1)
VSUB.QB.sv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] - \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

8 bit × 4 ベクトル減算 . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s1 が 1 の場合は VGPR[rs] の代わりに , スカラレジスタ (SGPR[rs]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VMULT.QB</b>											Vector Multiply Quad Byte													
ベクトル乗算											VECTOR													
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
111110				rs				rt				rd				0	s2	0	s0	s	011000			
<b>VINT.QB</b>											<b>0</b>				<b>0</b>				<b>MULT</b>					

#### ニーモニック:

VMULT.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULT.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULT.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULT.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULT.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULT.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULT.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULT.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

#### 機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

#### 例外:

Vector Integer Exception :

#### 概要:

8 bit × 4 符号付きベクトル乗算 . s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合、投機実行を抑制する .

<b>VMULTU.QB</b>										Vector Multiply Unsigned				
ベクトル乗算										VECTOR				
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
111110	rs		rt		rd		0	s2	0	s0	s	011001		
<b>VINT.QB</b>										<b>MULTU</b>				

ニーモニック:

VMULTU.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTU.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTU.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTU.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTU.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTU.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTU.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTU.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

例外:

Vector Integer Exception:

概要:

8 bit × 4 符号無しベクトル乗算。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VMULTH.QB</b>										Vector Multiply Quad Byte on High Bit															
ベクトル乗算										VECTOR															
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0				
111110				rs				rt				rd				0	s2	0	s0	s	010000				
<b>VINT.QB</b>														<b>0</b>	<b>0</b>	<b>MULTH</b>									

#### ニーモニック:

VMULTH.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTH.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTH.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTH.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTH.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTH.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTH.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTH.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

#### 機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

#### 例外:

Vector Integer Exception :

#### 概要:

8 bit × 2 符号付きベクトル乗算 . 8 bit 毎に演算結果の上位 bit(16-8bit) が VGPR[rd] に格納される . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VMULTUH.QB</b>										Vector Multiply Unsigned Quad Byte on High Bit														
ベクトル乗算										VECTOR														
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0										
111110			rs		rt		rd			0	s2	0	s0	s	010001									
<b>VINT.QB</b>										<b>0</b>		<b>0</b>		<b>MULTUH</b>										

ニーモニック:

VMULTUH.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTUH.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

例外:

Vector Integer Exception:

概要:

8 bit × 4 符号無しベクトル乗算 . 8 bit 毎に演算結果の上位 bit(16-8bit) が VGPR[rd] に格納される . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VMADD.QB</b>										Vector Multiply and Add Quad Byte														
ベクトル積和演算										VECTOR														
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
111110				rs				rt				rd				0	s2	0	s0	s	100001			
<b>VINT.QB</b>														<b>0</b>	<b>0</b>					<b>MADD</b>				

## ニーモニック:

VMADD.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMADD.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMADD.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMADD.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMADD.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMADD.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMADD.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMADD.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}] + \text{VGPR}[\text{rd}]$$

## 例外:

Vector Integer Exception:

## 概要:

8 bit × 4 ベクトル積和演算 .s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う .s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う .sync が 1 の場合、投機実行を抑制する .

<b>VMSUB.QB</b>										Vector Multiply and Subtract Quad Byte																
ベクトル積差演算										VECTOR																
31					26 25					21 20					16 15					11 10	9	8	7	6	5	0
111110				rs				rt				rd				0	s2	0	s0	s	100011					
<b>VINT.QB</b>														<b>0</b>	<b>0</b>					<b>MSUB</b>						

ニーモニック:

VMSUB.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMSUB.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMSUB.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMSUB.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMSUB.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMSUB.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMSUB.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMSUB.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}] - \text{VGPR}[\text{rd}]$$

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル積差演算 . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VACC.QB</b>				Vector Accumulate Quad Byte							
ベクトル累算				VECTOR							
31	26	25	21	20	16	15	11	10	6	5	0
111110		rs	rt		rd		00000		001010		
<b>VINT.QB</b>				<b>0</b>				<b>ACC</b>			

ニーモニック:

VACC.QB rd, rs (sync(s) = 0)

VACC.QB.sy rd, rs (sync(s) = 1)

機能:

$SGPR[rd] \leftarrow \sum VGPR[rs]$

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル累算。ベクトルの要素を全て加算する。sync が 1 の場合、投機実行を抑制する。

<b>VMAC.QB</b>										Vector Multiply and Accumulate Quad Byte																			
ベクトル積和演算										VECTOR																			
31					26 25					21 20					16 15					11 10	9	8	7	6	5	0			
111110					rs					rt					rd					0	s2	0	s0	s	001011				
<b>VINT.QB</b>																				<b>0</b>		<b>0</b>		<b>MAC</b>					

## ニーモニック:

VMAC.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMAC.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMAC.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMAC.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMAC.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMAC.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMAC.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMAC.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{SGPR}[\text{rd}] \leftarrow \sum \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

8 bit × 4 ベクトル積和演算 . 2 つのベクトル要素を乗算し , それを全て加算する . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VAND.QB</b>											Vector And Quad Byte				
ベクトル論理積											VECTOR				
31	26 25		21 20		16 15		11 10		9	8	7	6	5	0	
111110			rs		rt		rd		0	s2	0	s0	s	100100	
<b>VINT.QB</b>									<b>0</b>		<b>0</b>		<b>AND</b>		

## ニーモニック:

VAND.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VAND.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VAND.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VAND.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VAND.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VAND.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VAND.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VAND.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$VGPR[rd] \leftarrow VGPR[rs] \mathbf{and} VGPR[rt]$

## 例外:

Vector Integer Exception :

## 概要:

8 bit × 4 ベクトル論理積 . s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合、投機実行を抑制する .

<b>VOR.QB</b>											Vector Or				
ベクトル論理和											VECTOR				
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0	
111110			rs		rt		rd		0	s2	0	s0	s	100101	
<b>VINT.QB</b>											<b>OR</b>				
									<b>0</b>	<b>0</b>					

ニーモニック:

VOR.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VOR.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VOR.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VOR.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VOR.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VOR.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VOR.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VOR.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \text{ or } VGPR[rt]$

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル論理和 .s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う .s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う .sync が 1 の場合、投機実行を抑制する .

<b>VXOR.QB</b>										Vector Exclusive Or Quad Byte														
ベクトル排他的論理和										VECTOR														
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
111110				rs				rt				rd				0	s2	0	s0	s	100110			
<b>VINT.QB</b>														<b>0</b>	<b>0</b>					<b>XOR</b>				

## ニーモニック:

VXOR.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VXOR.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VXOR.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VXOR.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VXOR.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VXOR.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VXOR.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VXOR.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$VGPR[rd] \leftarrow VGPR[rs] \mathbf{xor} VGPR[rt]$

## 例外:

Vector Integer Exception :

## 概要:

8 bit × 4 ベクトル排他的論理和 . s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合、投機実行を抑制する .

<b>VNOR.QB</b>										Vector Not Or Paried HalfWord														
ベクトル否定論理和										VECTOR														
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
111110				rs				rt				rd				0	s2	0	s0	s	100111			
<b>VINT.QB</b>														<b>0</b>	<b>0</b>					<b>NOR</b>				

## ニーモニック:

VNOR.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VNOR.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VNOR.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VNOR.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VNOR.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VNOR.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VNOR.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VNOR.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$VGPR[rd] \leftarrow VGPR[rs] \text{ nor } VGPR[rt]$

## 例外:

Vector Integer Exception :

## 概要:

8 bit × 4 ベクトル否定論理和 . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .



<b>VSRLV.QB</b>										Vector Shift Right Logical Variable Quad Byte														
ベクトル右論理シフト										VECTOR														
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
111110				rs				rt				rd				0	s2	0	s0	s	000110			
<b>VINT.QB</b>														<b>0</b>	<b>0</b>					<b>SRLV</b>				

## ニーモニック:

VSRLV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRLV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRLV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRLV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRLV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRLV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRLV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRLV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rt}] \gg \text{VGPR}[\text{rs}]$$

## 例外:

Vector Integer Exception :

## 概要:

8 bit × 4 ベクトル右論理シフト。s0 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う。s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VSRV.QB</b>											Vector Shift Right Arithmetic Variable Quad Byte											
ベクトル右算術シフト											VECTOR											
31	26 25				21 20				16 15				11 10 9 8 7 6 5				0					
111110	rs				rt				rd				0	s2	0	s0	s	000111				
<b>VINT.QB</b>											<b>0</b>				<b>0</b>				<b>SRAV</b>			

## ニーモニック:

VSRV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rt}] \gg \text{VGPR}[\text{rs}]$$

## 例外:

Vector Integer Exception :

## 概要:

ベクトル右算術シフト。s0 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う。s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VRTLV.QB</b>										Vector Rotate Left Variable Quad Byte														
ベクトル左ローテーション										VECTOR														
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
111110				rs				rt				rd				0	s2	0	s0	s	000000			
<b>VINT.QB</b>														<b>0</b>	<b>0</b>					<b>SRTL</b>				

## ニーモニック:

VRTLV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTLV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTLV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTLV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTLV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTLV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTLV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTLV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rt}] \lll \text{VGPR}[\text{rs}]$$

## 例外:

Vector Integer Exception :

## 概要:

ベクトル左ローテーション .s0 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う .s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う .sync が 1 の場合、投機実行を抑制する .

<b>VRTRV.QB</b>											Vector Rotate Right Variable Quad Byte																					
ベクトル右ローテーション											VECTOR																					
31					26 25					21 20					16 15					11 10	9	8	7	6	5							0
111110					rs					rt					rd					0	s2	0	s0	s	000010							
<b>VINT.QB</b>											<b>SRTRV</b>																					

ニーモニック:

VRTRV.QB.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTRV.QB.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTRV.QB.vv.lo8 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTRV.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTRV.QB.vs.lo8 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTRV.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTRV.QB.vv.lo8.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTRV.QB.vs.lo8.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rt] \ggg VGPR[rs]$

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル右ローテーション。s0 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う。s2 が 1 の場合は VGPR[rt] の下位 8 bit を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VCMP.QB</b>										Vector Compare			
ベクトル比較										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
111110		rs		rt		rd		cond	s0	s	101000		
<b>VINT.QB</b>										<b>CMP</b>			

ニーモニック:

VCMP.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル比較命令。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=)、gt(>)、lt(<)、ne(≠)、le(≤)、ge(≥) のどれかを指定する。

<b>VSCMP.QB</b>										Vector Compare			
ベクトル比較										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
111110	rs		rt		rd		cond	s0	s	001100			
<b>VINT.QB</b>										<b>SCMP</b>			

ニーモニック:

VSCMP.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMP.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMP.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMP.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

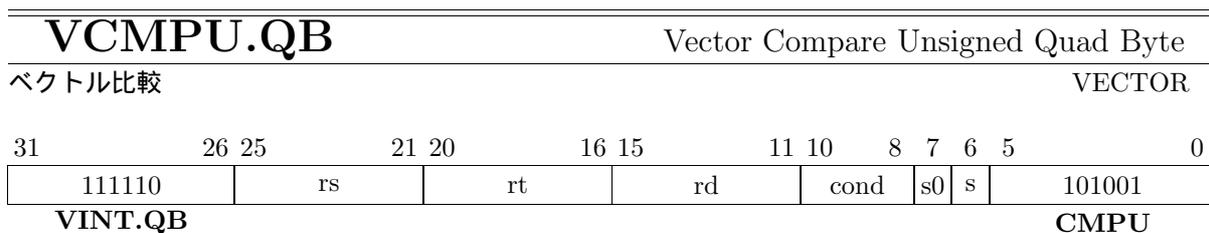
```

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル比較命令。VGPR[rt] の下位 8 bit を用いて演算を行う。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VCMPU.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPU.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPU.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPU.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

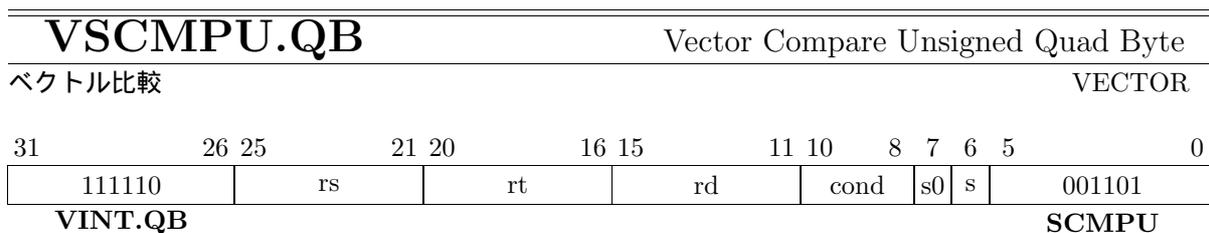
```

例外:

Vector Integer Exception :

概要:

8 bit × 4 符号無しベクトル比較命令。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VSCMPU.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPU.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPU.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPU.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

8 bit × 4 符号無しベクトル比較命令。VGPR[rt] の下位 8 bit を用いて演算を行う。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VCMPTS.QB</b>										Vector Compare to Scalar Register Quad Byte																				
ベクトル比較										VECTOR																				
31						26 25						21 20						16 15						11 10			8 7 6 5			0
111110					rs					rt					rd					cond		s0	s	101010						
<b>VINT.QB</b>										<b>CMPTS</b>																				

ニーモニック:

VCMPTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

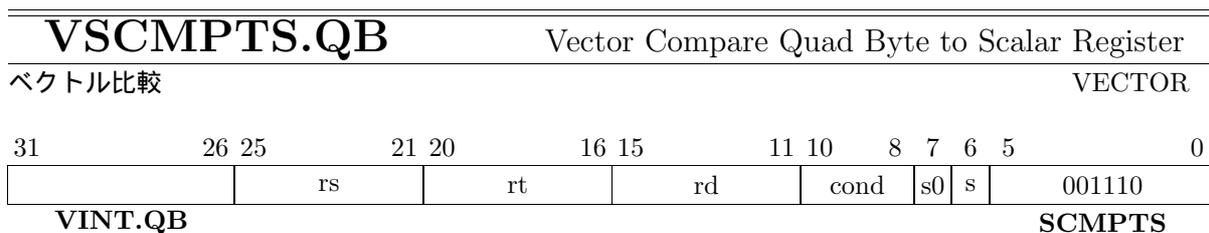
```

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VSCMPTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

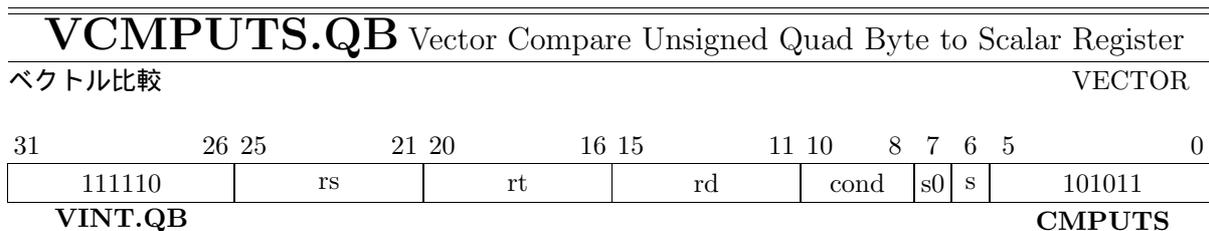
```

例外:

Vector Integer Exception:

概要:

8 bit × 4 ベクトル比較命令。VGPR[rt] の下位 8 bit を用いて演算を行う。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VCMPUTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPUTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPUTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPUTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

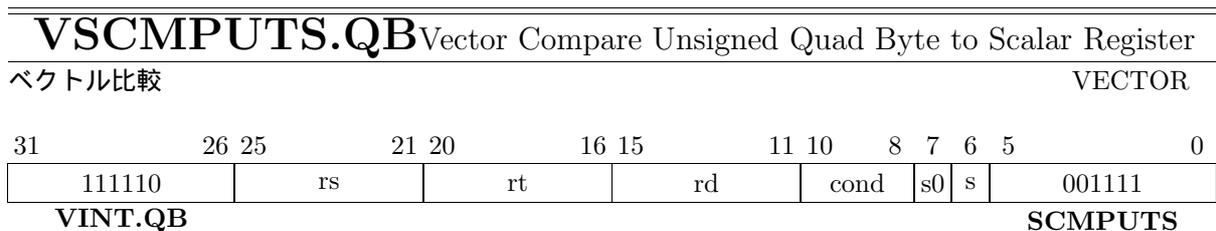
```

例外:

Vector Integer Exception:

概要:

8 bit × 4 符号無しベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VSCMPUTS.cond.QB.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPUTS.cond.QB.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPUTS.cond.QB.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPUTS.cond.QB.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

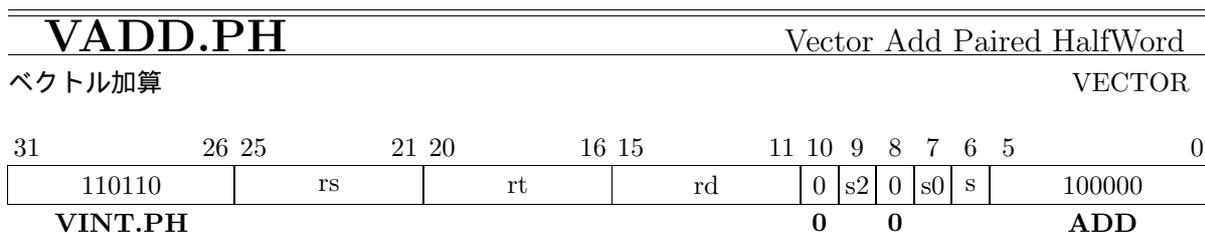
```

例外:

Vector Integer Exception:

概要:

8 bit × 4 符号無しベクトル比較命令。VGPR[rt] の下位 8 bit を用いて演算を行う。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VADD.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VADD.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VADD.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VADD.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VADD.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VADD.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VADD.PH.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VADD.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] + VGPR[rt]$

例外:

Vector Integer Exception:

概要:

16 bit × 2 ベクトル加算 . s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合、投機実行を抑制する .

<b>VSUB.PH</b>										Vector Subtract Paired HalfWord																			
ベクトル減算										VECTOR																			
31	26 25				21 20				16 15				11 10 9 8 7 6 5				0												
110110				rs				rt				rd				0 s2 s1 s0 s				100010									
<b>VINT.PH</b>										<b>0</b>										<b>SUB</b>									

## ニーモニック:

VSUB.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PH.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 0)
VSUB.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PH.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PH.sv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 0)
VSUB.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PH.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 0, sync(s) = 1)
VSUB.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) =0, scalar2(s2) = 1, sync(s) = 1)
VSUB.PH.sv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) =1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] - \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 ベクトル減算 . s0 が 1 の場合は VGPR[rt] の代わりに, スカラレジスタ (SGPR[rt]) を用いて演算を行う . s1 が 1 の場合は VGPR[rs] の代わりに, スカラレジスタ (SGPR[rs]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合, 投機実行を抑制する .

<b>VMULT.PH</b>										Vector Multiply Paired HalfWord										
ベクトル乗算										VECTOR										
31	26 25		21 20		16 15		11 10		9	8	7	6	5							0
110110			rs		rt		rd		0	s2	0	s0	s	011000						
<b>VINT.PH</b>										<b>0</b>		<b>0</b>		<b>MULT</b>						

## ニーモニック:

VMULT.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULT.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULT.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULT.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULT.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULT.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULT.PH.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULT.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 符号付きベクトル乗算 . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

VMULTU.PH										Vector Multiply Unsigned				
ベクトル乗算										VECTOR				
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
110110		rs		rt		rd		0	s2	0	s0	s	011001	
VINT.PH										MULTU				

## ニーモニック:

VMULTU.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTU.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTU.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTU.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTU.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTU.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTU.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTU.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 符号無しベクトル乗算 . s0 が 1 の場合は  $VGPR[rt]$  の代わりに, スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う . s2 が 1 の場合は  $VGPR[rt]$  の下位 16 bit を用いて演算を行う . sync が 1 の場合, 投機実行を抑制する .

<b>VMULTH.PH</b>										Vector Multiply Paired HalfWord on High Bit															
ベクトル乗算										VECTOR															
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0				
110110				rs				rt				rd				0	s2	0	s0	s	010000				
<b>VINT.PH</b>														<b>0</b>	<b>0</b>	<b>MULRH</b>									

## ニーモニック:

VMULTH.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTH.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTH.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTH.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTH.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTH.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTH.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTH.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

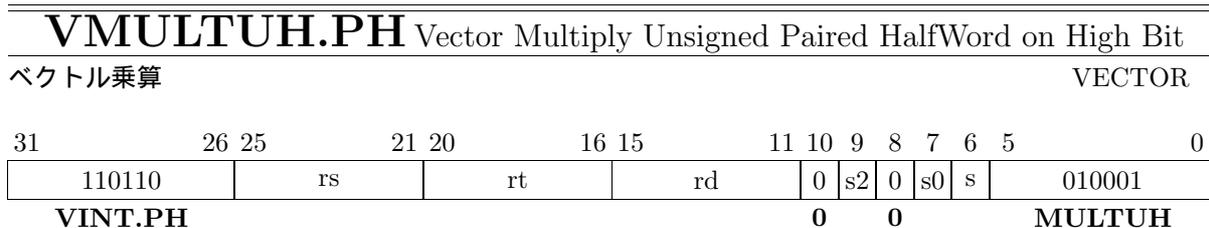
$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 符号付きベクトル乗算 . 16 bit 毎に演算結果の上位 bit(32-16bit) が VGPR[rd] に格納される . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .



ニーモニック:

VMULTUH.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMULTUH.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMULTUH.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMULTUH.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMULTUH.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \times VGPR[rt]$

例外:

Vector Integer Exception:

概要:

16 bit × 2 符号無しベクトル乗算 . 16 bit 毎に演算結果の上位 bit(32-16bit) が VGPR[rd] に格納される . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VMADD.PH</b>											Vector Multiply and Add Paired HalfWord													
ベクトル積和演算											VECTOR													
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
110110				rs				rt				rd				0	s2	0	s0	s	100001			
<b>VINT.PH</b>											<b>0</b>				<b>0</b>				<b>MADD</b>					

## ニーモニック:

VMADD.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMADD.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMADD.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMADD.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMADD.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMADD.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMADD.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMADD.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}] + \text{VGPR}[\text{rd}]$$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 ベクトル積和演算 . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VMSUB.PH</b>											Vector Multiply and Subtract Paired HalfWord										
ベクトル積差演算											VECTOR										
31	26 25		21 20		16 15		11 10		9	8	7	6 5		0							
110110	rs		rt		rd		0	s2	0	s0	s	100011									
<b>VINT.PH</b>											<b>0</b>	<b>0</b>	<b>MSUB</b>								

## ニーモニック:

VMSUB.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMSUB.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}] - \text{VGPR}[\text{rd}]$$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 ベクトル積差演算 . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VACC.PH</b>						Vector Accumulate Paired HalfWord					
ベクトル累算						VECTOR					
31	26	25	21	20	16	15	11	10	6	5	0
110110	rs		rt		rd		00000		001010		
<b>VINT.PH</b>						<b>0</b>		<b>ACC</b>			

ニーモニック:

VACC.PH rd, rs (sync(s) = 0)

VACC.PH.sy rd, rs (sync(s) = 1)

機能:

$SGPR[rd] \leftarrow \sum VGPR[rs]$

例外:

Vector Integer Exception:

概要:

16 bit × 2 ベクトル累算。ベクトルの要素を全て加算する。sync が 1 の場合、投機実行を抑制する。

<b>VMAC.PH</b>										Vector Multiply and Accumulate Paired HalfWord																			
ベクトル積和演算										VECTOR																			
31					26 25					21 20					16 15					11 10	9	8	7	6	5	0			
110110					rs					rt					rd					0	s2	0	s0	s	001011				
<b>VINT.PH</b>																				<b>0</b>	<b>0</b>	<b>MAC</b>							

### ニーモニック:

VMAC.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMAC.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMAC.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMAC.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMAC.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMAC.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMAC.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMAC.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

### 機能:

$$\text{SGPR}[\text{rd}] \leftarrow \sum \text{VGPR}[\text{rs}] \times \text{VGPR}[\text{rt}]$$

### 例外:

Vector Integer Exception :

### 概要:

16 bit × 2 ベクトル積和演算 . 2 つのベクトル要素を乗算し, それを全て加算する . s0 が 1 の場合は VGPR[rt] の代わりに, スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合, 投機実行を抑制する .

<b>VAND.PH</b>										Vector And Paired HalfWord																									
ベクトル論理積										VECTOR																									
31						26 25						21 20						16 15						11 10	9	8	7	6	5						0
110110					rs					rt					rd					0	s2	0	s0	s	100100										
<b>VINT.PH</b>																				<b>0</b>	<b>0</b>	<b>AND</b>													

## ニーモニック:

VAND.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VAND.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VAND.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VAND.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VAND.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VAND.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VAND.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VAND.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$VGPR[rd] \leftarrow VGPR[rs] \text{ and } VGPR[rt]$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 ベクトル論理積 . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合 , 投機実行を抑制する .

<b>VOR.PH</b>											Vector Or			
ベクトル論理和											VECTOR			
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
110110	rs		rt		rd		0	s2	0	s0	s	100101		
<b>VINT.PH</b>							<b>0</b>	<b>0</b>	<b>OR</b>					

ニーモニック:

VOR.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VOR.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VOR.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VOR.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VOR.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VOR.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VOR.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VOR.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \text{ or } VGPR[rt]$

例外:

Vector Integer Exception:

概要:

16 bit × 2 ベクトル論理和 .s0 が 1 の場合は  $VGPR[rt]$  の代わりに、スカラレジスタ ( $SGPR[rt]$ ) を用いて演算を行う .s2 が 1 の場合は  $VGPR[rt]$  の下位 16 bit を用いて演算を行う .sync が 1 の場合、投機実行を抑制する。

<b>VXOR.PH</b>										Vector Exclusive Or Paired HalfWord														
ベクトル排他的論理和										VECTOR														
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0			
110110				rs				rt				rd				0	s2	0	s0	s	100110			
<b>VINT.PH</b>														<b>0</b>	<b>0</b>					<b>XOR</b>				

ニーモニック:

VXOR.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VXOR.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VXOR.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VXOR.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VXOR.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VXOR.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VXOR.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VXOR.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rs] \mathbf{xor} VGPR[rt]$

例外:

Vector Integer Exception :

概要:

16 bit × 2 ベクトル排他的論理和 . s0 が 1 の場合は VGPR[rt] の代わりに、スカラーレジスタ (SGPR[rt]) を用いて演算を行う . s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う . sync が 1 の場合、投機実行を抑制する .



<b>VSLLV.PH</b>														
Vector Shift Left Logical Variable Paired HalfWord														
ベクトル左論理シフト														
VECTOR														
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
110110	rs		rt		rd		0	s2	0	s0	s	000100		
<b>VINT.PH</b>							<b>0</b>	<b>0</b>						<b>SLLV</b>

ニーモニック:

VSLLV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSLLV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSLLV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSLLV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSLLV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSLLV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSLLV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSLLV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rt}] \ll \text{VGPR}[\text{rs}]$$

例外:

Vector Integer Exception :

概要:

16 bit × 2 ベクトル左論理シフト。s0 が 1 の場合は VGPR[rs] の代わりに、スカラーレジスタ (SGPR[rs]) を用いて演算を行う。s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VSRLV.PH</b>											Vector Shift Right Logical Variable Paired HalfWord																		
ベクトル右論理シフト											VECTOR																		
31	26 25					21 20					16 15					11 10 9 8 7 6 5					0								
110110					rs					rt					rd					0	s2	0	s0	s	000110				
<b>VINT.PH</b>											<b>0</b>					<b>0</b>					<b>SRLV</b>								

## ニーモニック:

VSRLV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRLV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRLV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRLV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRLV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRLV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRLV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRLV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rt}] \gg \text{VGPR}[\text{rs}]$$

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 ベクトル右論理シフト。s0 が 1 の場合は VGPR[rs] の代わりに、スカラーレジスタ (SGPR[rs]) を用いて演算を行う。s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VSRV.PH</b>														
Vector Shift Right Arithmetic Variable Paired HalfWord														
ベクトル右算術シフト														
VECTOR														
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
110110	rs		rt		rd		0	s2	0	s0	s	000111		
<b>VINT.PH</b>							<b>0</b>	<b>0</b>						<b>SRAV</b>

## ニーモニック:

VSRV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VSRV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VSRV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VSRV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VSRV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VSRV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VSRV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VSRV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VGPR}[\text{rd}] \leftarrow \text{VGPR}[\text{rt}] \gg \text{VGPR}[\text{rs}]$$

## 例外:

Vector Integer Exception :

## 概要:

ベクトル右算術シフト。s0 が 1 の場合は VGPR[rs] の代わりに、スカラレジスタ (SGPR[rs]) を用いて演算を行う。s2 が 1 の場合は VGPR[rt] の下位 16 bit を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VRTLV.PH</b>										Vector Rotate Left Variable Paired HalfWord																			
ベクトル左ローテーション										VECTOR																			
31		26		25		21		20		16		15		11		10		9		8		7		6		5		0	
110110		rs		rt		rd		0		s2		0		s0		s		000000											
<b>VINT.PH</b>										<b>0</b>		<b>0</b>		<b>SRTL</b>															

ニーモニック:

VRTLV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTLV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTLV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTLV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTLV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTLV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTLV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTLV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rt] \lll VGPR[rs]$

例外:

Vector Integer Exception:

概要:

ベクトル左ローテーション .s0 が 1 の場合は  $VGPR[rs]$  の代わりに、スカラーレジスタ ( $SGPR[rs]$ ) を用いて演算を行う .s2 が 1 の場合は  $VGPR[rt]$  の下位 16 bit を用いて演算を行う .sync が 1 の場合、投機実行を抑制する。

<b>VRTRV.PH</b>										Vector Rotate Right Variable Paired HalfWord																			
ベクトル右ローテーション										VECTOR																			
31		26		25		21		20		16		15		11		10		9		8		7		6		5		0	
110110		rs		rt		rd		0		s2		0		s0		s		000010											
<b>VINT.PH</b>										<b>0</b>		<b>0</b>		<b>SRTRV</b>															

ニーモニック:

VRTRV.PH.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VRTRV.PH.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VRTRV.PH.vv.lo16 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VRTRV.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VRTRV.PH.vs.lo16 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VRTRV.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VRTRV.PH.vv.lo16.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VRTRV.PH.vs.lo16.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

機能:

$VGPR[rd] \leftarrow VGPR[rt] \ggg VGPR[rs]$

例外:

Vector Integer Exception:

概要:

ベクトル右ローテーション .s0 が 1 の場合は  $VGPR[rs]$  の代わりに、スカラーレジスタ ( $SGPR[rs]$ ) を用いて演算を行う .s2 が 1 の場合は  $VGPR[rt]$  の下位 16 bit を用いて演算を行う .sync が 1 の場合、投機実行を抑制する .

VCMP.PH										Vector Compare			
ベクトル比較										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
110110		rs		rt		rd		cond		s0	s	101000	
VINT.PH										CMP			

## ニーモニック:

VCMP.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

```

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 ベクトル比較命令。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VSCMP.PH</b>										Vector Compare			
ベクトル比較										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
rs			rt			rd		cond	s0	s	001100		
<b>VINT.PH</b>										<b>SCMP</b>			

ニーモニック:

VSCMP.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMP.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMP.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMP.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

16 bit × 2 ベクトル比較命令。VGPR[rt] の下位 16 bit を用いて演算を行う。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 には、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

VCMPU.PH										Vector Compare Unsigned Paired HalfWord										
ベクトル比較										VECTOR										
31	26	25	21	20	16	15	11	10	8	7	6	5								0
110110				rs		rt		rd		cond		s0	s	101001						
VINT.PH										CMPU										

## ニーモニック:

VCMPU.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPU.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPU.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPU.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

```

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 符号無しベクトル比較命令 . 条件 (cond) により VGPR[rd] に 1 または 0 が入る . s0 が 1 の場合は VGPR[rt] の代わりに , スカラレジスタ (SGPR[rt]) を用いて演算を行う . sync が 1 の場合は , 投機実行を抑制する . 比較条件 (cond) には , eq(=) , gt(>) , lt(<) , ne(≠) , le(≤) , ge(≥) のどれかを指定する .

<b>VSCMPU.PH</b>										Vector Compare Unsigned Paired HalfWord																	
ベクトル比較										VECTOR																	
31		26		25		21		20		16		15		11		10		8		7		6		5		0	
110110		rs		rt		rd		cond		s0		s		001101													
<b>VINT.PH</b>										<b>SCMPU</b>																	

## ニーモニック:

VSCMPU.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPU.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPU.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPU.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

```

if VGPR[rs] cond VGPR[rt] then
    VGPR[rd] ← 1
else
    VGPR[rd] ← 0
endif

```

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 符号無しベクトル比較命令。VGPR[rt] の下位 16 bit を用いて演算を行う。条件 (cond) により VGPR[rd] に 1 または 0 が入る。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VCMPTS.PH</b>										Vector Compare to Scalar Register Paired HalfWord										
ベクトル比較										VECTOR										
31	26	25	21	20	16	15	11	10	8	7	6	5	0							
110110				rs		rt		rd		cond		s0	s	101010						
<b>VINT.PH</b>										<b>CMPTS</b>										

## ニーモニック:

VCMPTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

```

## 例外:

Vector Integer Exception :

## 概要:

16 bit × 2 ベクトル比較命令．結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される．s0 が 1 の場合は VGPR[rt] の代わりに，スカラレジスタ (SGPR[rt]) を用いて演算を行う．sync が 1 の場合は，投機実行を抑制する．比較条件 (cond) には，eq(=)，gt(>)，lt(<)，ne(≠)，le(≤)，ge(≥) のどれかを指定する．

<b>VSCMPTS.PH</b> Vector Compare Paired HalfWord to Scalar Register													
ベクトル比較													
VECTOR													
31	26	25	21	20	16	15	11	10	8	7	6	5	0
110110		rs		rt		rd		cond	s0	s	001110		
<b>VINT.PH</b>										<b>SCMPTS</b>			

ニーモニック:

VSCMPTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

16 bit × 2 ベクトル比較命令。VGPR[rt] の下位 16 bit を用いて演算を行う。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

---



---

**VCMPUTS.PH** Vector Compare Unsigned Paired HalfWord to Scalar Register

ベクトル比較

VECTOR

31	26 25	21 20	16 15	11 10	8 7	6 5	0
110110	rs	rt	rd	cond	s0	s	101011
<b>VINT.PH</b>				<b>CMPUTS</b>			

ニーモニック:

VCMPUTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPUTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPUTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPUTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

16 bit × 2 符号無しベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

---



---

## VSCMPUTS.PH Vector Compare Unsigned Paired HalfWord to Scalar Register

ベクトル比較 VECTOR

---

31	26	25	21	20	16	15	11	10	8	7	6	5	0
110110	rs		rt		rd		cond	s0	s	001111			
VINT.PH										SCMPUTS			

### ニーモニック:

VSCMPUTS.cond.PH.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPUTS.cond.PH.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPUTS.cond.PH.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPUTS.cond.PH.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

### 機能:

```

if VGPR[rs] cond VGPR[rt] then
    SGPR[rd] ← 1
else
    SGPR[rd] ← 0
endif

```

### 例外:

Vector Integer Exception :

### 概要:

16 bit × 2 符号無しベクトル比較命令。VGPR[rt] の下位 16 bit を用いて演算を行う。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VGPR[rt] の代わりに、スカラレジスタ (SGPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

## 3.3.9 浮動小数点ベクトル命令

<b>VADD.S</b>										Vector Add Single			
ベクトル加算										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011111	rs		rt		rd		000	s0	s	000000			
<b>VFP</b>							<b>0</b>		<b>ADD.S</b>				

ニーモニック:

VADD.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VADD.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VADD.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VADD.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VFPR[rd] \leftarrow VFPR[rs] + VFPR[rt]$

例外:

Vector Floating Point Exception:

概要:

単精度ベクトル加算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VADD.D</b>										Vector Add Double			
ベクトル加算										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011111	rs		rt		rd		000	s0	s	001000			
<b>VFP</b>							<b>0</b>		<b>ADD.D</b>				

ニーモニック:

VADD.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VADD.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VADD.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VADD.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] + \text{VFPR}[\text{rt}]$

例外:

Vector Floating Point Exception:

概要:

倍精度ベクトル加算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VSUB.S</b>											Vector Subtract Single				
ベクトル減算											VECTOR				
31	26 25		21 20		16 15		11 10 9		8	7	6	5	0		
011111	rs		rt		rd		00	s1	s0	s	000001				
<b>VFP</b>							<b>0</b>			<b>SUB.S</b>					

## ニーモニック:

VSUB.S.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VSUB.S.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VSUB.S.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VSUB.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VSUB.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VSUB.S.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

## 機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] - \text{VFPR}[\text{rt}]$$

## 例外:

Vector Floating Point Exception :

## 概要:

単精度ベクトル減算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。s1 が 1 の場合は VFPR[rs] の代わりに、スカラレジスタ (SFPR[rs]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VSUB.D</b>											Vector Subtract Double					
ベクトル減算											VECTOR					
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0		
011111			rs			rt			rd	00	s1	s0	s	001001		
<b>VFP</b>											<b>0</b>				<b>SUB.D</b>	

ニーモニック:

VSUB.D.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VSUB.D.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VSUB.D.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VSUB.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VSUB.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VSUB.D.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] - \text{VFPR}[\text{rt}]$$

例外:

Vector Floating Point Exception :

概要:

倍精度ベクトル減算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。s1 が 1 の場合は VFPR[rs] の代わりに、スカラレジスタ (SFPR[rs]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VMUL.S</b>										Vector Multiply Single			
ベクトル乗算										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011111	rs		rt		rd		000		s0	s	000010		
<b>VFP</b>							<b>0</b>		<b>MUL.S</b>				

ニーモニック:

VMUL.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMUL.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMUL.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMUL.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \times \text{VFPR}[\text{rt}]$

例外:

Vector Floating Point Exception:

概要:

単精度ベクトル乗算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VMUL.D</b>										Vector Multiply Double			
ベクトル乗算										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011111	rs		rt		rd		000	s0	s	001010			
<b>VFP</b>							<b>0</b>			<b>MUL.D</b>			

## ニーモニック:

VMUL.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMUL.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMUL.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMUL.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \times \text{VFPR}[\text{rt}]$

## 例外:

Vector Floating Point Exception :

## 概要:

倍精度ベクトル乗算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合、投機実行を抑制する。

<b>VDIV.S</b>											Vector Divide Single			
ベクトル除算											VECTOR			
31	26	25	21	20	16	15	11	10	9	8	7	6	5	0
011111	rs		rt		rd		00	s1	s0	s	000011			
<b>VFP</b>							<b>0</b>			<b>DIV.S</b>				

ニーモニック:

VDIV.S.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIV.S.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIV.S.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIV.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIV.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIV.S.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能:

$VFPR[rd] \leftarrow VFPR[rs] \div VFPR[rt]$

例外:

Vector Floating Point Exception:

概要:

単精度ベクトル除算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。s1 が 1 の場合は VFPR[rs] の代わりに、スカラレジスタ (SFPR[rs]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VDIV.D</b>											Vector Divide Double														
ベクトル除算											VECTOR														
31						26 25					21 20				16 15				11 10 9	8	7	6	5	0	
011111				rs				rt				rd				00		s1	s0	s	001011				
<b>VFP</b>											<b>0</b>											<b>DIV.D</b>			

ニーモニック:

VDIV.D.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 0)
VDIV.D.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 0)
VDIV.D.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 0)
VDIV.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, sync(s) = 1)
VDIV.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, sync(s) = 1)
VDIV.D.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, sync(s) = 1)

機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \div \text{VFPR}[\text{rt}]$$

例外:

Vector Floating Point Exception :

概要:

倍精度ベクトル除算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。s1 が 1 の場合は VFPR[rs] の代わりに、スカラレジスタ (SFPR[rs]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VABS.S</b>										Vector Absolute Single		
ベクトル絶対値演算										VECTOR		
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		00000			rd		0000		s	000101	
<b>VFP</b>			<b>0</b>			<b>0</b>		<b>ABS.S</b>				

ニーモニック:

VABS.S rd, rs (sync(s) = 0)

VABS.S.sy rd, rs (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow |VFPR[rs]|$

例外:

Vector Floating Point Exception:

概要:

単精度ベクトル絶対値演算。sync が 1 の場合は、投機実行を抑制する。

<b>VABS.D</b>										Vector Absolute Double																		
ベクトル絶対値演算										VECTOR																		
31						26 25						21 20						16 15						11 10			7 6 5	0
011111					rs					00000					rd					0000		s	001101					
<b>VFP</b>					<b>0</b>					<b>0</b>					<b>0</b>					<b>0</b>		<b>0</b>	<b>ABS.D</b>					

ニーモニック:

VABS.D rd, rs (sync(s) = 0)  
 VABS.D.sy rd, rs (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow |VFPR[rs]|$

例外:

Vector Floating Point Exception :

概要:

倍精度ベクトル絶対値演算。sync が 1 の場合は、投機実行を抑制する。

<b>VMOV.S</b>										Vector Move Single																					
ベクトル転送										VECTOR																					
31						26 25						21 20						16 15						11 10			7 6 5				0
011111					rs					00000					rd					0000		s	000110								
<b>VFP</b>					<b>0</b>					<b>0</b>					<b>0</b>					<b>0</b>		<b>0</b>	<b>MOV.S</b>								

ニーモニック:

VMOV.S rd, rs (sync(s) = 0)  
 VMOV.S.sy rd, rs (sync(s) = 1)

機能:

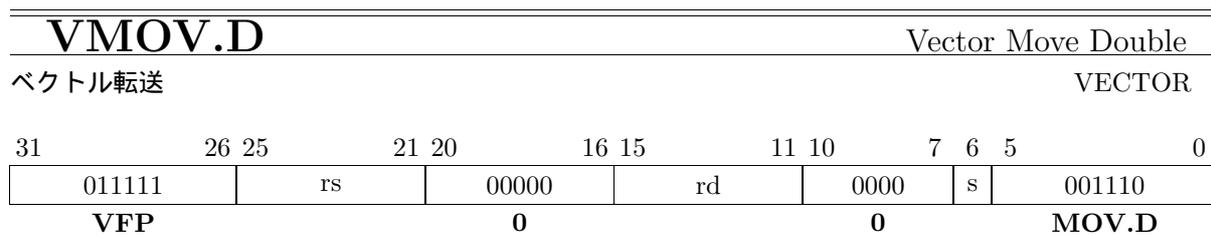
$VFPR[rd] \leftarrow VFPR[rs]$

例外:

Vector Floating Point Exception :

概要:

単精度ベクトル転送命令 . sync が 1 の場合は , 投機実行を抑制する .



ニーモニック:

VMOV.D rd, rs (sync(s) = 0)

VMOV.D.sy rd, rs (sync(s) = 1)

機能:

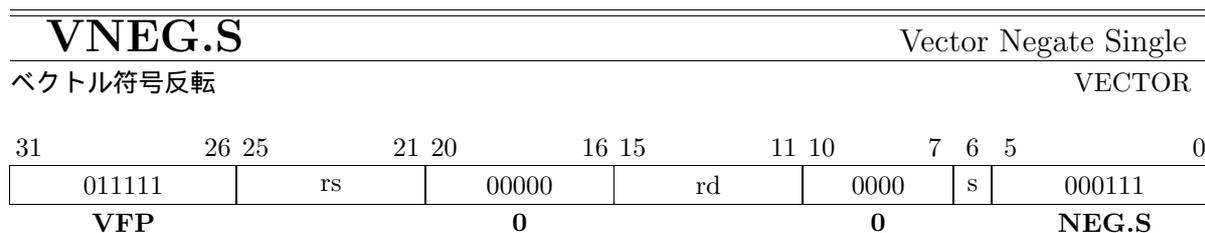
$VFPR[rd] \leftarrow VFPR[rs]$

例外:

Vector Floating Point Exception :

概要:

倍精度ベクトル転送命令 . sync が 1 の場合は , 投機実行を抑制する .



ニーモニック:

VNEG.S rd, rs (sync(s) = 0)  
 VNEG.S.sy rd, rs (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow -1 \times VFPR[rs]$

例外:

Vector Floating Point Exception :

概要:

単精度ベクトル符号反転演算。sync が 1 の場合は、投機実行を抑制する。

<b>VNEG.D</b>										Vector Negate Double		
ベクトル符号反転										VECTOR		
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		00000			rd		0000		s	001111	
<b>VFP</b>			<b>0</b>			<b>0</b>		<b>NEG.D</b>				

ニーモニック:

VNEG.D rd, rs (sync(s) = 0)  
 VNEG.D.sy rd, rs (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow -1 \times VFPR[rs]$

例外:

Vector Floating Point Exception :

概要:

倍精度ベクトル符号反転演算。sync が 1 の場合は、投機実行を抑制する。

<b>VMADD.S</b>										Vector Multiply and Add Single									
ベクトル積和演算										VECTOR									
31	26 25		21 20		16 15		11 10		8	7	6	5	0						
011111	rs		rt		rd		000		s0	s	010000								
<b>VFP</b>							<b>0</b>			<b>MADD.S</b>									

## ニーモニック:

VMADD.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMADD.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMADD.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMADD.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \times \text{VFPR}[\text{rt}] + \text{VFPR}[\text{rd}]$$

## 例外:

Vector Floating Point Exception:

## 概要:

単精度ベクトル積和演算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMADD.D</b>										Vector Multiply and Add Double									
ベクトル積和演算										VECTOR									
31	26 25		21 20		16 15		11 10		8	7	6	5					0		
011111	rs		rt		rd		000		s0	s	011000								
<b>VFP</b>										<b>0</b>		<b>MADD.D</b>							

ニーモニック:

VMADD.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMADD.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMADD.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMADD.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

$VFPR[rd] \leftarrow VFPR[rs] \times VFPR[rt] + VFPR[rd]$

例外:

Vector Floating Point Exception:

概要:

倍精度ベクトル積和演算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMSUB.S</b>										Vector Multiply and Subtract Single												
ベクトル積差演算										VECTOR												
31	26	25	21	20	16	15	11	10	8	7	6	5	0									
011111			rs			rt			rd			000		s0	s	010001						
VFP										MSUB.S												

## ニーモニック:

VMSUB.S.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMSUB.S.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMSUB.S.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMSUB.S.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \times \text{VFPR}[\text{rt}] - \text{VFPR}[\text{rd}]$$

## 例外:

Vector Floating Point Exception:

## 概要:

単精度ベクトル積差演算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMSUB.D</b>										Vector Multiply and Subtract Double										
ベクトル積差演算										VECTOR										
31	26	25	21	20	16	15	11	10	8	7	6	5								0
011111			rs			rt			rd			000		s0	s	011001				
VFP										0					MSUB.D					

## ニーモニック:

VMSUB.D.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VMSUB.D.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VMSUB.D.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VMSUB.D.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

## 機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \times \text{VFPR}[\text{rt}] - \text{VFPR}[\text{rd}]$$

## 例外:

Vector Floating Point Exception :

## 概要:

倍精度ベクトル積差演算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VCMP.S</b>										Vector Compare Single			
ベクトル比較										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011111	rs		rt		rd		cond	s0	s	010010			
<b>VFP</b>										<b>CMP.S</b>			

ニーモニック:

VCMP.S.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.S.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.S.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.S.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VFPR[rs] cond VFPR[rt] then
    VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif

```

例外:

Vector Floating Point Exception:

概要:

単精度ベクトル比較命令。条件 (cond) により VFPR[rd] の値が決定する。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

VCMP.D										Vector Compare Double			
ベクトル比較										VECTOR			
31	26	25	21	20	16	15	11	10	8	7	6	5	0
011111		rs		rt		rd		cond	s0	s	011010		
VFP										CMP.D			

ニーモニック:

VCMP.D.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.D.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.D.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.D.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VFPR[rs] cond VFPR[rt] then    VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif

```

例外:

Vector Floating Point Exception:

概要:

倍精度ベクトル比較命令。条件 (cond) により VFPR[rd] の値が決定する。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VCMPTS.S</b>										Vector Compare Single to Scalar Register																							
ベクトル比較										VECTOR																							
31	26	25	21	20	16	15	11	10	8	7	6	5	0																				
011111					rs					rt					rd					cond					s0		s		010011				
VFP										CMPTS.S																							

ニーモニック:

VCMPTS.S.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.S.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.S.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.S.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```
if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFRP[rd] ← 0
endif
```

例外:

Vector Floating Point Exception :

概要:

単精度ベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VCMPTS.D</b>										Vector Compare Double to Scalar Register																		
ベクトル比較										VECTOR																		
31					26 25					21 20					16 15					11 10	8	7	6	5	0			
011111					rs					rt					rd					cond		s0	s	011011				
<b>VFP</b>										<b>CMPTS.D</b>																		

ニーモニック:

VCMPTS.D.cond.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.D.cond.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.D.cond.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.D.cond.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFPR[rd] ← 0
endif

```

例外:

Vector Floating Point Exception :

概要:

倍精度ベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には、eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VCVT.S.D</b>												
Vector Convert to Single from Double												
ベクトルフォーマット変換												
VECTOR												
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		00000		rd		0000		s	101000		
<b>VFP</b>			<b>0</b>		<b>0</b>		<b>0</b>		<b>VCVT.S.D</b>			

ニーモニック:

VCVT.S.D rd, rs (sync(s) = 0)  
 VCVT.S.D.sy rd, rs (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow \text{Double\_to\_Single}(VFPR[rs])$

例外:

Vector Floating Point Exception :

概要:

倍精度フォーマットから単精度フォーマットへ変換する。sync が 1 の場合は、投機実行を抑制する。

<b>VCVT.S.W</b>							Vector Convert to Single from Word					
ベクトルフォーマット変換							VECTOR					
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		00000		rd		0000		s	100010		
<b>VFP</b>			<b>0</b>		<b>0</b>		<b>0</b>		<b>s</b>	<b>VCVT.S.W</b>		

ニーモニック:

VCVT.S.W rd, rs (sync(s) = 0)  
 VCVT.S.W.sy rd, rs (sync(s) = 1)

機能:

$\text{VFPR}[\text{rd}] \leftarrow \text{Word\_to\_Single}(\text{VFPR}[\text{rs}])$

例外:

Vector Floating Point Exception :

概要:

整数フォーマットから単精度フォーマットへ変換する。sync が 1 の場合は、投機実行を抑制する。

<b>VCVT.D.S</b>												
Vector Convert to Double from Single												
ベクトルフォーマット変換												
VECTOR												
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		00000		rd		0000		s	100001		
<b>VFP</b>			<b>0</b>			<b>0</b>			<b>VCVT.D.S</b>			

ニーモニック:

VCVT.D.S rd, rs (sync(s) = 0)  
 VCVT.D.S.sy rd, rs (sync(s) = 1)

機能:

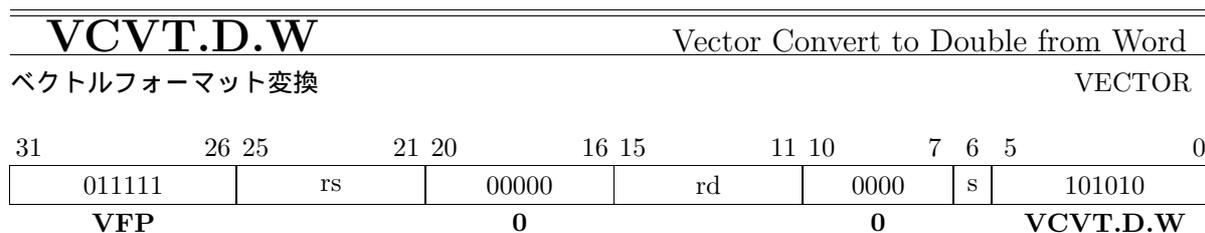
$VFPR[rd] \leftarrow \text{Single\_to\_Double}(VFPR[rs])$

例外:

Vector Floating Point Exception :

概要:

単精度フォーマットから倍精度フォーマットへ変換する。sync が 1 の場合は、投機実行を抑制する。



ニーモニック:

VCVT.D.W rd, rs (sync(s) = 0)  
 VCVT.D.W.sy rd, rs (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow \text{Word\_to\_Double}(VFPR[rs])$

例外:

Vector Floating Point Exception :

概要:

整数フォーマットから倍精度フォーマットへ変換する。sync が 1 の場合は、投機実行を抑制する。

<b>VCVT.W.S</b>												
Vector Convert to Word from Single												
ベクトルフォーマット変換												
VECTOR												
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		00000		rd		0000		s	100100		
<b>VFP</b>			<b>0</b>			<b>0</b>			<b>VCVT.W.S</b>			

ニーモニック:

VCVT.W.S rd, rs (sync(s) = 0)

VCVT.W.S.sy rd, rs (sync(s) = 1)

機能:

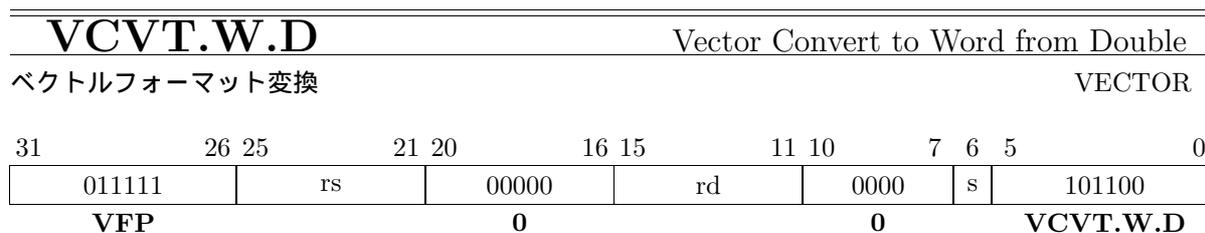
$VFPR[rd] \leftarrow \text{Single\_to\_Word}(VFPR[rs])$

例外:

Vector Floating Point Exception :

概要:

単精度フォーマットから整数フォーマットへ変換する。sync が 1 の場合は、投機実行を抑制する。



ニーモニック:

VCVT.W.D rd, rs (sync(s) = 0)  
 VCVT.W.D.sy rd, rs (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow \text{Double\_to\_Word}(VFPR[rs])$

例外:

Vector Floating Point Exception :

概要:

倍精度フォーマットから整数フォーマットへ変換する。sync が 1 の場合は、投機実行を抑制する。

<b>VFMFC</b>										Move from Vector Floating-Point Control Register									
制御レジスタリード										VECTOR									
31	26	25	21	20	16	15	11	10	7	6	5				0				
011111			rs			00000			rd			0000			s	110000			
VFP			0			0			0			MFC							

ニーモニック:

VFMFC rd, rs (sync(s) = 0)  
 VFMFC.sy rd, rs (sync(s) = 1)

機能:

$FPR[rd] \leftarrow VFCTRL[rs]$

例外:

概要:

浮動小数点ベクトル制御レジスタリード命令 . rs で指定された制御レジスタの値を浮動小数点レジスタに格納する . sync が 1 の場合は , 投機実行を抑制する .

<b>VFMTc</b>												
Move to Vector Floating-Point Control Register												
制御レジスタライト							VECTOR					
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		00000		rd		0000		s	110001		
<b>VFP</b>			<b>0</b>			<b>0</b>			<b>MTC</b>			

ニーモニック:

VFMTc rd, rs (sync(s) = 0)  
 VFMTc.sy rd, rs (sync(s) = 1)

機能:

$VFCTRL[rd] \leftarrow FPR[rs]$

例外:

概要:

浮動小数点ベクトル制御レジスタライト命令。rd で指定された制御レジスタに浮動小数点レジスタの値を格納する。sync が 1 の場合は、投機実行を抑制する。

<b>VFMFS</b>		Move from Vector Floating-Point Scalar Register				
浮動小数点スカラレジスタリード		VECTOR				
31	26 25	21 20	16 15	11 10	7 6 5	0
011111	rs	00000	rd	0000	s	110010
<b>VFP</b>		<b>0</b>		<b>0</b>		<b>MFS</b>

ニーモニック:

VFMFS rd, rs (sync(s) = 0)  
 VFMFS.sy rd, rs (sync(s) = 1)

機能:

$FPR[rd] \leftarrow SFPR[rs]$

例外:

Vector Floating Point Exception :

概要:

浮動小数点スカラレジスタリード命令 . rs で指定された浮動小数点スカラレジスタの値を浮動小数点レジスタに格納する . sync が 1 の場合は , 投機実行を抑制する .

<b>VFMTS</b>										Move to Vector Floating-Point Scalar Register															
浮動小数点スカラレジスタライト										VECTOR															
31		26		25		21		20		16		15		11		10		7		6		5		0	
011111		rs		00000		rd		0000		s		110011													
VFP				0						0				MTS											

ニーモニック:

VFMTS rd, rs (sync(s) = 0)

VFMTS.sy rd, rs (sync(s) = 1)

機能:

SFPR[rd] ← FPR[rs]

例外:

Vector Floating Point Exception :

概要:

浮動小数点スカラレジスタライト命令 . rd で指定された浮動小数点スカラレジスタに浮動小数点レジスタの値を格納する . sync が 1 の場合は , 投機実行を抑制する .

<b>VFMFV</b>		Move from Vector Floating-Point Vector Register					
浮動小数点ベクトルレジスタリード						VECTOR	
31	26 25	21 20	16 15	11 10	7 6 5	0	
011111	rs	rt	rd	0000	s	110100	
<b>VFP</b>				<b>0</b>	<b>MFV</b>		

ニーモニック:

VFMFV rd, rs, rt (sync(s) = 0)  
 VFMFV.sy rd, rs, rt (sync(s) = 1)

機能:

SFPR[rd] ← VFPR[rs][rt]

例外:

Vector Floating Point Exception :

概要:

浮動小数点ベクトルレジスタリード命令 . rs で指定された浮動小数点ベクトルレジスタの rt 番目の要素の値を浮動小数点レジスタに格納する . sync が 1 の場合は , 投機実行を抑制する .

<b>VFMTV</b>												
Move to Vector Floating-Point Vector Register												
浮動小数点ベクトルレジスタライト												
VECTOR												
31	26	25	21	20	16	15	11	10	7	6	5	0
011111	rs		rt		rd		0000		s	110101		
<b>VFP</b>							<b>0</b>		<b>MTV</b>			

ニーモニック:

VFMTV rd, rs, rt (sync(s) = 0)

VFMTV.sy rd, rs, rt (sync(s) = 1)

機能:

SFPR[rd] ← VFPR[rs][rt]

例外:

Vector Floating Point Exception :

概要:

浮動小数点ベクトルレジスタライト命令。rd で指定された浮動小数点ベクトルレジスタの rt 番目の要素に浮動小数点レジスタの値を書き込む。sync が 1 の場合は、投機実行を抑制する。

<b>VFMTM</b>		Move to Vector Floating-Point Mask Register	
浮動小数点ベクトルマスクレジスタライト		VECTOR	
31	26 25	21 20	7 6 5 0
011111	rs	0000000000000000	s 011110
<b>VFP</b>		<b>0</b>	<b>MTM</b>

ニーモニック:

VFMTM rs (sync(s) = 0)  
 VFMTM.sy rs (sync(s) = 1)

機能:

$VFCTRL[Mask Register] \leftarrow VSFPR[rs]$

例外:

Vector Floating Point Exception :

概要:

浮動小数点ベクトルマスクレジスタライト命令 . rd で指定した浮動小数点スカラレジスタの値をマスクレジスタに格納する . sync が 1 の場合は , 投機実行を抑制する .

<b>VFRSV</b>		Vector Floating-Point Register Reserve					
浮動小数点ベクトルレジスタ予約						VECTOR	
31	26 25	21 20	16 15	11 10	7 6 5	0	
011111	rs	00000	rd	0000	s	110110	
<b>VFP</b>		<b>0</b>		<b>0</b>		<b>RSV</b>	

ニーモニック:

VFRSV rd, rs (sync(s) = 0)  
 VFRSV.sy rd, rs (sync(s) = 1)

機能:

```
reserve_vector_register(GPR[rs])
if success_reserve_operation then
  GPR[rd] ← 1
else
  GPR[rd] ← 0
endif
```

例外:

None

概要:

浮動小数点ベクトルレジスタ予約命令。GPR[rs] に予約するレジスタの構成を指定する。予約に成功した場合は GPR[rd] に 1 が、失敗した場合は 0 が格納される。sync が 1 の場合は、投機実行を抑制する。

<b>VFRLS</b>		Vector Floating-Point Register Release			
浮動小数点ベクトルレジスタ開放					VECTOR
31	26 25	16 15	11 10	7 6 5	0
011111	0000000000	rd	0000	s	110111
<b>VFP</b>		<b>0</b>	<b>0</b>		<b>RLS</b>

ニーモニック:

VFRLS rd (sync(s) = 0)  
 VFRLS.sy rd (sync(s) = 1)

機能:

```
release_vector_register()
if success_release_operation then
  GPR[rd] ← 1
else
  GPR[rd] ← 0
endif
```

例外:

None

概要:

浮動小数点ベクトルレジスタ開放命令。開放に成功した場合は GPR[rd] に 1 が、失敗した場合は 0 が格納される。sync が 1 の場合、投機実行を抑制する。

<b>VFDCI</b>		Vector Floating-Point Define Compound Instruction					
浮動小数点ベクトル複合命令定義						VECTOR	
31	26 25	21 20	16 15	11 10	7 6 5	0	
011111	rs	00000	rd	0000	s	101110	
<b>VFP</b>		<b>0</b>		<b>0</b>		<b>DCI</b>	

ニーモニック:

VFDCI rd, rs (sync(s) = 0)  
 VFDCI.sy rd, rs (sync(s) = 1)

機能:

VFCPD[rd] ← GPR[rs]

例外:

Vector Floating Point Exception :

概要:

浮動小数点ベクトル複合命令の定義を行う。GPR[rs] で定義した命令を rd で指定した複合命令バッファのアドレスに格納する。sync が 1 の場合、投機実行を抑制する。

<b>VFECI</b>		Vector Floating-Point Execute Compound Instruction									
浮動小数点ベクトル複合命令実行						VECTOR					
31	26	25	21	20	16	15	11	10	6	5	0
011111		rs		rt		rd		no		101111	
<b>VFP</b>						<b>ECI</b>					

ニーモニック:

VFDCI rd, rs, rt, no

機能:

$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \text{ op } \text{VFPR}[\text{rt}]$

例外:

Vector Floating Point Exception :

概要:

浮動小数点ベクトル複合命令の実行を行う。no で指定した複合命令バッファのアドレスから命令を実行する。

<b>VFLW</b>		Vector Floating-Point Load Word	
浮動小数点ベクトルロード		VECTOR	
31	26 25	21 20	16 15
011111	base	rt	00000000
<b>VFP</b>			<b>0</b>
			7 6 5
			s
			111010
			<b>LW</b>
			0

ニーモニック:

VFLW rt, base (sync(s) = 0)  
 VFLW.sy rt, base (sync(s) = 1)

機能:

VFPR[rt] ← MEM.WORD[GPR[base]]

例外:

Vector Floating Point Exception :  
 D-TLB No Entry Matched :  
 D-TLB Protection Error :  
 Data Address Miss Align ( Load ) :

概要:

メモリから浮動小数点ベクトルレジスタにロードする .sync が 1 の場合 , 投機実行を抑制する .

<b>VFLD</b>		Vector Floating-Point Load Double			
浮動小数点ベクトルロード		VECTOR			
31	26 25	21 20	16 15	7 6 5	0
011111	base	rt	00000000	s	111011
<b>VFP</b>		<b>0</b>		<b>LD</b>	

## ニーモニック:

VFLD rt, base (sync(s) = 0)  
 VFLD.sy rt, base (sync(s) = 1)

## 機能:

VFPR[rt] ← MEM.DWORD[GPR[base]]

## 例外:

Vector Floating Point Exception :  
 D-TLB No Entry Matched :  
 D-TLB Protection Error :  
 Data Address Miss Align ( Load ) :

## 概要:

メモリから浮動小数点ベクトルレジスタにロードする。sync が 1 の場合、投機実行を抑制する。

<b>VFSW</b>										Vector Floating-Point Store Word																				
浮動小数点ベクトルストア										VECTOR																				
31						26 25						21 20						16 15						7 6 5						0
011111					base					rt					00000000					s	111110									
<b>VFP</b>										<b>0</b>										<b>SW</b>										

ニーモニック:

VFSW rt, base (sync(s) = 0)  
 VFSW.sy rt, base (sync(s) = 1)

機能:

MEM.WORD[GPR[base]] ← VFPR[rt]

例外:

Vector Floating Point Exception :  
 D-TLB No Entry Matched :  
 D-TLB Protection Error :  
 Data Address Miss Align ( Store ) :

概要:

浮動小数点ベクトルレジスタからメモリにストアする .sync が 1 の場合 , 投機実行を抑制する .

<b>VFP</b>		<b>SD</b>	
011111		111111	
base		s	
rt		00000000	
VFP		SD	

ニーモニック:

VFSD rt, base (sync(s) = 0)  
 VFSD.sy rt, base (sync(s) = 1)

機能:

MEM.DWORD[GPR[base]] ← VFPR[rt]

例外:

Vector Floating Point Exception :  
 D-TLB No Entry Matched :  
 D-TLB Protection Error :  
 Data Address Miss Align ( Store ) :

概要:

浮動小数点ベクトルレジスタからメモリにストアする .sync が 1 の場合, 投機実行を抑制する .

<b>VADD.PS</b>										Vector Add Paired Single																									
ベクトル加算										VECTOR																									
31						26 25						21 20						16 15						11 10	9	8	7	6	5						0
111111					rs					rt					rd					0	s2	0	s0	s	000000										
<b>VFP.PS</b>																				<b>0</b>		<b>0</b>		<b>ADD.S</b>											

## ニーモニック:

VADD.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VADD.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VADD.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VADD.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VADD.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VADD.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VADD.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VADD.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] + \text{VFPR}[\text{rt}]$$

## 例外:

Vector Integer Exception:

## 概要:

32bit × 2 単精度ベクトル加算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。s2 が 1 の場合 VFPR[rt] の下位 32bit を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VSUB.PS</b>										Vector Subtract Paired Single																			
ベクトル減算										VECTOR																			
31					26 25					21 20					16 15					11 10 9 8 7 6 5	0								
111111					rs					rt					rd					0	s2	s1	s0	s	000001				
<b>VFP.PS</b>										<b>0</b>										<b>SUB.S</b>									

### ニーモニック:

VSUB.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 0)
VSUB.PS.sv rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 0, sync(s) = 0)
VSUB.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 0)
VSUB.PS.sv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 1, sync(s) = 0)
VSUB.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 0, sync(s) = 1)
VSUB.PS.sv.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 0, sync(s) = 1)
VSUB.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 1)
VSUB.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar1(s1) = 0, scalar2(s2) = 1, sync(s) = 1)
VSUB.PS.sv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar1(s1) = 1, scalar2(s2) = 1, sync(s) = 1)

### 機能:

$VFPR[rd] \leftarrow VFPR[rs] - VFPR[rt]$

### 例外:

Vector Integer Exception:

### 概要:

32bit × 2 単精度ベクトル減算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラーレジスタ (SFPR[rt]) を用いて演算を行う。s1 が 1 の場合は VFPR[rs] の代わりに、スカラーレジスタ (SFPR[rs]) を用いて演算を行う。s2 が 1 の場合 VFPR[rt] の下位 32bit を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMUL.PS</b>										Vector Multiply Paired Single																			
ベクトル乗算										VECTOR																			
31						26 25						21 20						16 15						11 10 9 8 7 6 5	0				
111111					rs					rt					rd					0	s2	0	s0	s	000010				
<b>VFP.PS</b>																				<b>0</b>	<b>0</b>	<b>MUL.S</b>							

## ニーモニック:

VMUL.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMUL.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMUL.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMUL.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMUL.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMUL.PS.vs.sync rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMUL.PS.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMUL.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \times \text{VFPR}[\text{rt}]$$

## 例外:

Vector Integer Exception :

## 概要:

32bit × 2 単精度ベクトル乗算 . s0 が 1 の場合は VFPR[rt] の代わりに , スカラレジスタ (SFPR[rt]) を用いて演算を行う . s2 が 1 の場合 VFPR[rt] の下位 32bit を用いて演算を行う . sync が 1 の場合は , 投機実行を抑制する .

<b>VABS.PS</b>										Vector Absolute Paired Single									
ベクトル絶対値演算										VECTOR									
31	26	25	21	20	16	15	11	10	7	6	5				0				
111111			rs			00000			rd			0000			s	000101			
<b>VFP.PS</b>					<b>0</b>					<b>0</b>					<b>ABS.S</b>				

ニーモニック:

VABS.PS rd, rs, rt (sync(s) = 0)

VABS.PS.sy rd, rs, rt (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow |VFPR[rs]|$

例外:

Vector Integer Exception:

概要:

32bit × 2 単精度ベクトル絶対値演算 . sync が 1 の場合は, 投機実行を抑制する .

<b>VNEG.PS</b>										Vector Negate Paired Single																													
ベクトル符号反転										VECTOR																													
31						26 25						21 20						16 15						11 10			7 6 5				0								
111111					rs					00000					rd					0000					s		000111												
<b>VFP.PS</b>										<b>0</b>										<b>0</b>										<b>NEG.S</b>									

ニーモニック:

VNEG.PS rd, rs, rt (sync(s) = 0)

VNEG.PS.sy rd, rs, rt (sync(s) = 1)

機能:

$VFPR[rd] \leftarrow -1 \text{ times } VFPR[rs]$

例外:

Vector Integer Exception:

概要:

32bit × 2 単精度ベクトル符号反転演算。sync が 1 の場合は、投機実行を抑制する。

<b>VMADD.PS</b>										Vector Multiply and Add Paired Single									
ベクトル積和演算										VECTOR									
31	26 25		21 20		16 15		11 10		9	8	7	6 5		0					
111111			rs		rt		rd		0	s2	0	s0	s	010000					
<b>VFP.PS</b>										<b>0</b>		<b>0</b>		<b>MADD.S</b>					

#### ニーモニック:

VMADD.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMADD.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMADD.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMADD.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMADD.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMADD.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMADD.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMADD.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

#### 機能:

$VFPR[rd] \leftarrow VFPR[rs] \textit{ times } VFPR[rt] + VFPR[rd]$

#### 例外:

Vector Integer Exception :

#### 概要:

32bit × 2 単精度ベクトル積和演算。s0 が 1 の場合は VFPR[rt] の代わりに、スカラーレジスタ (SFPR[rt]) を用いて演算を行う。s2 が 1 の場合は VFPR[rt] の下位 32bit を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。

<b>VMSUB.PS</b>											Vector Multiply and Subtract Paired Single											
ベクトル積差演算											VECTOR											
31	26 25				21 20				16 15				11 10 9 8 7 6 5				0					
111111	rs				rt				rd				0	s2	0	s0	s	010001				
<b>VFP.PS</b>											<b>0</b>				<b>0</b>				<b>MSUB.S</b>			

## ニーモニック:

VMSUB.PS.vv rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PS.vs rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 0)
VMSUB.PS.vv.lo32 rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PS.vs.lo32 rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 0)
VMSUB.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 0, sync(s) = 1)
VMSUB.PS.vv.lo32.sy rd, rs, rt	(scalar0(s0) = 0, scalar2(s2) = 1, sync(s) = 1)
VMSUB.PS.vs.lo32.sy rd, rs, rt	(scalar0(s0) = 1, scalar2(s2) = 1, sync(s) = 1)

## 機能:

$$\text{VFPR}[\text{rd}] \leftarrow \text{VFPR}[\text{rs}] \times \text{VFPR}[\text{rt}] - \text{VFPR}[\text{rd}]$$

## 例外:

Vector Integer Exception :

## 概要:

32bit × 単精度ベクトル積差演算 . s0 が 1 の場合は VFPR[rt] の代わりに, スカラレジスタ (SFPR[rt]) を用いて演算を行う . s2 が 1 の場合は VFPR[rt] の下位 32bit を用いて演算を行う . sync が 1 の場合は, 投機実行を抑制する .

<b>VCMP.PS</b>										Vector Compare Paired Single																								
ベクトル比較										VECTOR																								
31						26 25						21 20						16 15						11 10	8	7	6	5						0
111111					rs					rt					rd					cond		s0	s	010010										
<b>VFP.PS</b>										<b>CMP.S</b>																								

#### ニーモニック:

VCMP.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMP.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMP.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMP.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

#### 機能:

```

if VFPR[rs] cond VFPR[rt] then
    VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif

```

#### 例外:

Vector Integer Exception :

#### 概要:

32bit × 2 単精度ベクトル比較命令。条件 (cond) により VFPR[rd] の値が決定する。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。

<b>VSCMP.PS</b>										Vector Compare Paired Single																	
ベクトル比較										VECTOR																	
31		26		25		21		20		16		15		11		10		8		7		6		5		0	
111111		rs		rt		rd		cond		s0		s		010010													
<b>VFP.PS</b>										<b>SCMP.S</b>																	

ニーモニック:

VSCMP.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMP.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMP.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMP.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VFPR[rs] cond VFPR[rt] then
    VFPR[rd] ← 1
else
    VFPR[rd] ← 0
endif

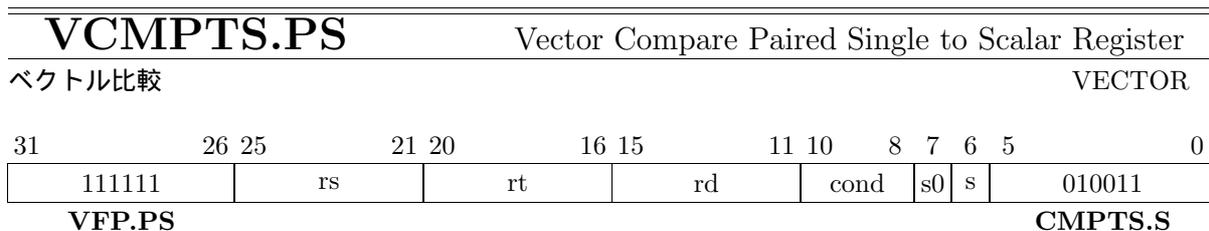
```

例外:

Vector Integer Exception:

概要:

32bit × 2 単精度ベクトル比較命令。条件 (cond) により VFPR[rd] の値が決定する。VFPR[rt] の下位 32bit を用いて演算を行う。s0 が 1 の場合は VFPR[rt] の代わりに、スカラーレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VCMPTS.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VCMPTS.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VCMPTS.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VCMPTS.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFRP[rd] ← 0
endif

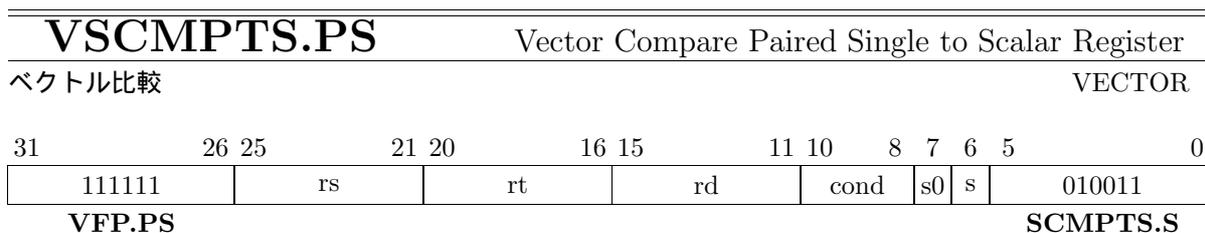
```

例外:

Vector Integer Exception:

概要:

32bit × 2 単精度ベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラーレジスタに格納される。s0 が 1 の場合は VFPR[rt] の代わりに、スカラーレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



ニーモニック:

VSCMPTS.cond.PS.vv rd, rs, rt	(scalar0(s0) = 0, sync(s) = 0)
VSCMPTS.cond.PS.vs rd, rs, rt	(scalar0(s0) = 1, sync(s) = 0)
VSCMPTS.cond.PS.vv.sy rd, rs, rt	(scalar0(s0) = 0, sync(s) = 1)
VSCMPTS.cond.PS.vs.sy rd, rs, rt	(scalar0(s0) = 1, sync(s) = 1)

機能:

```

if VFPR[rs] cond VFPR[rt] then
    SFPR[rd] ← 1
else
    SFPR[rd] ← 0
endif

```

例外:

Vector Integer Exception:

概要:

32bit × 2 単精度ベクトル比較命令。結果は各要素ごとに 1bit を割り当ててスカラレジスタに格納される。VFPR[rt] の下位 32bit を用いて演算を行う。s0 が 1 の場合は VFPR[rt] の代わりに、スカラレジスタ (SFPR[rt]) を用いて演算を行う。sync が 1 の場合は、投機実行を抑制する。比較条件 (cond) には eq(=), gt(>), lt(<), ne(≠), le(≤), ge(≥) のどれかを指定する。



# 4

## アドレスデコーダ

### 4.1 レジスタインターフェース

アドレスデコーダの各種設定レジスタはシステムレジスタとして定義されている．そのため各種レジスタを設定するためには `mtc0` 命令を用いる．

設定する値はモジュールにより異なる．

- 標準 (TYPE A)

31	16 15	8 7	0
-	Address	Mask	

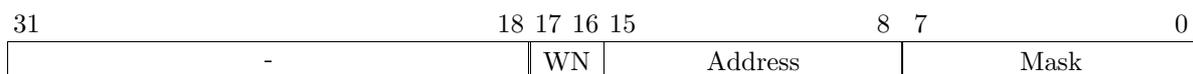
- I/O (TYPE B)

31	12 11	8 7	4 3	0
-	Address	-	Mask	

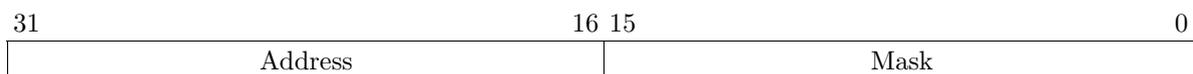
- 外部バス (TYPE C)

31	19 18 17 16 15	8 7	0
-	AR WN	Address	Mask

- リンク メモリ (TYPE D)



- I/O Base (TYPE E)



bit 名	機能
Address	ベースアドレス
Mask	マスク
WN	Word Number
AR	Auto Ready

## 4.2 アドレスマップ

接続されるモジュール	初期デコードアドレス	設定レジスタのアドレス	タイプ
ROM (EXT_0)	0x00000000 ~ 0x00ffffff	0xa0	TYPE C
EXT_1	0x20000000 ~ 0x20ffffff	0xa1	TYPE C
SDRAM IF0	0x80000000 ~ 0x87ffffff	0xa2	TYPE A
SDRAM IF1	0x88000000 ~ 0x8fffffff	0xa3	TYPE A
SDRAM IF2	0x90000000 ~ 0x97ffffff	0xa4	TYPE A
SRAM	0x98000000 ~ 0x9fffffff	0xa5	TYPE A
LINK SDRAM	0x04000000 ~ 0x04ffffff	0xa6	TYPE D
LINK DPM	0xc0000000 ~ 0xcfffffff	0xa7	TYPE D
DMAC0	0xffff0000 ~ 0xffff0fff	0xa8	TYPE B
DMAC1	0xffff1000 ~ 0xffff1fff	0xa9	TYPE B
DMAC2	0xffff2000 ~ 0xffff2fff	0xaa	TYPE B
PCI	0xffff3000 ~ 0xffff3fff	0xab	TYPE B
DMAC3	0xffff4000 ~ 0xffff4fff	0xac	TYPE B
Ether	0xffff5000 ~ 0xffff5fff	0xad	TYPE B
UART	0xffff6000 ~ 0xffff6fff	0xae	TYPE B
PP	0xffff7000 ~ 0xffff7fff	0xaf	TYPE B
IEEE1394	0xffff8000 ~ 0xffff8fff	0xb0	TYPE B
LINK	0xfffe0000 ~ 0xfffeffff	0xb1	TYPE E
IRC	0xffff9000 ~ 0xffff9fff	0xb2	TYPE B
CLK Generator	0xffffa000 ~ 0xffffafff	0xb3	TYPE B
SPI	0xffffb000 ~ 0xffffbfff	0xb7	TYPE B
Parallel I/O	0xffffc000 ~ 0xffffcfff	0xbe	TYPE B
MDMAC	0xffffd000 ~ 0xffffdfff	0xb4	TYPE B
LINK SDRAM Mode	0xfffe000 ~ 0xfffefff	0xb5	TYPE B
SDRAM Mode	0xffff000 ~ 0xffff7fff	0xb6	TYPE B
RTC	0xfffff800 ~ 0xffffffff	0xc1	TYPE B
I/O Base	0xffff0000 ~ 0xffffffff	0xb8	TYPE E



# 5

## MMU

---

Responsive Multithreaded Processor のキャッシュシステムは命令キャッシュ、データキャッシュともに物理キャッシュなので MMU でのアドレス変換はプロセッシングコアとキャッシュの間で行う。また、アドレス空間上に、TLB によるアドレス変換が行なわれない領域は存在しない。

### 5.1 TLB エントリ

本 MMU における TLB エントリ数は命令 MMU、データ MMU ともに 64 エントリである。エントリへの設定方法は full associative 方式とし、設定を行うページ番号に関わらずどのエントリでも設定を行うことを可能である。

以下本 MMU における TLB エントリの機能の詳細と特徴について述べて行く。

表 5.1 に TLB エントリの設定項目の一覧を示す。TLB エントリは全部で 8byte であるが、設定に際しては 32bit の整数型データを用いて行うため便宜上エントリ 1 とエントリ 2 に分かれる。

また TLB エントリに指定した仮想アドレスとコンテキスト ID が一致したかどうかの判断はエントリ番号の大きな順に行われる。そのため複数のエントリが一致した場合には、よりエントリ番号が大きな TLB エントリの設定値を用いてアドレス変換が行われる。

TLB のエントリを設定した直後、そのエントリの LRU 情報はもっとも最近に参照されたものとして扱われる。

TLB エントリを初期化した場合、各フィールドの値は表 5.2 のようになる。また、TLB エントリの LRU 情報を初期化すると LRU の順序はエントリ 0 がもっとも最近アクセスされた tlb エントリとなり、エントリ 1,2,3,... 62, 63 と順にアクセスが古い、という状態になる。

#### VPN

この VPN フィールドには、アドレス変換を行う仮想アドレスがエントリを検索するために必要な仮想ページ番号を保持する。Responsive Multithreaded Processor は仮想アドレスに 32bit の信号を用い、最小ページサイズが 4KB であるため、TLB エントリには表 5.1 に示すように仮想アドレスの上位 20bit を保持する。VPN フィールドはエントリ 1 に属し、エントリ設定時には設定を行う仮想ページ番号を設定データの上位 20bit に指定する。

表 5.1: TLB エントリー一覧

フィールド名	エントリ番号	データ割り当て	機能
VPN	エントリ 1	[31:12]	仮想ページ番号 (Virtual Page Number)
LOCK	エントリ 1	[11]	エントリのロック
PROTECT	エントリ 1	[10:8]	保護情報
SHARE_TH	エントリ 1	[7:0]	エントリの有効情報と共有情報
PPN	エントリ 2	[31:12]	物理ページ番号 (Physical Page Number)
PSZ	エントリ 2	[11:10]	ページサイズ
GROUP	エントリ 2	[9:4]	コンテキストグループ番号
CACHE_LOCK	エントリ 2	[3]	該当ページのキャッシュでのロック
UNCACHE	エントリ 2	[2]	該当ページのキャッシュ不可
BURST	エントリ 2	[1:0]	内部バスアクセス時のバースト転送長

表 5.2: TLB の初期化時の値

フィールド名	初期化時の値
VPN	全 bit 0
LOCK	0 (ロックオフ)
PROTECT	000 (KER_R モード)
SHARE_TH	11111111 (全コンテキスト無効)
PPN	全 bit 0
PSZ	00 (4K Byte)
GROUP	全 bit 0
CACHE_LOCK	0 (ロックオフ)
UNCACHE	0 (無効)
BURST	11 (バーストなし)

## LOCK

TLB ミスが起こると、そのミスを起こした仮想アドレスを変換するための新たな設定を TLB エントリに行う必要がある。全てのエントリがすでに使われていると、いずれかのエントリを選択して設定の入れ換えを行わなければならない。本 MMU では各 TLB エントリへのアクセス情報を記録した LRU 情報を用い、最もアクセスがなされていないエントリを入れ換えの対象とする。

この LOCK フィールドを設定する (1 にする) と、その TLB エントリを LRU 情報を用いた入れ換えの対象から外すことができる。ただし設定を行うエントリを直接指定した場合にはこの LOCK フィールドの設定は無効となる。

また、LOCK フィールドが設定された TLB エントリを使ってアドレス変換を行った場合、その TLB エントリはもっとも最近に参照されたものとして LRU 情報が更新される。

この LOCK フィールドはエントリ 1 に属する。

## PROTECT

ページ単位でのメモリ領域の保護を行うため、この PROTECT フィールドにそのための保護情報を指定する。表 5.3 に指定可能な保護情報の一覧を示す。

尚 Responsive Multithreaded Processor では制限の厳しい順にカーネル・スーパーバイザー・ユーザの 3 つのスレッドの動作モードが規定されているが、モードフィールドとして 2bit が利用可能であるため、本フィールドにはユーザモードよりも更に制限の緩い場合を設定できる。

この PROTECT フィールドはエントリ 1 に属する。

## SHARE\_TH

Responsive Multithreaded Processor は同時に最大 8 個のスレッドが動作するため、TLB エントリのミス率が高くなってしまう。ミス率を少しでも低く抑えるために、コンテキスト毎に TLB エントリに有効情報を持つようにする。Responsive Multithreaded Processor はスレッド ID(32bit) ではなくコンテキスト ID(3bit) を用いてスレッドの実行を制御している。特にコンテキストが有効かどうかは各コンテキストにつき 1bit の情報で与えられるので、TLB エントリにはそれに対応する bit を用意している。この SHARE\_TH フィールドには、各コンテキストの有効情報を保持する。エントリの有効情報は仮想アドレスの比較に用いられるだけではなく、エントリの共有と入れ換えエントリの選択にも用いる。

入れ替えを行う TLB エントリは特に指定がなければ LRU 情報に基づいて選択されるが、その前に SHARE\_TH フィールドを調べて無効なエントリが存在する場合はそれを入れ替えの対象とする。

また、有効であったコンテキストが無効化され、かつそのコンテキストの MMU でのアドレス変換が有効であった場合には、自動的にこの SHARE\_TH フィールドは無効に設定される。

このフィールドはエントリ 1 に属する。

## PPN

この PPN フィールドには、VPN フィールドの仮想ページ番号がマップされている物理ページ番号が保持される。Responsive Multithreaded Processor は物理アドレスに 32bit の信号を用い、最小ページサイズが 4KB であるため、TLB エントリで変換される物理アドレスは表 5.1 に示すように上位 20bit である。PPN フィールドはエントリ 2 に属し、エントリ設定時には設定を行う物理ページ番号を設定データの上位 20bit に指定する。

表 5.3: TLB エントリに指定可能なページ保護情報

保護モード	設定コード	保護の詳細
ALL_RW	111	全モードでの読み出しと書き込みを許可
ALL_R	110	全モードでの読み込み、ユーザモード以上での書き込みを許可
USR_RW	101	ユーザモード以上での読み出しと書き込みを許可
USR_R	100	ユーザモード以上の読み込み、スーパーバイザーモード以上の書き込みを許可
SV_RW	011	スーパーバイザーモード以上での読み出しと書き込みを許可
SV_R	010	スーパーバイザーモード以上の読み込み、カーネルモードでの書き込みを許可
KER_RW	001	カーネルモードでの読み出しと書き込みを許可
KER_R	000	カーネルモードでの読み込みのみを許可

## PSZ

Responsive Multithreaded Processor では、複数ページサイズのサポートをしている。TLB エントリでも複数のページサイズを用いることができるようにしており、表 5.4 に指定可能なページサイズを示す。

表 5.4: TLB エントリに指定可能なページサイズ

ページサイズ	設定コード
4K byte	00
64K byte	01
1M byte	10
16M byte	11

この PSZ フィールドはエントリ 2 に属する。

## GROUP

Responsive Multithreaded Processor ではコンテキスト ID を用いた制御が行なわれるため、特に一度コンテキストキャッシュに退避されたスレッドが実行を再開する場合には、退避前の TLB エントリの設定値は全く使うことができない。これはコンテキスト単位で仮想アドレスを識別しているために、実行スレッドが切り替わる際にはどうしても無効化しなければならないからである。各々のスレッドのアドレスマップは通常独立であるから、エントリの無効化は問題にはならない。しかし共有メモリ領域のエントリでは、退避前までエントリを共有していたスレッドが、実行再開後は全く別のエントリに設定しなければならないことになる。

そこでそのような無駄を省くためにこの GROUP フィールドを用いる。各 TLB エントリはこのフィールドに設定された ID を用いて有効情報の変更が可能になっている (5.2)。そこで共有メモリ領域を持つスレッドは、その領域の TLB エントリに設定されたコンテキストグループ番号を知っていれば、実行を再開したコンテキスト番号をそのコンテキストグループに属する TLB エントリに通知するだけで、容易に TLB エントリの有効化を行なうことができる。

この GROUP フィールドはエントリ 2 に属する。

## CACHE\_LOCK

このフィールドを有効にすることで、該当ページのデータブロックをキャッシュ上にロックすることができる。機能は TLB エントリのロックフィールド (5.1) と同等である。

この CACHE\_LOCK フィールドはエントリ 2 に属し、MMU が無効状態でのデフォルトの値は無効 (ロック不可) となる。

## UNCACHE

この UNCACHE フィールドを有効 (1 と設定) にすると、そのページのブロックはキャッシュされない。キャッシュシステム内にはキャッシュメモリ本体以外にもデータが置かれるバッファがいくつかあるが、このフィールドが有効になっているページのデータは、書き込みデータのマージ機構 (6.1.4) や内部バス要求キューでの複数データヒット機構 (6.1.4)、victim buffer (6.1.3) が無効になる。

このフィールドはエントリ 2 に属し、MMU が無効状態でのデフォルトの値は有効 (キャッシュ不可) となる。

## BURST

アクセスしたいデータがキャッシュミスとなると、内部バスへ要求を出すことになる。このBURST フィールドには、その場合の内部バスに対するバースト読み出しの転送長を指定する。設定可能な転送長を表 5.5 に示す。

表 5.5: TLB エントリで指定可能な内部バスのバースト転送長

転送長	設定コード	転送データ量
無し	11	32byte
2	10	64byte
4	01	128byte
8	00	256byte

このフィールドの値は書き込み要求には適応されない。また I/O など 32bit バスのデータ読み出しにも適応されない。I/O であるかどうかはアドレス空間を用いて識別する。

このフィールドはエントリ 2 に属し、MMU が無効状態でのデフォルトの値はバースト転送無しとなる。

## 5.2 MMU の制御

MMU のコントロールレジスタの一覧を表 5.7 に示す。

各レジスタは通常のアドレス空間や、プロセッシングコアのコントロールレジスタのアドレス空間とは異なる独自のアドレス空間にマッピングされている。そのため MMU のコントロールレジスタへのアクセスは表 5.6 に示す 4 つの専用命令を用いて行う。

表 5.6: MMU のコントロールレジスタアクセス用命令

命令	用途
MFIMM	命令用 MMU のコントロールレジスタの値を読み出す
MTIMM	命令用 MMU のコントロールレジスタに値を設定
MFDMM	データ用 MMU のコントロールレジスタの値を読み出す
MTDMM	データ用 MMU のコントロールレジスタに値を設定

コントロールレジスタの値の設定方法には、設定するデータをそのまま指定するものと、一定の形式に合わせて指定するものがある。後者の一定の形式は MMU のコントロールレジスタの設定のみならず、キャッシュコントローラのコントロールレジスタの設定にも用いられる場合がある (6.1.5)。そこでこの一定の形式のことを共通設定形式 (図 5.1) と呼ぶことにする。共通設定形式では 1 を設定することで該当レジスタの機能を有効化することができる。またコンテキストグループ (5.1) に対してはこの共通設定形式を拡張した独自の形式 (図 5.4) を用いて設定を行う。

コントロールレジスタの設定のタイミングはプロセッサの実行状況によって全く異なるため、実行中のスレッドに対してページ設定以外の設定情報の変更 (MMU のオン・オフやエントリのフラッシュ) を行う場合は注意が必要である。

またコントロールレジスタの多くは設定要求 (書き込み要求) のみを規定しており、そのようなレジスタに対する読み出し要求には返戻値として 0 が返る。

S	select field	1に設定すると該当コンテキストを撰択	
V	value field	1に設定すると有効化、0に設定すると無効化	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
context7		context6		context5		context4		context3		context2		context1		context0	
S	V	S	V	S	V	S	V	S	V	S	V	S	V	S	V

図 5.1: コントロールレジスタの共通設定形式

### MMU\_SPR\_START

MMU\_SPR\_START レジスタは MMU 機能の有効・無効を示す。

Responsive Multithreaded Processor はスレッド毎ではなくコンテキスト毎に制御を行うため、MMU\_SPR\_START レジスタもコンテキスト毎に用意されている。コンテキストの指定は書き込みデータの低位 8bit[7:0] で行う。

またこのレジスタの読み出し要求に対しては、データの低位 8bit[7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される。

設定には図 5.1 に示した共通設定形式を用いる。

### MMU\_SPR\_ALL\_FLUSH

このレジスタに対して書き込み要求を行うと、全ての TLB エントリの設定データとエントリアクセスの LRU 情報を初期化する。書き込むデータに制約はなく、設定を行うと次クロックで自動的にクリアされる。

### MMU\_SPR\_TLB\_FLUSH

このレジスタに指定した番号の TLB エントリのみを初期化する。TLB エントリの指定は書き込みデータの低位 6bit[5:0] で行う。設定を行うと次クロックで自動的にクリアされる。

### MMU\_SPR\_THREAD\_FLUSH

各 TLB エントリにおいてこのレジスタに指定したコンテキストのみを無効化する。設定方法は図 5.1 に示した共通設定形式を用いる。設定を行うと次クロックで自動的にクリアされる。

### MMU\_SPR\_GROUP\_FLUSH

このレジスタに指定したコンテキストグループに属する TLB エントリを無効化する (5.1)。コンテキストグループの指定は書き込みデータの低位 6bit[5:0] で行う。設定を行うと次クロックで自動的にクリアされる。

### MMU\_SPR\_LRU\_FLUSH

このレジスタに書き込み要求を行うと、TLB エントリのアクセスに関する LRU 情報を初期化する。書き込むデータに制約はない。設定を行うと次クロックで自動的にクリアされる。

表 5.7: MMU のコントロールレジスタ一覧

アドレス [7:0]	レジスタ名	設定方法	機能
0x00	MMU_SPR_START	共通形式	アドレス変換の有効・無効
0x04	MMU_SPR_ALL_FLUSH	設定値無し	エントリの無効化と LRU 情報の初期化
0x08	MMU_SPR_TLB_FLUSH	直接指定	指定したエントリを無効化
0x0c	MMU_SPR_THREAD_FLUSH	共通形式	指定したコンテキストを無効化
0x10	MMU_SPR_GROUP_FLUSH	直接指定	指定したグループを全て無効化
0x14	MMU_SPR_LRU_FLUSH	直接指定	LRU 情報を初期化
0x18	MMU_SPR_MAX_LOCK	直接指定	エントリをロックできる最大数
0x1c	MMU_SPR_ENTRY1	直接指定	TLB エントリのエントリ 1 を設定
0x20	MMU_SPR_ENTRY2	直接指定	TLB エントリのエントリ 2 を設定
0x24	MMU_SPR_ENTRY_INDEX	設定値無し	指定した TLB エントリを設定
0x28	MMU_SPR_ENTRY_LRU	設定値無し	LRU 情報によりエントリを設定
0x2c	MMU_SPR_GROUP	特殊形式	指定したグループのエントリの有効化・無効化
0x30	MMU_SPR_EXP_ADDR	設定値無し	例外が発生したアドレス
0x34	MMU_SPR_EXP_LOG	設定値無し	発生した例外の詳細情報
0x38	MMU_READ_TLB_ADDR	設定値無し	TLB の読みたいエントリのアドレス
0x3c	MMU_READ_TLB_DATA	設定値無し	TLB の読みたいエントリのデータ

### MMU\_SPR\_MAX\_LOCK

このレジスタには TLB エントリをロックし、LRU 情報によるエントリの入れ換え対象計算から外すことのできるエントリ数を設定する。既定値は 16 エントリ、最小値は 0 エントリ、最大値は 63 エントリであり、このレジスタの値以上のエントリをロックすることは基本的にできない。しかしロックエントリ数の計算に 1 クロック要するため、エントリの設定命令が 2 クロック連続するとこのレジスタの値を越えてロックが設定される可能性がある。全 64 エントリがロックされてしまった場合、ページフォルト発生時にページテーブルを設定できなくなってしまう。そのため、64 エントリがロックされると例外を発生させる (5.3)。MMU\_SPR\_ENTRY\_LRU でエントリ 1 の LOCK フィールドを 1 にした TLB エントリをセットする時に MMU\_SPR\_MAX\_LOCK の値以上にセットしようとした場合、ロックはされないが TLB エントリのほかの内容はセットされる。例えば、MMU\_SPR\_MAX\_LOCK が 16 で既にロックされている TLB エントリも 16 個存在する時に MMU\_SPR\_ENTRY\_LRU でロックしたエントリをセットする場合、LOCK フィールドを 0 にして TLB エントリをセットすることになる。

このレジスタの読み出し要求に対しては、データの下位 6bit[5:0] に現在の設定値を格納する。

### MMU\_SPR\_ENTRY1

各 TLB エントリが持つエントリフィールドのうち、このレジスタには仮想アドレス、エントリのロック指定、ページ保護情報、エントリ共有情報を設定する。設定形式を図 5.2 に示す。

### MMU\_SPR\_ENTRY2

各 TLB エントリが持つエントリフィールドのうち、このレジスタには物理アドレス、ページサイズ、コンテキストグループ、該当ページのキャッシュでのロックの可否、該当ページのキャッシュの可否、該当

仮想ページ番号 : VPN    ページ保護情報 : PRO  
 エントリロック : LCK    共有情報 : SHR

31:12	11	10:8	7:0
VPN	LCK	PRO	SHR

図 5.2: ENTRY1 の設定形式

ページのバスアクセス時のバースト転送長を指定する．設定形式を図 5.3 に示す．

物理ページ番号 : PPN    キャッシュロック : CLC  
 ページサイズ : PSZ    キャッシュ不可 : UNC  
 コンテキストグループ : GRP    バースト転送長 : BRT

31:12	11:10	9:4	3	2	1:0
PPN	PSZ	GRP	CLC	UNC	BRT

図 5.3: ENTRY2 の設定形式

### MMU\_SPR\_ENTRY\_INDEX

このレジスタに書き込み要求を行い TLB エントリ番号を指定することで、事前に設定しておいた MMU\_SPR\_ENTRY1 フィールドと MMU\_SPR\_ENTRY2 フィールドの値を、その指定された TLB エントリに設定する．TLB エントリの指定は書き込むデータの低位 6bit[5:0] で行う．

### MMU\_SPR\_ENTRY\_LRU

このレジスタに書き込み要求を行うことで、事前に設定しておいた MMU\_SPR\_ENTRY1 フィールドと MMU\_SPR\_ENTRY2 フィールドの値を、LRU 情報を元にして最もアクセスがなされていない TLB エントリに設定する．書き込むデータに制約はない．

### MMU\_SPR\_GROUP

このレジスタに書き込み要求を行うことで、指定したコンテキストグループに所属する TLB エントリの、指定したコンテキストの有効化・無効化を行う (5.1)．コンテキストグループとコンテキストの指定形式を図 5.4 に示す．

<b>S</b>	select field	1に設定すると該当コンテキストを選択
<b>V</b>	value field	1に設定すると有効化、0に設定すると無効化

21		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
コンテキストグループ番号		context7		context6		context5		context4		context3		context2		context1		context0		
		S	V	S	V	S	V	S	V	S	V	S	V	S	V	S	V	

図 5.4: コンテキストグループの設定形式

### MMU\_SPR\_EXP\_ADDR

このレジスタはアドレス変換において該当する TLB エントリが存在しなかった場合に、その仮想アドレスを保持する。アドレスはコンテキスト毎に保持され、その値を読み出すにはデータの低位 3bit[2:0] にコンテキスト番号を指定する。

このレジスタに対する書き込み要求は、実行したコンテキストに対応するレジスタの値がクリアされるだけである。

### MMU\_SPR\_EXP\_LOG

このレジスタにはページ保護の違反が確認された時にその違反コード (表 5.8) が保持される。

表 5.8: MMU のページ保護違反コード

コード名	違反コード	違反内容
MMU_EXP_NONE	000	違反無し
MMU_EXP_PRO_ALL_R	001	全モードでの書き込み制限違反
MMU_EXP_PRO_USR_RW	010	ユーザモード以上に限定されたページへのアクセス違反
MMU_EXP_PRO_USR_R	011	ユーザモード以上に限定されたページへの書き込み違反
MMU_EXP_PRO_SPV_RW	100	スーパーバイザーモード以上に限定されたページへのアクセス違反
MMU_EXP_PRO_SPV_R	101	スーパーバイザーモード以上に限定されたページへの書き込み違反
MMU_EXP_PRO KER_RW	110	カーネルモード以上に限定されたページへのアクセス違反
MMU_EXP_PRO KER_R	111	カーネルモード以上に限定されたページへの書き込み違反

このレジスタの読み出し要求に対しては、4bit 目に違反発生の有無が (1 で保護違反発生)、低位 3bit[2:0] に違反コードが格納される。

またこのレジスタに対する書き込み要求は、実行したコンテキストに対応するレジスタの値がクリアされるだけである。

### MMU\_READ\_TLB\_ADDR

### MMU\_READ\_TLB\_DATA

この2つのレジスタを用いて TLB の中身を読むことができる。まず読みたい TLB のエントリを MMU\_READ\_TLB\_ADDR レジスタに書き込み、次に MMU\_READ\_TLB\_DATA レジスタを読み出す。MMU\_READ\_TLB\_ADDR レ

ジスタに書き込む値は 6:1 bit 目にエントリの番号, 0bit 目に Entry1 ならば 0, Entry2 ならば 1 を指定する。

### 5.3 MMU が発生させる例外

本 MMU が発生させる例外を表 5.9 に示す。

命令用 MMU, データ用 MMU 共に発生させる例外の種類は同じであるが, 命令とデータの区別をつけて例外を扱う。

また命令要求が発生させた例外とデータ要求が発生させた例外とではプロセッシングコアでの扱いが異なるが, MMU に対して設定を行うのはデータ要求であるため, 命令用 MMU で発生した設定用の例外 (表 5.9 の例外の種類の設定) はデータ用 MMU の例外コードと一緒に扱われる。

表 5.9: MMU の発生させる例外

例外名	例外の種類	命令用 MMU のコード	データ用 MMU のコード
エントリミス	要求	0x3	0x8
ページ保護違反	要求	0x5	0xa

#### エントリミス (TLB ミス)

命令要求やデータ要求によって指定された仮想アドレスとコンテキスト ID が, どのエントリの設定値とも一致しなかった場合に発生する。例外を起こした仮想アドレスをコントロールレジスタに保持するが (5.2), MMU の状態は変化しない。例外発生後もアドレス変換や TLB エントリの設定は通常通り可能である。例外発生時, MMU\_SPR\_EXP\_LOG の値は 0(違反なし) に設定される。

尚命令用 MMU で本例外が発生した場合, 命令フェッチユニットの仕様により命令を返さなければ例外処理に進むことができないため, フェッチ命令幅の全てを No-op コードとして返す。

#### ページ保護違反

命令要求やデータ要求によって指定された仮想アドレスとコンテキスト ID が一致した TLB エントリにおいて, その要求が設定された保護情報に反する場合に発生する。コントロールレジスタには例外コード (表 5.8) が格納されるが, MMU の状態は変化しない。

命令用 MMU でこの例外が発生した場合, エントリミスと同様に命令フェッチ幅の全てを No-op として返す。

# 6

## CACHE

---

### 6.1 キャッシュシステム

#### 6.1.1 概要

Responsive Multithreaded Processor のキャッシュシステムの特徴を以下に示す．またモジュール構成を図 6.1 に示す．本節ではこれらキャッシュシステムを構成する各要素について述べる．

- 32 KB 8-way set-associative 方式
- ブロックサイズ，ラインサイズともに 32byte
- Look Through
- ノンブロッキング
- 下位メモリとのデータ一貫性の維持はライトバック方式
- 書き込み要求ミスの処理はライトアロケート方式
- キャッシュポートは 1 ポート
- 物理タグでデータを保持
- 転送ブロック数が可変
- キャッシュのロックが可能
- 3 サイクルのアクセス遅延
- マルチタグ，シングルデータ方式
- 16 エントリの victim buffer
- 最大 16 個のキャッシュミスと同時に保持
- 入れ換えを行うブロックの選択方法は LRU と優先度の 2 通り
- バス待ちキューでの優先度による要求の追い越しが可能

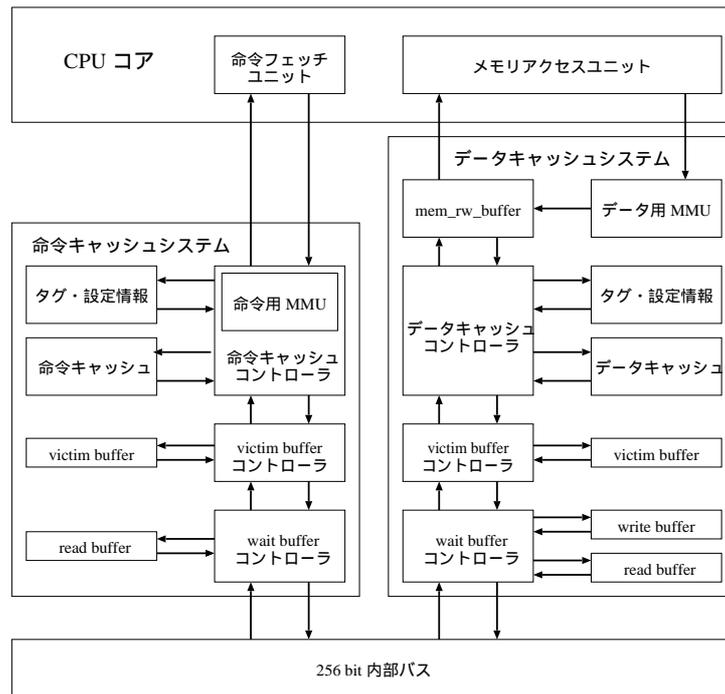


図 6.1: キャッシュシステムのモジュール構成

### 6.1.2 キャッシュ制御

キャッシュの制御は図 6.1 の命令キャッシュコントローラ、データキャッシュコントローラで行う。

キャッシュ要求がキャッシュミスを起こした場合には、先に victim buffer で保持されているデータと比較され (6.1.3)、そこでも該当データを発見できなければ wait buffer から内部バスへアクセスを行う (6.1.4)。

#### キャッシュでのデータ一貫性の維持

命令、データの両キャッシュコントローラは、内部バスで発生する書き込み要求を常に監視する。そしてもしキャッシュしているデータが書き込みを受けた場合にそのデータを無効化する。

### 6.1.3 victim buffer

victim buffer では、キャッシュブロックの入れ換えに伴いキャッシュメモリを追い出されたデータを、full associative 方式でエントリに保持する。そしてキャッシュミスを起こした要求のアドレスを現在保持しているデータのタグと比較し、もし一致するデータがあれば該当データをキャッシュへと送り込む。

### 6.1.4 wait buffer

#### 概要

wait buffer は内部バス要求キュー、read buffer とその管理機構からなるキャッシュコントローラの内部バスインタフェースであり、命令用とデータ用のそれぞれに分かれる。データキャッシュ用の wait buffer には更に write buffer とその管理機構が付随する。

## 内部バス要求

内部バス要求キューと write buffer は 16 エントリから成り、victim buffer から送られてくる様々な要求を順にエントリに格納して行く。

### 内部バス要求キューの要求順位入れ換え機能

内部バス要求キューの要求順位を入れ替える機構がある。その方法は読み出し要求を書き込み要求よりも優先して行う方法と、優先度が高いコンテキストの要求を優先して行う方法の 2 種類である。ただし、ライトアロケート方式を用いているため通常の書き込み要求は読み出し要求と同じくデータの読み出しを行うため、追い抜きの対象は write back 要求になっている。

### 書き込み要求のマージ機能

通常書き込み要求のデータは 1 byte の文字型や 4 byte の整数型、8 byte の倍精度浮動小数点型のデータであるため、1 キャッシュラインのデータ幅である 32 byte に対しては小さい。よって同じキャッシュラインに対するデータの書き込みは 1 つのエントリにまとめることができるようにしている。

ただし I/O への書き込み要求であった場合には、データのマージは行わない。

## 6.1.5 キャッシュのコントロールレジスタ

キャッシュのコントロールレジスタの一覧を表 6.1 に示す。これらのレジスタはプロセッシングコアのコントロールレジスタと同じアドレス空間にマッピングされており、それらと同じ命令 (表 6.2) を用いてアクセスする。尚これらのレジスタの設定方法は TLB のコントロールレジスタの設定 (5.2) と同じようにデータを直接指定するか、図 5.1 に示した共通設定形式を用いる。

表 6.1: キャッシュのコントロールレジスタ

レジスタ名	設定方法	機能	命令用アドレス [7:0]	データ用アドレス [7:0]
ON	共通形式	キャッシュの有効・無効	0x80	0x86
REP_MODE	直接指定	入れ換え方法の指定	0x81	0x87
ACC_SCHE	直接指定	要求の追い越し指定	0x82	0x88
LOCK	共通形式	ロックの有効・無効	0x83	0x89
RESET	直接指定	キャッシュのリセット	0x84	0x8a
FLUSH	共通形式	write back の指定	無し	0x8b
ALL_FLUSH	直接指定	全て write back	無し	0x8c

表 6.2: コントロールレジスタをアクセスする命令

命令	用途
MFC0	コントロールレジスタの値を読み出す
MTC0	コントロールレジスタに値を設定

## ON

このレジスタを設定することで、コンテキスト毎にキャッシュの有効・無効を指定できる。

初期状態の設定値は全コンテキスト共に無効になっており、またコンテキストが無効化されるとそのコンテキストに対応するフィールドは自動的に無効となる。

このレジスタの設定には共通設定形式(図 5.1)を用いる。またこのレジスタの読み出し要求に対しては、データの下位 8 bit [7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される。

## REP\_MODE

このレジスタにはキャッシュブロックの入れ換え方法を指定する。0 を設定すると LRU 情報に基づく方法、1 を設定するとオーナーコンテキストの優先度に基づく方法となる。

設定はデータの 1 bit 目で行われる。このレジスタの初期値は LRU を用いた方法である。またこのレジスタの読み出し要求に対しては、データの最下位 bit [0] に設定値が格納される。

## ACC\_SCHE

このレジスタには、6.1.4 で述べた優先度に従った内部バス要求キューの要求入れ換え機能の有効、無効を指定する。このレジスタに 1 を設定することでその機能を有効にできる。

設定は REP\_MODE レジスタと同様にデータの 1 bit 目で行われ、初期値は無効である。またこのレジスタの読み出し要求に対しては、データの最下位 bit [0] に設定値が格納される。

## LOCK

このレジスタには 5.1 で述べたコンテキスト毎のキャッシュロックの有効、無効を指定する。このレジスタと TLB エントリの CACHE\_LOCK フィールドが設定されることで、該当コンテキストがキャッシュデータをロックすることが可能になる。どちらか一方の設定だけではキャッシュロックを行うことはできない。

一旦キャッシュをロックしてしまうと、そのコンテキストが有効である限りそのデータがキャッシュから追い出されることはない。これはキャッシュの入れ換えを優先度に従う方式で行っていても同様のため、低優先度のスレッドに対するロック許可や、ロック許可状態での高優先度のスレッドの実行を行う場合には注意が必要である。

初期状態の設定値は全コンテキスト共に無効になっており、またコンテキストが無効化されるとそのコンテキストに対応するフィールドは自動的に無効となる。

このレジスタの設定には共通設定形式(図 5.1)を用いる。またこのレジスタの読み出し要求に対しては、データの下位 8 bit [7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される。

## RESET

このレジスタに 1 を設定することで、キャッシュと victim buffer のエントリを全て無効化することができる。ただしデータ用のエントリに収められているデータでもその書き戻しは行わない。

このレジスタの読み出し要求に対しては、データの最下位 bit [0] に現在の状態を格納する。1 が読み出された場合は、現在キャッシュの無効化が行われていることを意味する。

**FLUSH (データキャッシュコントローラのみ)**

このレジスタを有効に設定することで、キャッシュデータと victim buffer のデータの下位メモリへの書き戻しを開始する。書き戻しが終了すると、自動的に無効状態になる。設定には共通設定形式 (図 5.1) を用い、コンテキスト単位で書き戻し要求を指定できるが、無効状態のコンテキストに対する指定でも書き戻しを行う。

またこのレジスタの読み出し要求に対しては、データの下位 8 bit [7:0] の上位から順番にコンテキスト番号 7 からコンテキスト番号 0 までの設定値が格納される。

**ALL\_FLUSH**

このレジスタに書き込み要求を行うと、それだけで全コンテキストの書き戻しを開始する。指定するデータに制限はない。



## 7

## システムレジスタ

システムレジスタはMFC0, MTC0 命令でアクセスする。アクセスしたいレジスタ番号を入れたレジスタを rd に指定する。

### 7.1 レジスタマップ

offset	31	24 23	16 15	8 7	0
0x00	Status Register ( Thread0 )				
0x01	Status Register ( Thread1 )				
0x02	Status Register ( Thread2 )				
0x03	Status Register ( Thread3 )				
0x04	Status Register ( Thread4 )				
0x05	Status Register ( Thread5 )				
0x06	Status Register ( Thread6 )				
0x07	Status Register ( Thread7 )				
0x08	Thread Table Register ( Thread0 )				
0x09	Thread Table Register ( Thread1 )				
0x0a	Thread Table Register ( Thread2 )				
0x0b	Thread Table Register ( Thread3 )				
0x0c	Thread Table Register ( Thread4 )				
0x0d	Thread Table Register ( Thread5 )				
0x0e	Thread Table Register ( Thread6 )				
0x0f	Thread Table Register ( Thread7 )				
0x10	Thread ID Register ( Thread0 )				
0x11	Thread ID Register ( Thread1 )				
0x12	Thread ID Register ( Thread2 )				
0x13	Thread ID Register ( Thread3 )				
0x14	Thread ID Register ( Thread4 )				
0x15	Thread ID Register ( Thread5 )				
0x16	Thread ID Register ( Thread6 )				
0x17	Thread ID Register ( Thread7 )				

offset	31	24 23	16 15	8 7	0
0x18	Instruction Count Register ( Thread0 )				
0x19	Instruction Count Register ( Thread1 )				
0x1a	Instruction Count Register ( Thread2 )				
0x1b	Instruction Count Register ( Thread3 )				
0x1c	Instruction Count Register ( Thread4 )				
0x1d	Instruction Count Register ( Thread5 )				
0x1e	Instruction Count Register ( Thread6 )				
0x1f	Instruction Count Register ( Thread7 )				
0x20	Count Register ( Thread0 )				
0x21	Count Register ( Thread1 )				
0x22	Count Register ( Thread2 )				
0x23	Count Register ( Thread3 )				
0x24	Count Register ( Thread4 )				
0x25	Count Register ( Thread5 )				
0x26	Count Register ( Thread6 )				
0x27	Count Register ( Thread7 )				
0x28	Compare Register ( Thread0 )				
0x29	Compare Register ( Thread1 )				
0x2a	Compare Register ( Thread2 )				
0x2b	Compare Register ( Thread3 )				
0x2c	Compare Register ( Thread4 )				
0x2d	Compare Register ( Thread5 )				
0x2e	Compare Register ( Thread6 )				
0x2f	Compare Register ( Thread7 )				
0x30	Floating-Point Control Register ( Thread0 )				
0x31	Floating-Point Control Register ( Thread1 )				
0x32	Floating-Point Control Register ( Thread2 )				
0x33	Floating-Point Control Register ( Thread3 )				
0x34	Floating-Point Control Register ( Thread4 )				
0x35	Floating-Point Control Register ( Thread5 )				
0x36	Floating-Point Control Register ( Thread6 )				
0x37	Floating-Point Control Register ( Thread7 )				
0x38	Issue Mode Register				
0x39	CPU Count Register (Low)				
0x3a	CPU Count Register (High)				
0x3b ~ 0x47	MMU Register				
0x48	Exception PC Register ( Thread0 )				
0x49	Exception PC Register ( Thread1 )				
0x4a	Exception PC Register ( Thread2 )				
0x4b	Exception PC Register ( Thread3 )				
0x4c	Exception PC Register ( Thread4 )				
0x4d	Exception PC Register ( Thread5 )				
0x4e	Exception PC Register ( Thread6 )				
0x4f	Exception PC Register ( Thread7 )				
0x50	Exception Cause Register ( Thread0 )				
0x51	Exception Cause Register ( Thread1 )				
0x52	Exception Cause Register ( Thread2 )				
0x53	Exception Cause Register ( Thread3 )				
0x54	Exception Cause Register ( Thread4 )				
0x55	Exception Cause Register ( Thread5 )				
0x56	Exception Cause Register ( Thread6 )				
0x57	Exception Cause Register ( Thread7 )				

offset	31	24 23	16 15	8 7	0
0x58	Interruption Wait Register ( Thread0 )				
0x59	Interruption Wait Register ( Thread1 )				
0x5a	Interruption Wait Register ( Thread2 )				
0x5b	Interruption Wait Register ( Thread3 )				
0x5c	Interruption Wait Register ( Thread4 )				
0x5d	Interruption Wait Register ( Thread5 )				
0x5e	Interruption Wait Register ( Thread6 )				
0x5f	Interruption Wait Register ( Thread7 )				
0x60	External Interruption Level Register ( Thread0 )				
0x61	External Interruption Level Register ( Thread1 )				
0x62	External Interruption Level Register ( Thread2 )				
0x63	External Interruption Level Register ( Thread3 )				
0x64	External Interruption Level Register ( Thread4 )				
0x65	External Interruption Level Register ( Thread5 )				
0x66	External Interruption Level Register ( Thread6 )				
0x67	External Interruption Level Register ( Thread7 )				
0x68 ~ 0x69	-				
0x6a	Exception Base Address Register				
0x6b ~ 0x6f	-				
0x70 ~ 0x73	Event Link In Register				
0x74 ~ 0x77	Event Link Out Register				
0x80 ~ 0x84	Instruction Cache Control Register				
0x86 ~ 0x8c	Data Cache Control Register				
0x8e	ROM Status				
0x8f	EXT Status				
0x90	Multiplexer Arbitor Mode 256bit Bus				
0x91	Multiplexer Arbitor Mode 32bit Bus				
0x92	Multiplexer Watchdog Timer 256bit Bus Enable				
0x93	Multiplexer Watchdog Timer 256bit Bus Mode				
0x94	Multiplexer Watchdog Timer 256bit Bus Reset				
0x95	Multiplexer Watchdog Timer 256bit Bus Count				
0x96	Multiplexer Error Handler State 256bit Bus				
0x97	Multiplexer Error Handler State 32bit Bus				
0x98	Multiplexer Error Handler Instruction Cache				
0x99	Multiplexer Error Handler Data Cache				
0x9a	Multiplexer Error Handler DMAC0				
0x9b	Multiplexer Error Handler DMAC1				
0x9c	Multiplexer Error Handler DMAC2				
0x9d	Multiplexer Error Handler PCI				
0x9e	Multiplexer Error Handler Bus Interface Unit				
0x9f	Multiplexer Error Handler MDMAC256				
0xa0 ~ 0xb8	Address Decoder Control Register				
0xb9	Multiplexer Watchdog Timer 32bit Bus Enable				
0xba	Multiplexer Watchdog Timer 32bit Bus Mode				
0xbb	Multiplexer Watchdog Timer 32bit Bus Reset				
0xbc	Multiplexer Watchdog Timer 32bit Bus Count				
0xbd	Multiplexer Error Handler MDMAC32				
0xe0	Own Status Register				
0xe1	Own Thread Table Register				
0xe2	Own Thread ID Register				
0xe3	Own Instruction Count Register				
0xe4	Own Count Register				

offset	31	24 23	16 15	8 7	0
0xe5	Own Compare Register				
0xe6	Own Floating-Point Control Register				
0xe7	Own Bad Virtual Address Register				
0xe8	Own Exception PC Register				
0xe9	Own Exception Cause Register				
0xea	Own Interruption Wait Register				
0xeb	Own External Interruption Level Register				

### 7.1.1 Status Register

アドレス: 0x00 ~ 0x07 (各スレッド毎)

スレッド毎の状態を示す。リセット後は 0x00000000 に初期化される。

31	25 24 23 22 21	12 11	8 7 6 5 4 3 2 1 0
-	TS PE EV	-	IM   -   EB   C   MO   -   EL   IE

bit 名	機能
TS	Timer Start 1: タイマーをスタートする . 0: タイマーをストップする .
PE	Period 1: Timer Interrupt を周期的に発生する 0: One Shot
EV	Exception Vector Location . 1: Bootstrap ... 例外発生時にブート時用の例外ベクタへ制御が移る . 0: Normal ... 例外発生時に通常の例外ベクタへ制御が移る .
IM	Interruption Mask . 1 をセットすると , 対応する種類の割り込みをマスクする . 11: Tiemr Interruption 10: Hardware Interruption 9: Software Interruption1 8: Software Interruption0
EBAE (EB)	Exception Base Address Enable 1: Variable ... TBA(Table Base Address) を基準とした番地の例外ベクタを使用する . 0: Fixed ... 固定番地の例外ベクタを使用する .
CRAM (C)	Control Register Access Mode 0: Precise ... 直前の命令がコミットされるまで制御レジスタに対するアクセス命令の発行を待たせる .
MO	Mode Bit 00: Kernel Mode 01: Supervisor Mode 10: User Mode
EL	Exception Level . 例外が発生すると 1 にセットされる . ERET 命令で 0 にセットされる .
IE	Interruption Mode 0: 全ての割り込みが無効 1: 全ての割り込みが有効

### 7.1.2 Thread Table Register

アドレス: 0x08 ~ 0x0f (各スレッド毎)

スレッドの状態を示す . 基本的にスレッド制御命令によって変更を行う . 読み書き可能であるが , 強制的に書き込みを行った場合の動作は保証しない . 0x08 意外は 0x00000000 に初期化される .

31	14 13 12	9 8 7	0
-	E	STATE	K PRIOR

bit 名	機能
E	Thread Enable 0: そのコンテキストにアクティブスレッドが割り当てられていない。 1: そのコンテキストにアクティブスレッドが割り当てられている。
STATE	Thread State 0000: Invalid 0001: Run 0010: Ready 0011: Not Ready 0100: Backup Now 0101: Restore Now 0110: Backup Wait 0111: Restore Wait 1000: Copy or Swap Now 1001: Stop Wait
KEEP (K)	Keep Active Thread 0: 通常 1: スレッドをコンテキストキャッシュに退避する命令を無効にする。
PRIOR	Thread Priority . 256 Level .

### 7.1.3 Thread ID Register

アドレス: 0x10 ~ 0x17 (スレッド毎)



bit 名	機能
Thread ID	スレッドに対するアクセスの際のスレッドの指定に用いる。

### 7.1.4 Instruction Counter Register

アドレス: 0x18 ~ 0x1f (スレッド毎)



bit 名	機能
Instruction Counter	各スレッドが生成されてからコミットした命令の総数をカウントする。

### 7.1.5 Count Register

アドレス: 0x20 ~ 0x27 (スレッド毎)

31	Count	0
----	-------	---

bit 名	機能
Count	毎クロックカウントアップされるカウンタ。Compare Register と等しくなると 0 にクリアされる。

### 7.1.6 Compare Register

アドレス: 0x28 ~ 0x2f (スレッド毎)

31	Compare	0
----	---------	---

bit 名	機能
Compare	このレジスタに 0 以外の値がセットされており、かつ Count Register の値がこのレジスタの値と等しくなった時、タイマ割り込みを発生する。タイマ割り込みは Status Register の IM フィールドと IE フィールドで有効または無効に設定される。

### 7.1.7 Floating-Point Control Register

アドレス: 0x30 ~ 0x37 (スレッド毎)

31	-	6 5 4 3	0
		RND	EM



bit 名	機能
Sub Assign0, 1, 2, 3 (SA0, 1, 2, 3)	発行スロットにスレッドを割り当てる発行方式 (TH_ASSIGN) で, SUB_POLICY フィールドが SUB_FIX に設定されている状態で, スロット毎のメインで割り当てられているスレッドから命令を発行できない場合にこのフィールドで設定されたスレッドから命令を発行する .
Main Assign0, 1, 2, 3 (MA0, 1, 2, 3)	発行スロットにスレッドを割り当てる発行方式 (TH_ASSIGN) で, スロット毎にメインで割り当てるスレッドを指定する .
Sub Policy (SP)	発行命令選択ポリシーのサブポリシーを設定する . <ul style="list-style-type: none"> <li>● 1INST_1TH <p>00: NORMAL</p> <p>01: PRED_STOP ... 次に発行すべき命令が分岐予測の結果として発行される命令であり, キャンセルされる可能性がある場合はそのスレッドの優先度を低下させる .</p> <p>10: MINST_STOP ... あるスレッドのリオーダバッファの半数以上のエントリが埋まっている場合はそのスレッドの優先度を低下させる .</p> </li> <li>● TH_ASSIGN <p>00: SUB_PRIOR ... スロットにメインで割り当てられているスレッドに発行できる命令がない場合は, スロットに割り当てられていないスレッドの中で最も優先度の高いスレッドから命令を発行する .</p> <p>01: SUB_FIX ... スロットにメインで割り当てられているスレッドに発行できる命令がない場合は, スロットに対しサブで割り当てられているスレッドから命令を発行する . サブのスレッドにも発行できる命令がない場合は, 空きスロットとなる .</p> </li> </ul>
Policy (PO)	発行選択ポリシーを設定する . <p>00: 1INST_1TH ... 毎クロックサイクル, 1 スレッドから最大 1 命令だけを発行可能とするポリシー . 4 スレッド以上のスレッドが実行されていないと, 発行スロットに空きができてしまうことになる .</p> <p>01: HIGHEST_FAST ... 毎クロックサイクル, 最高優先度のスレッドから発行できるだけの命令を発行し, 余った発行スロットは次に高い優先度を持つスレッドに割り当てる . さらに余った場合は 3 番目に高い優先度を持つスレッドでも同様に行う .</p> <p>10: TH_ASSIGN ... 発行スロットごとに特定のスレッドを割り当てて, そのスレッドから命令を発行できない場合のみ他のスレッドの命令を発行する .</p>

### 7.1.9 CPU Count Register

アドレス: 0x39, 0x3a

31	CPU Count	0
----	-----------	---

bit 名	機能
CPU Count	64bit カウンタ。リセット時から毎クロック 1 ずつカウントアップしていく。0x39 が下位 32bit, 0x3a が上位 32bit を示す。

### 7.1.10 MMU Register

アドレス: 0x3b ~ 0x47

MMU 関連の設定レジスタ。

### 7.1.11 Exception PC Register

アドレス: 0x48 ~ 0x4f (スレッド毎)

31	Exception PC	0
----	--------------	---

bit 名	機能
Exception PC	例外を生じた命令, もしくは例外が発生した時点で最後にコミットされた PC を保持するレジスタ。ERET 命令で例外処理からの戻り番地として参照する。

### 7.1.12 Exception Cause Register

アドレス: 0x50 ~ 0x57 (スレッド毎)

発生した例外の情報を保持するレジスタ。0x00000000 に初期化される。

31	30	17	16	12	11	10	9	8	7	6	2	1	0
D	-	HIRL	TI	HI	S1	S0	-	CODE	-	-	-	-	-

bit 名	機能
Delay Bit (D)	例外が発生した命令が Delay Slot の命令である場合に 1 がセットされる .
Hardware Interruption Level (HIRL)	外部割込みのレベル .
TI	Timer Interruption Pending
HI	Hardware Interruption Pending
S1	Software Interruption 1 Pending
S0	Software Interruption 0 Pending
Exception Code (CODE)	最後に発生した例外のコードを保持する .

### 7.1.13 Interruption Wait Register (スレッド毎)

アドレス: 0x58 ~ 0x5f

31	Interruption Wait	0
----	-------------------	---

bit 名	機能
Interruption Wait	各ビットが割り込みレベル (IRL) に対応している . ビットが 1 ならそのスレッドは対応する IRL の外部割込みを受け付ける .

### 7.1.14 External Interruption Level Register (スレッド毎)

アドレス: 0x60 ~ 0x67

31	External Interruption Level	0
----	-----------------------------	---

bit 名	機能
External Interruption Level	最後に IRC から入力された外部割り込みの IRL を保持する .

### 7.1.15 Interruption Pending Register

アドレス: 0x68

現在 (多分) 使用していない。

### 7.1.16 Interruption Clear Register

アドレス: 0x69

現在 (多分) 使用していない。

### 7.1.17 Exception Base Address Register

アドレス: 0x6a



bit 名	機能
Exception Base Address	例外ベクタのベースアドレスを保持する。Status Register の EBAE ビットを 1 に設定すると、例外発生時に EBA に例外の内容に従ったオフセットを加えた番地に制御が移る。

### 7.1.18 Event Link In Register

アドレス: 0x70 ~ 0x73

### 7.1.19 Event Link Out Register

アドレス: 0x74 ~ 0x77

### 7.1.20 Instruction Cache Control Register

アドレス: 0x80 ~ 0x84

6章 (CACHE) のキャッシュのコントローレジスタを参照。

### 7.1.21 Data Cache Control Register

アドレス: 0x86 ~ 0x8c

6章 (CACHE) のキャッシュのコントローレジスタを参照。

## 7.1.22 ROM Status

アドレス: 0x8e



bit 名	機能
Uncache (UC)	1 を指定すると ROM アクセス時に 32bit I/O バスに Uncache 信号を生成する (現在は使用されていない)。Read 時は反転した値が読み出される。
Count	ROM アクセス時の Auto Ready を返すカウントを指定する。

## 7.1.23 EXT Status

アドレス: 0x8f



bit 名	機能
Uncache (UC)	1 を指定すると外部バスアクセス時に 32bit I/O バスに Uncache 信号を生成する (現在は使用されていない)。Read 時は反転した値が読み出される。
Count	外部バスアクセス時の Auto Ready を返すカウントを指定する。

## 7.1.24 Multiplexer Arbitor Mode 256bit Bus

アドレス: 0x90



bit 名	機能
Mode (MO)	バスアービトレーションのモードを指定する。0 を指定すると固定優先度、1 を指定するとラウンドロビン方式でバスマスタにバス権を与える。

### 7.1.25 Multiplexer Arbitor Mode 32bit Bus

アドレス: 0x91



bit 名	機能
Mode (MO)	バスアービトレーションのモードを指定する。0 を指定すると固定優先度、1 を指定するとラウンドロビン方式でバスマスタにバス権を与える。

### 7.1.26 Multiplexer Watchdog Timer 256bit Bus Enable

アドレス: 0x92



bit 名	機能
Mode (MO)	バスアービトレーションのモードを指定する。0 を指定すると固定優先度、1 を指定するとラウンドロビン方式でバスマスタにバス権を与える。

### 7.1.27 Multiplexer Watchdog Timer 256bit Bus Mode

アドレス: 0x93  
使用していない。

### 7.1.28 Multiplexer Watchdog Timer 256bit Bus Reset

アドレス: 0x94

### 7.1.29 Multiplexer Watchdog Timer 256bit Bus Count

アドレス: 0x95

### 7.1.30 Multiplexer Error Handler State 256bit Bus

アドレス: 0x96

**7.1.31 Multiplexer Error Handler State 32bit Bus**

アドレス: 0x97

**7.1.32 Multiplexer Error Handler Instruction Cache**

アドレス: 0x98

**7.1.33 Multiplexer Error Handler Data Cache**

アドレス: 0x99

**7.1.34 Multiplexer Error Handler DMAC0**

アドレス: 0x9a

**7.1.35 Multiplexer Error Handler DMAC1**

アドレス: 0x9b

**7.1.36 Multiplexer Error Handler DMAC2**

アドレス: 0x9c

**7.1.37 Multiplexer Error Handler PCI**

アドレス: 0x9d

**7.1.38 Multiplexer Error Handler Bus Interface Unit**

アドレス: 0x9e

**7.1.39 Multiplexer Error Handler MDMAC256**

アドレス: 0x9f

**7.1.40 Address Decoder Control Register**

アドレス: 0xa0 ~ 0xb8

4章(アドレスデコーダ)のアドレスマップを参照。

**7.1.41 Multiplexer Watchdog Timer 32bit Bus Enable**

アドレス: 0xb9

**7.1.42 Multiplexer Watchdog Timer 32bit Bus Mode**

アドレス: 0xba

**7.1.43 Multiplexer Watchdog Timer 32bit Bus Reset**

アドレス: 0xbb

**7.1.44 Multiplexer Watchdog Timer 32bit Bus Count**

アドレス: 0xbc

**7.1.45 Multiplexer Error Handler MDMAC32**

アドレス: 0xbd

**7.1.46 Own Status Register**

アドレス: 0xe0

**7.1.47 Own Thread Table Register**

アドレス: 0xe1

**7.1.48 Own Thread ID Register**

アドレス: 0xe2

**7.1.49 Own Instruction Count Register**

アドレス: 0xe3

**7.1.50 Own Count Register**

アドレス: 0xe4

**7.1.51 Own Compare Register**

アドレス: 0xe5

**7.1.52 Own Floating-Point Control Register**

アドレス: 0xe6

**7.1.53 Own Bad Virtual Address Register**

アドレス: 0xe7

**7.1.54 Own Exception PC Register**

アドレス: 0xe8

**7.1.55 Own Exception Cause Register**

アドレス: 0xe9

**7.1.56 Own Interruption Wait Register**

アドレス: 0xea

**7.1.57 Own External Interruption Level Register**

アドレス: 0xe8





bit 名	機能										
Trigger	各チャンネルのトリガモードの設定 . offset 0x00 ではチャンネル 31-17 まで , offset 0x04 ではチャンネル 16-1 のトリガモードの設定を行う .										
	<table border="1"> <thead> <tr> <th>bit</th> <th>Trigger Mode</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>High Level</td> </tr> <tr> <td>01</td> <td>Low Level</td> </tr> <tr> <td>10</td> <td>Rise Edge</td> </tr> <tr> <td>11</td> <td>Fall Edge</td> </tr> </tbody> </table>	bit	Trigger Mode	00	High Level	01	Low Level	10	Rise Edge	11	Fall Edge
bit	Trigger Mode										
00	High Level										
01	Low Level										
10	Rise Edge										
11	Fall Edge										

### 8.1.3 Request Sense Register

31	1 0
Request Sense Register	
	0

bit 名	機能
Request Sense	Trigger Mode Register で設定されたトリガが端子 IRLIN,IRQIN に入力されると , その割り込みチャンネルに対応したビットに 1 がセットされる . bit31 が IRQ31 , bit1 が IRQ1 に対応 . Read のみ

### 8.1.4 Request Clear Register

31	1 0
Request Clear Register	
	0

bit 名	機能
Request Clear	Request Clear Register の bit31-1 の中で 1 がセットされるとそれに対応する保持されていた割り込み要求が 0 になる . write のみ .

### 8.1.5 Mask Register

31	1 0
Mask Register	
	MI

bit 名	機能
Mask	31-1 ビットが割り込みチャンネルの 31-1 に対応し、1 をセットすることで割り込みをマスクできる。ただし、Mask が 1 の場合でも Request Sense Register はセットされる。
MI	0 ならば、IRLOUT に割り込みレベルラッチの内容を出力。1 ならばマスクし、IRLOUT には “L” を出力

### 8.1.6 IRL Latch/Clear

31	26'h0	6 5 4	0
		CL	IRL Latch

bit 名	機能
IRL Latch	割り込みレベルラッチの内容を出力
CL	1 を書き込むことで、割り込みレベルラッチの内容をクリアし、次の割り込みレベルをラッチする。

### 8.1.7 IRC Mode Register

31	31'h0	1 0
		Mode

bit 名	機能
Mode	1 ならば端子 IRLIN を IRQIN[31:27] として使用。0 ならば端子 IRLIN をそのまま IRLOUT に出力。

## 8.2 動作/使用方法

### 8.2.1 IRC

IRC モードレジスタが 1 に設定されると、入力端子 IRLIN[4:0](IRQIN[31:27]), IRQIN[26:1] に入力された割り込み信号はトリガモードレジスタに設定されたトリガモードに従ってその割り込みを保持します。保持された割り込みは MASK レジスタでマスクされていないもののうちでプライオリティが一番高いものがコード化されて割り込みレベルラッチ (IRL Latch) に保持されます。保持された IRL Latch のデータは MASK レジスタの MI ビットが (ビット 0) が 0 の場合、端子 IRLOUT[4:0] に出力されます。

IRC モードレジスタが 0 に設定されると、端子 IRLIN[4:0] に入力されたデータがそのまま IRLOUT[4:0] に出力されます。

表 8.1: 割り込みマップ

IRQ31	Bus Error
IRQ30	Address Error
IRQ29	Watch Dog Timer Error
IRQ28	Link
IRQ27	DMAC3
IRQ26	DMAC2
IRQ25	DMAC1
IRQ24	DMAC0
IRQ23	PCI
IRQ22	Reserved
IRQ21	IEEE1394
IRQ20	Ether MAC
IRQ19	SPI 1
IRQ18	SPI 0
IRQ17	PWM Input 2
IRQ16	Reserved
IRQ15	Pulse Counter 2
IRQ14	Pulse Counter 1
IRQ13	Pulse Counter 0
IRQ12	UART 1
IRQ11	UART 0
IRQ10	External IO 1
IRQ9	External IO 0
IRQ8	PWM Input 1
IRQ7	PWM Input 0
IRQ6	GPIO
IRQ5	RTC (MRMTP1 から)
IRQ1-4	Low に固定

### 8.2.2 RMT 固有機能

- IRL Unit

割り込みが入る度に実行中の全てのスレッドが割り込みハンドラを起動するのでは、非効率的であり、割り込みに対するレスポンス時間の増加を招いてしまいます。そのため、RMT では IRL Unit という IRL を各スレッドに割り当てる機構を持ちます。

IRL Unit にはスレッド毎に Interruption Wait Register が用意されています。Interruption Wait Register の各ビットは IRL に対応しています。IRC が出力した IRL を受け取り、Interruption Wait Register の IRL に対応するビットが 1 のとき、割り込みを受け付け、Exception Unit に外部割り込みが発生したこととその IRL を通知します。

Interruption Wait Register はコンテキストスイッチの際にレジスタセットと同様にバックアップ及びリストアされるために、コンテキストスイッチの度に設定する必要はありません。

- 割り込みによるスレッド起動

外部イベントによるスレッド制御の際のレスポンス時間を短縮するために、停止状態にあるスレッド (Status Register の STATE フィールドが 0010 or 0011) に対して外部割り込みがかかった場合はそのスレッドを実行状態にします。

ただし、この際に Interruption Wait Register の対象とする割り込みに対応するビットが 1 である必要があります。

### 8.2.3 例外処理プロセス

タイマ割り込み、ハードウェア割り込み (外部割り込み)、ソフトウェア割り込みが発生すると、Exception Cause Register の対応する Pending Register が 1 にセットされます。また、外部割り込みの場合は Exception Cause Register の HIRL に通知された IRL をセットします。これらの割り込みが通常通り発生するには Status Register の Interruption Mode(bit0) が 0、Interruption Mask(bit11-8) の対応するビットが 0 である必要があります。

これらの割り込みや RMTPU で例外が発生した場合、Status Register の Exception Level(bit1) が 0 ならば通常通り例外処理が発生します。Exception Unit では CPU core の exception 信号や Exception Cause Register の Pending Register を参照し、実際に例外処理を行います。例外処理の優先度は例外、タイマ割り込み、ハードウェア割り込み、ソフトウェア割り込みの順です。

例外処理が発生すると、次の動作が行われます。

- Status Register の Exception Level を 1 にセット
- その時点のモードを保持し、カーネルモードへ移行
- 例外を生じた命令、もしくは例外が発生した時点で最後にコミットされた PC を Exception PC Register に保持
- Exception Code Register の CODE フィールドに例外を識別するコード (表 8.2) を書き込む
- Status Register の Exception Base Address Enable が 1 ならば Exception Base Address Register の値に例外の内容に従ったオフセットの値を加えた番地に制御が移る  
Exception Base Address Enable が 0 ならば固定番地 (Exception Vector Location が 1 ならば 0xbfc00200, 0 ならば 0x80000000) に制御が移る

外部割り込みが起こった際に、対象となるスレッドが停止状態にある場合は通常の例外処理と異なり、スレッドが実行状態へ移行する処理のみ行われます。

外部割り込みに対する例外処理ルーチンでは処理に対応した IRC に保持されている割り込み要求をクリアし、IRL Latch をクリアする必要があります。その結果次の保持されている割り込みを IRL Latch に保持することができ、Exception Cause Register の対応する Pending フィールドが更新されます。タイマ割り込みやソフトウェア割り込みの場合は明示的に Exception Cause Register の対応する Pending フィールドをクリアする必要があります。

例外処理の終了時には ERET 命令を実行することで次の動作が行われます。

- Status Register の Exception Level を 0 にセット
- モードを例外発生時のものに戻す
- Exception PC Register に格納された番地に制御が移る

表 8.2: Exception Code, Offset

種類	コード	オフセット
I-TLB SPR Address Miss	0x01	0x010
I-TLB All Entry Locked	0x02	0x020
I-TLB No Entry Matched	0x03	0x030
I-TLB Thread Mode Error	0x04	0x040
I-TLB Protection Error	0x05	0x050
D-TLB SPR Address Miss	0x06	0x060
D-TLB All Entry Locked	0x07	0x070
D-TLB No Entry Matched	0x08	0x080
D-TLB Thread Mode Error	0x09	0x090
D-TLB Protection Error	0x0a	0x0a0
Coprocessor Unusable	0x0b	0x0b0
Reserved (Invalid) Instruction	0x0c	0x0c0
Sytem Call	0x0d	0x0d0
Break Point	0x0e	0x0e0
Integer Overflow	0x0f	0x0f0
Divide By Zero	0x10	0x100
Trap	0x11	0x110
Data Address Miss Align ( Load )	0x12	0x120
Data Address Miss Align ( Store )	0x13	0x130
Floating Point Overflow	0x14	0x140
Floating Point Underflow	0x15	0x150
Floating Point Divide By 0	0x16	0x160
Floating Point Inexact	0x17	0x170
Floating Point Invalid Operation	0x18	0x180
Reserve	0x19	0x190
Vector Integer Exception	0x1a	0x1a0
Vector Floating Exception	0x1b	0x1b0
Timer Interruption	0x1c	0x1c0
Hardware Interruption	0x1d	0x1d0
Software Interruption	0x1e	0x1e0
Software Interruption	0x1f	0x1f0

## 9

## クロックジェネレータ

## 9.1 接続図

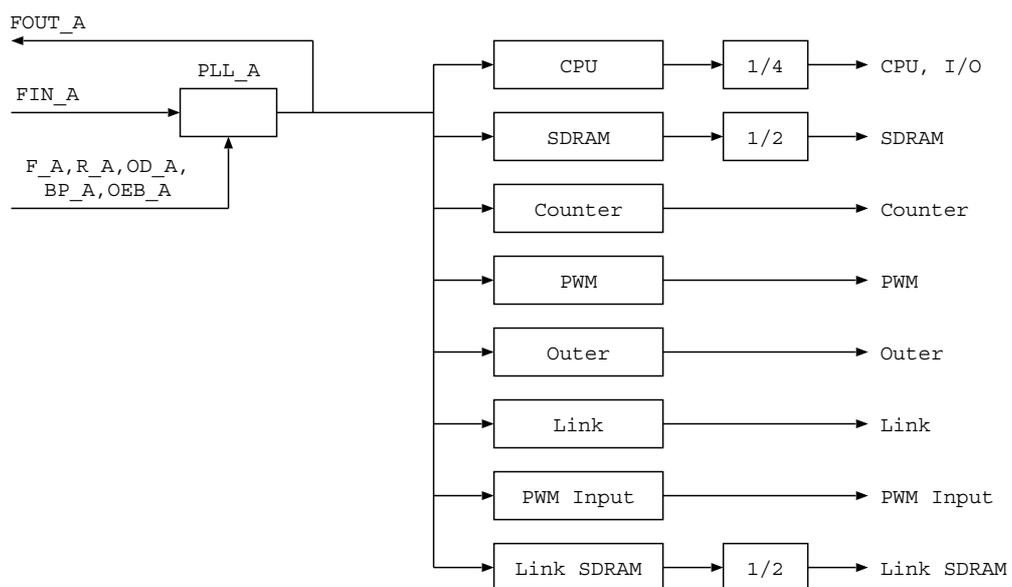


図 9.1: クロック生成部

ピン名	概要	デフォルト値
FIN_A	クロック入力	-
FOUT_A	PLL 出力	-
F_A	PLL の通倍数を制御する	16
R_A	PLL の通倍数を制御する	3
OD_A	PLL の通倍数を制御する	0
PD_A	PLL Power Down Mode (1:Power Down)	0
BP_A	PLL Bypass Mode (1: Bypass)	0
OEB_A	PLL Output Enable (0:Enable)	0

デフォルトでは FIN\_A より 75MHz のクロックを入力し、PLL\_A から 800MHz が出力される。これらを分周器で分周し各モジュールにクロックを供給する。

## 9.2 制御レジスタ

初期アドレス: 0xffffa000

### 9.2.1 Clock Enable

オフセット: 0x0000

各クロックを有効/無効にする。対応するビットを 1 で無効、0 で有効になる。初期値は全て 0。

31	22 21	0
Reserve	Enable	

bit 名	機能
Enable	21: Link Sdram, 20: PWM Input, 19: Link, 18: Outer, 17: PWM, 16: Counter, 15: SDRAM, 14: Vector Floating-Point, 13: Vector Integer, 12: Synchronize, 11: Floating-Point Reservation Station, 10: SIMD, 9: FPU, 8: Complex INT, 7: Context Cache, 6: PCI, 5: USB, 4: IEEE1394, 3: DMAC2, 2: DMAC1, 1: DMAC0, 0: CPU

### 9.2.2 Soft Reset

オフセット: 0x0004

各モジュールにリセットをかける。対応するビットを 0 にするとリセットがかかる。ビットを 1 に戻さない限りリセット状態が続く (CPU を除く)。

31	22 21	0
Reserve	Reset	

bit 名	機能
Reset	21: Link SDRAM, 20: PWM Input, 19: Link, 18: Outer, 17: PWM, 16: Counter, 15: SDRAM, 14: Vector Floating-Point, 13: Vector Integer, 12: Synchronize, 11: Floating-Point Reservation Station, 10: SIMD, 9: FPU, 8: Complex INT, 7: Context Cache, 6: PCI, 5: USB, 4: IEEE1394, 3: DMAC2, 2: DMAC1, 1: DMAC0, 0: CPU

### 9.2.3 Divider Ratio

オフセット: 0x0008 ~ 0x001c, 0x0028, 0x002c

各クロックの分周率を設定する．対応する分周器のアドレスと初期値は以下の通り．

分周器	デフォルト値	アドレスオフセット
CPU	1/2	0x0008
SDRAM	1/4	0x000c
Counter	1/8	0x0010
PWM	1/512	0x0014
Outer	1/8	0x0018
Link	1/1	0x001c
PWM Input	1/512	0x0028
Link SDRAM	1/4	0x002c

31	17 16 15	0
Reserve	T	Ratio

bit 名	機能
T	分周せずにクロックをスルーする (1/1 指定時)
Ratio	クロックの分周率を指定する．指定した数値の半分 (小数点以下切捨て) でクロックが立ち下がり，指定した数値でクロックが立ち上がる．1 を指定した場合の動作は保証外．1/1 の場合は T ビットを 1 にすること．

### 9.2.4 Clock Synchronization

オフセット: 0x0020

1 を指定したクロックの立ち上りエッジを CPU のクロックにそろえる．次のクロックで自動的に値はリセットされる．

31	8 7	1 0
Reserve	Sync	-

bit 名	機能
Sync	7: Link SDRAM, 6: PWM Input, 5: Link, 4: Outer, 3: PWM, 2: Counter, 1: SDRAM

### 9.2.5 All Reset

オフセット: 0x0024

このアドレスに書き込みを行うと全てにリセットをかける。

# 10

## スレッド制御

---

*Responsive Multithreaded Processor* におけるスレッド制御方法について述べる。

### 10.1 スレッドの種類

*RMT Processor* のスレッドは 2 つに分類される。

- アクティブスレッド
- キャッシュスレッド

アクティブスレッドとはレジスタファイルやプログラムカウンタなどの資源が確保され、プロセッサ内ですぐにも実行可能なスレッドを示す。キャッシュスレッドとはコンテキストキャッシュ内に保持されているスレッドを示す。*RMT Processor* が実行するスレッドはアクティブスレッドで実行状態にあるスレッドのみである。リセット時、スレッド ID が 0 のスレッドが優先度 0 でアドレス 0 から実行される。

### 10.2 スレッド制御命令

#### 10.2.1 作成・削除

新しくスレッドを作成する場合は `mkth` 命令を用いる。また、アクティブスレッドをコピーして新しいスレッドを作成することも可能である。

- `mkth`

新しくアクティブスレッドを作成する。`rs` でスレッド ID、`rt` でスタートアドレスを設定する。`mkth` 命令はアクティブスレッドを作成するだけで実行は開始しない。つまり `mkth` 命令で作成されたスレッドはストップ状態にある。実行を開始するためには `runth` 命令を使用する。スレッドの作成に成功すると `rd` に 1 が返り、失敗すると 0 が返る。

- delth  
アクティブスレッドを削除する。rs で削除するスレッドの ID を指定する。スレッドの削除に成功すると rd に 1 が返り、失敗すると 0 が返る。この命令が成功すると指定されたスレッドはプロセッサから削除される。
- cpthtoa  
アクティブスレッドを別のアクティブスレッドとしてコピーする。rs でコピー元のスレッド ID, rt で新たに作成するスレッドの ID を指定する。cpthtoa 命令はアクティブスレッドのコピーを新しいアクティブスレッドとして作成する。作成したコピーはストップ状態にあり、実行を開始するためには runth 命令を使用する。スレッドのコピーに成功すると rd に 1 が返り、失敗すると 0 が返る。
- cpthtom  
アクティブスレッドを別のキャッシュスレッドとしてコピーする。rs でコピー元のスレッド ID, rt で新たに作成するスレッドの ID を指定する。cpthtom 命令はアクティブスレッドのコピーを新しいキャッシュスレッドとして作成する。作成したコピーはコンテキストキャッシュ内にあるため、実行を開始するためには rstrth 命令などでアクティブスレッドにしなければならない。スレッドのコピーに成功すると rd に 1 が返り、失敗すると 0 が返る。

### 10.2.2 状態制御

アクティブスレッドは実行・停止のいずれかの状態にある。これらの状態は以下の命令を用いて制御する。

- runth  
停止状態のアクティブスレッドを実行状態にする。rs でスレッド ID を指定する。指定されたスレッドは実行状態になり、優先度に従って実行が開始される。実行開始に成功すると rd に 1 が返り、失敗すると 0 が返る。
- stopth  
実行状態のアクティブスレッドを停止状態にする。rs でスレッド ID を指定する。指定されたスレッドは停止状態になり、命令実行のスケジューリングからはずされる。再び実行するためには runth 命令を実行する。停止に成功すると rd に 1 が返り、失敗すると 0 が返る。
- stopself  
自分自身を停止状態にする。停止に成功すると rd に 1 が返り、失敗すると 0 が返る。
- chgpr  
スレッドの優先度を変更する。rs で変更するスレッドの ID, rt で新しい優先度を指定する。優先度の変更に成功すると rd に 1 が返り、失敗すると 0 が返る。優先度が変わると、つぎのクロックから新しい優先度で命令実行が制御される。

### 10.2.3 転送

*RMT Processor* はコンテキストキャッシュを持ち、コンテキストスイッチにおけるオーバーヘッドを軽減している。以下にコンテキストキャッシュとの転送命令を示す。





# 11

## 同期

---

### 11.1 共有レジスタ

31 個 (レジスタ番号 32 は設定レジスタ) の共有レジスタ (64bit) を持ち、そのレジスタにロックをかけることで同期を取ることが可能です。

共有レジスタには次の 3bit とスレッド ID が割り当てられ、その共有レジスタの使用権利をどのスレッドが持つのか意味します。

- Full/Empty bit
- Exclusive / Producer-Consumer bit
- Barrier bit

Full/Empty bit が現在このレジスタにロックがかかっているかを示し、他 2bit はどの種類のロックがかかっているかを示します。

### 11.2 同期命令

共有レジスタには次の命令を用いてアクセス可能です。

- RGPEX, WGPEX, RFPEX, WFPEX

Read 命令は対象となる共有レジスタの F/E bit が 0 のときに実行され、共有レジスタの値をデスティネーションレジスタに書き込みます。Read 命令が成功すると、対象のレジスタの F/E bit が 1 にし、自分のスレッド ID を書き込みます。

Write 命令は対象の共有レジスタの F/E bit が 1 でなおかつ自分のスレッド ID が共有レジスタの権利を所持しているスレッド ID と等しい場合に実行されます。成功すると、ソースレジスタの値を共有レジスタに書き込み、F/E bit を 0 にすることでロックを開放します。ロックが掛かっていない場合には NOP となります。

- GPCO, GPPR, FPCO, FPPR

Write 命令は対象となる共有レジスタの F/E bit が 0 のときに実行され、ソースレジスタの値を共有

レジスタに書き込みます。成功すると対象のレジスタの F/E bit を 1 にします。また、同時にスレッド ID を命令で指定し、その ID をロックをかけた対象の共有レジスタの権利者として書き込みます。Read 命令は対象となる共有レジスタの F/E bit が 1 でなおかつ共有レジスタの権利を所持しているスレッド ID が自分のスレッド ID と一致する場合に実行され、共有レジスタの値をデスティネーションレジスタに読み出します。命令の終了時には F/E bit を 0 にします。

- RGPSH, WGPSH, RFPSH, WFPSH  
ロックをかけない共有レジスタアクセス命令です。対象の F/E bit が 1 のときは実行できません。
- BAR  
バリア命令です。共有レジスタは到着スレッド数を数えるのに使用されます。ソースレジスタは対象となるスレッド数を示します。初めて実行される場合、対象の共有レジスタの F/E bit が 0 のときに実行できます。成功すると共有レジスタの値を 1 にし、F/E bit を 1 にします。2 番目以降のバリア命令では共有レジスタの値を increment しソースレジスタの値と等しくなるまで、そのスレッドはストールします。ソースレジスタの値と等しくなると、全てのスレッドのストールを解除します。

共有レジスタアクセス命令の実行条件を表 11.1 に示します。

表 11.1: 共有レジスタアクセスの実行条件

命令種別	実行条件			実行後			注釈
	F/E	E/P	bar	F/E	E/P	bar	
EX_READ	E	x	x	F	E	0	
EX_WRITE	F	E	0	E	x	x	TH ID 一致, 条件以外では NOP
CON_READ	F	P	0	E	x	x	対象 TH ID と一致
PRO_WRITE	E	x	x	F	P	0	
SH_READ	E	x	x	E	x	x	
SH_WRITE	E	x	x	E	x	x	
BARRIER	E	x	x	F	x	1	バリアに初めに到着する命令
	F	x	1	F	x	1	バリア待ち
				E	x	0	バリア解放

同期命令の失敗時にはデッドロックの回避のために、対象のスレッドのパイプライン中の命令を全て開放し、フェッチを止めることでストールさせます。対象のレジスタのロックが開放される (CON\_READ では書き込みが起こる) とストールが解除され、再び実行されます。

また、同期命令の失敗時に次のスレッド ID を調べ、そのスレッドがコンテキストキャッシュ内に退避されている場合は同期命令を失敗したスレッドと入れ換えます。

- 対象の共有レジスタにロックが掛かっている場合  
ロックを獲得しているスレッド
- Read Consumer 命令が値が書き込まれておらず失敗した場合  
Read Consumer 命令で指定する相手スレッド

また、バリア命令により他の到着スレッドを待つ場合には、同じバリア命令を実行するグループのスレッドを調べます。その際、そのスレッドが属するグループを設定する命令として PBAR 命令があります。PBAR 命令はバリアに使用する共有レジスタを指定します。バリア待ちのスレッドは現在使用している共

有レジスタ番号と同じ値を PBAR 命令で設定されたスレッドがコンテキストキャッシュ内にあるか調べ、存在する場合は自分と入れ換えます。

この同期命令失敗時のスレッド切替え機能は共有レジスタの 31 番に 0 以外の値を書き込むことで有効になります。default では無効化されています。



# 12

## Vector Unit

### 12.1 概要

RMT Processor の Vector Unit のブロック図を図 12.1 に示す。Vector Integer Unit、Vector Floating Point Unit 共に大きく 3 つの部分から成る。

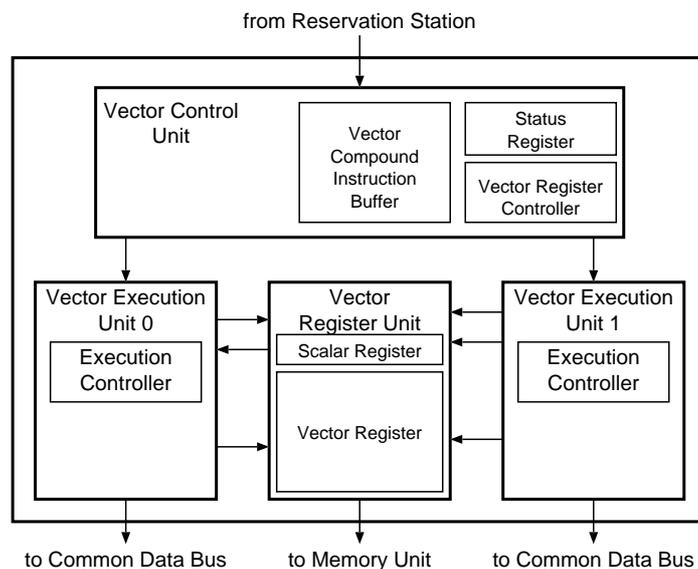


図 12.1: Vector Unit のブロック図

- Vector Control Unit

演算ユニットの制御、命令発行を行い、後述する Vector Register の割り当て、解放を行う。Vector Length、Mask Bit などの Vector Unit による演算に必要な制御情報を管理する。

- Vector Register Unit

ベクトル演算を行うためのレジスタを持つ。このレジスタは Vector Execution Unit との接続ポートの他に Memory Unit との接続ポートを持ち、Memory とのデータ転送が行われる。RMT Processor は 512 個のレジスタを持つ。

- Vector Execution Unit

Vector Register Unit からベクトル要素を取り出し、ベクトル演算を行い、結果を Vector Register Unit に格納する。

### 12.1.1 Vector Execution Unit

Vector Execution Unit のブロック図を図 12.2 に示す。

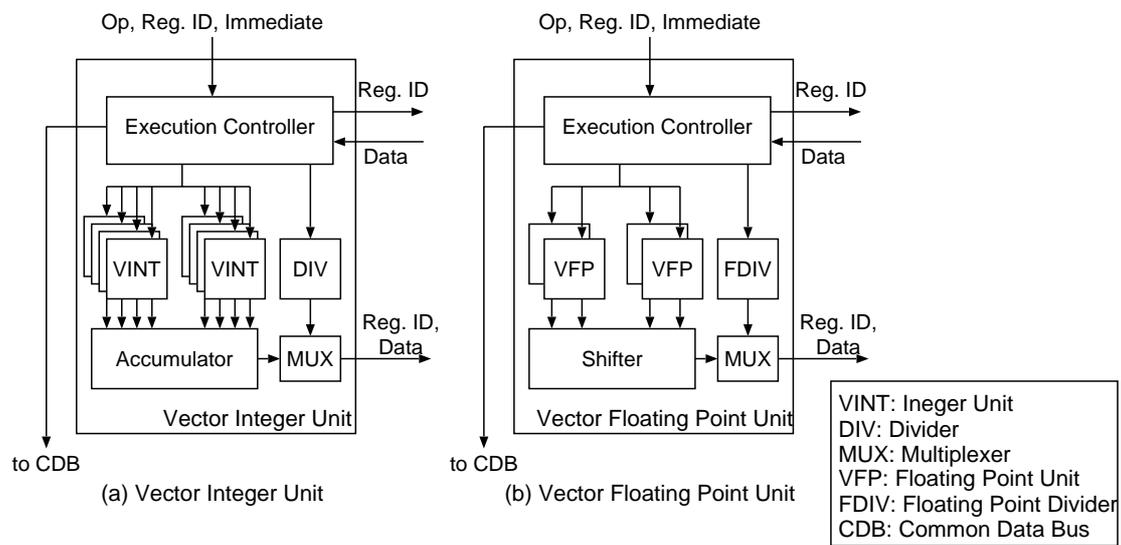


図 12.2: Vector Execution Unit のブロック図

Execution Controller は Vector Register Unit から必要なベクトルデータを取り出し、Vector Integer (Floating Point) Unit へ送る。

RMT Processor ではベクトル演算性能を向上させるために、Vector Integer Unit は 8 つ、Vector Floating Point Unit は 4 つの演算器を持つ。それぞれの演算器はパイプライン化され、1 クロックに 1 つの演算を開始する。

割り算は使用頻度が低いため、RMT Processor では Vector Divide Unit を 2 つある Vector Execution Unit のうちの片方のみ除算回路を持つ。

### 12.1.2 命令フォーマット

ベクトル演算命令のフォーマットは R-Type を拡張したものをを用いる。Opcode フィールドには Vector Integer 命令用に 011110 (Word)、0x110110 (Paired HalfWord)、111110 (Quad Byte)、Vector Floating Point 命令用に 011111 (Double / Single)、111111 (Paired Single) を用いる。図 12.3 にベクトル演算命令のフォーマットを示す。

rs、rt フィールドはベクトル演算の Source Register、rd フィールドは Destination Register を指定する。function フィールドにはベクトル演算の種類を指定する。subfunc フィールドはベクトル演算命令により

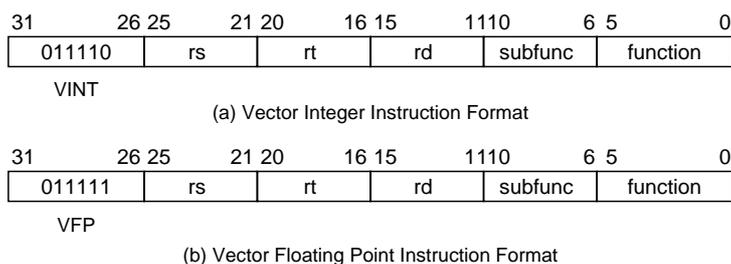


図 12.3: ベクトル演算命令フォーマット

用途が異なり、ベクトル - スカラ演算を行う際に用いる scalar bit や比較命令において比較条件を指定する cond bit、命令の順序制御を行う sync bit が含まれる。

## 12.2 Reserve/Release 命令

ベクトル演算を行うためには大きなベクトルレジスタが必要になる。これを各スレッドに持たせるとゲートサイズが大きくなり、また、ベクトル演算を行わないスレッドがある場合にはレジスタが無駄になる。よって1つのベクトルレジスタを用意し、それを複数のスレッドで共有して使用することによりベクトル演算を行う。ベクトルレジスタを必要な量だけ確保して使うことにより、複数のスレッドでベクトルレジスタを共有する。ベクトルレジスタの共有は、図 12.4 のようにベクトルレジスタを4つの領域に分け、128 エントリ、256 エントリ、512 エントリの固定サイズで確保する。スカラレジスタも同様に4つの領域に分け、確保したベクトルレジスタの大きさに応じて使用できるスカラレジスタの大きさが決定する。

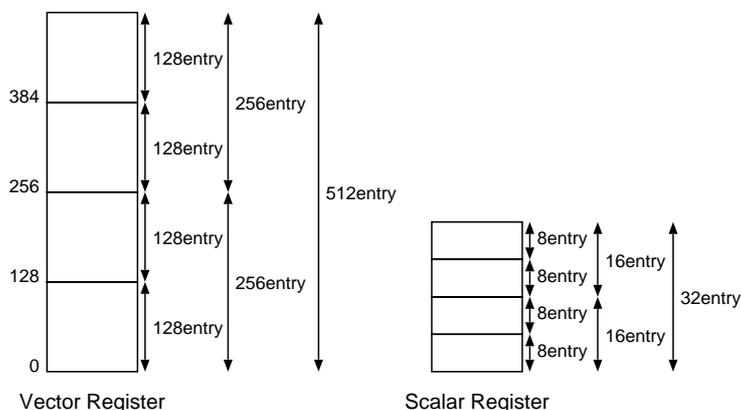


図 12.4: ベクトルレジスタのサイズ

確保したベクトルレジスタは、必要なベクトル長により分割される。ベクトル長は、8length、16length、32length、64length の中から選択する。図 12.5 に RMTProcessor で選択できるベクトルレジスタの構成を示す。

(a) は 128 個のベクトルレジスタを確保した場合の構成を示している。この場合、選択できる構成は、ベクトル長 8 のレジスタを 16 個、もしくはベクトル長 16 のレジスタを 8 個のどちらかとなる。(b) は 256 個のベクトルレジスタを確保した場合の構成で、ベクトル長 8 のレジスタを 32 個、ベクトル長 16 のレジスタを 16 個、ベクトル長 32 のレジスタを 8 個持つ構成の中から選択する。(c) は 512 個のベクトルレジ

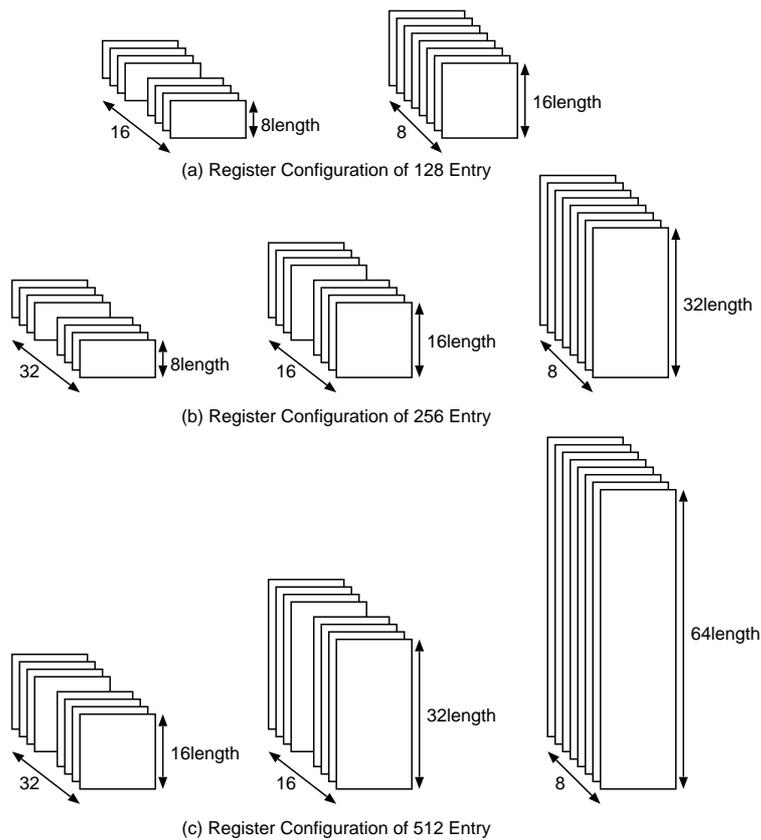


図 12.5: ベクトルレジスタの構成

スタを確保した場合で、ベクトル長 16 のレジスタを 32 個、ベクトル長 32 のレジスタを 16 個、ベクトル長 64 のレジスタを 8 個といった構成の中から一つを選択する。

Vector Unit で演算を行う場合、ベクトル演算を行う前にまず使用する分だけベクトルレジスタを確保する。ベクトルレジスタの確保は Vector Reserve 命令で行う。また Vector Unit で演算を行い、これ以上 Vector Unit を使用しなくなった場合は Vector Release 命令で確保していたベクトルレジスタを開放することにより、別のスレッドが新たに Vector Unit で演算を行うことが可能となる。Reserve、Release 命令を用いたプログラム例を図 12.6 に示す。

```

addu    $11, $0, 0x000A    # 256 Entry ( 32Depth x 8 ) Mode
virsv   $10, $11          # Reserve Instruction

== Vector Execution ==

virsl   $10                # Release Instruction

```

図 12.6: ベクトル演算のプログラム例

Vector Reserve 命令のオペランド (rs) で図 12.5 のどの構成でベクトルレジスタを使用するのかを指定する。指定する値 (Mode) は表 12.1 の中から選択する。値は上位 2bit が確保するレジスタの大きさ、下

位 2bit がベクトル長を示す。Vector Reserve 命令はベクトルレジスタの確保が成功すると rd に 1 を返す。rs で指定されたサイズのベクトルレジスタが確保できない場合は rd に 0 を返す。

表 12.1: Vector Register Mode の指定

(128 Entry)	
8 Depth × 16	0x4
16 Depth × 8	0x5
(256 Entry)	
8 Depth × 32	0x8
16 Depth × 16	0x9
32 Depth × 8	0xA
(512 Entry)	
16 Depth × 32	0xD
32 Depth × 16	0xE
64 Depth × 8	0xF

Vector Reserve 命令によりベクトルレジスタが確保されると、Vector Control Unit の Vector Register Controller 内にある Register Status Table に、確保したベクトルレジスタの情報を書きこむ。図 12.7 に Vector Status Table のフォーマットを示す。

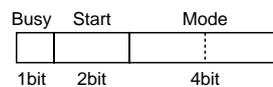


図 12.7: Vector Status Table

Busy Bit はそのスレッドがベクトルレジスタを確保しているかどうかを示す。Start Address は図 12.4 に示した 4 つに分割したベクトルレジスタのどの部分からベクトルレジスタを確保しているのかを示す。Mode は Vector Reserve 命令の rs で指定されたベクトルレジスタの構成を格納する。

Vector Release 命令を実行すると、確保していたベクトルレジスタを開放し、rd に 1 を返す。ベクトルレジスタを確保していない時に Vector Release 命令を実行すると rd に 0 を返す。

## 12.3 Status Register

Status Register はベクトル演算を行うために必要な以下の情報を各スレッドごとに保持する。

Status Register へのアクセスは vimfc, vfmfc (読み込み)、vimtc, vfmtc(書き込み) 命令を用いて行う。

## 12.4 複合演算命令

本ベクトル演算器では、ユーザが複合演算命令を定義し、複合演算実行命令 1 命令で定義された複合演算命令を処理することにより Vector Unit の使用率を向上させる。複合演算は Vector Control Unit の Compound Instruction Controller 内の Compound Instruction Buffer に定義する。Compound Instruction Buffer はベクトルレジスタやスカラレジスタと同じように 4 つに分割し、確保したベクトルレジスタと同

Address	Name	Description
0x00	Mask (Low)	ベクトルレジスタの要素 (下位) に対応し、1 を立てることにより演算をマスクする。マスクは最下位ビットが 1 番目の要素、最上位ビットが 32 番目の要素に対応する。
0x01 (Int)	Mask (High)	ベクトルレジスタの要素 (上位) に対応し、1 を立てることにより演算をマスクする。マスクは最下位ビットが 33 番目の要素、最上位ビットが 64 番目の要素に対応する。
0x01 (FP)	Rouding Mode	浮動小数点の丸めモードを指定する (0: Round to Nearest, 1: Round to 0, 2: Round to + , 3: Round to - )
0x02	Length	演算を行うベクトル長 (実際に指定するのはベクトル長 - 1) を指定する
0x03	Stride	Load / Store 時のアドレスのストライドを指定する。実際には各要素の間隔をワード数で指定する。0 を指定した場合は連続した番地からベクトル要素を読み込む。1 を指定すると 1 ワードおきに要素を読み込む。

じ領域を使うようにする。Compound Instruction Buffer 全体を 32 エントリであるため、ベクトルレジスタを 128 個確保した場合、使用できるエントリ数は 8 個、256 個確保した場合は、使用できるエントリ数は 16 個、512 個確保した場合は使用できるエントリ数は 32 個となる。

図 12.8 に Compound Instruction Buffer のフォーマットを示す。

31	30	29	28	23	22	17	16	11	10	0
N	SIMD	Rd	Rt	Rs	Op					

図 12.8: Compound Instruction Buffer のフォーマット

一つのエントリに一つの命令を定義し、それを複数合わせることで複合演算命令を定義する。Next(N) bit は次のエントリに複合命令が続くことを示す。複合命令の最後の命令は Next Bit を 0 にする。Next Bit を 0 にして複合命令を区切ることで、複数の複合命令を定義することができる。

SIMD フィールドには SIMD 演算を行う場合のビット幅を指定する。整数演算の場合、0x0 で 32bit 演算 (SIMD 演算を行わない)、0x1 で 16bit × 2 演算、0x2 で 8bit × 4 演算を行う。浮動小数点演算の場合、0x0 で通常の演算 (SIMD 演算を行わない)、0x1 で 32bit × 2 演算を行う。

Rs, Rt はソースレジスタ、Rd はデスティネーションレジスタを指定する。レジスタの指定は以下の通りである。

5	4	0
V	ID	

図 12.9: レジスタの指定

ベクトルレジスタを使用する場合、V ビットを 1 にする。スカラレジスタを使用する場合、V ビットを 0

にする。ID には使用するレジスタの ID を指定する。図 12.5 で指定した構成に従って ID の中で有効となるビット幅が決定する。例えば  $8\text{length} \times 16$  個の構成では ID のうち下位 4 ビットが有効となる。 $16\text{length} \times 32$  個の構成では ID の 5 ビットが有効となる。

実際に使用される rs、rt、rd は次に述べる VIECI、VFECI 命令で指定された rs、rt、rd のオフセット値として用いられる。例えば VIECI 命令の rs が 1 で Compound Instruction Buffer の rs の ID が 3 の場合、実際に指定される Register ID は  $1 + 3$  の 4 となる。

operation には演算を行う命令を指定する。以下に整数演算の場合のフォーマットを示す。

10	9	8	7	4	3	0
ACC	S	SUB OP		OP		

図 12.10: Operation のフォーマット (整数演算)

OP には以下を指定する。

- NOP (0x0)  
何も行わない。
- AND (0x1)  
論理積を計算する。
- OR (0x2)  
論理和を計算する。6 ビット目 (SUB OP) を 1 にすると NOR オペレーションとなる。
- XOR (0x3)  
排他的論理和を計算する。
- ADD (0x4)  
加算する。6 ビット目 (SUB OP) を 1 にすると減算となる。
- MULT (0x5)  
乗算する。4 ビット目 (SUB OP) を 1 にすると符号なし演算となる。6 ビット目 (SUB OP) を 1 にすると演算結果 (64bit 中) の上位 32 ビットを返す。
- SHIFT (0x6)  
シフト演算を行う。4 ビット目 (SUB OP) が 0 の場合は左シフト、1 の場合は右シフトとなる。6 ビット目 (SUB OP) が 1 の場合は算術シフトとなる。5 ビット目 (SUB OP) が 1 の場合はシフトではなくローテーションとなる。
- COMPARE (0x7)  
比較演算を行う。4 ビット目 (SUB OP) が 1 の場合はオペランドを符号無し数値として扱う。5-7 ビット目 (SUB OP) で比較条件を指定する。比較条件は 0x0: 常に偽、0x1: =、0x2: >=、0x3: >、0x4: 常に真、0x5: ≠、0x6: <、0x7: <= となる。
- THROUGH (0x8)  
rs の値を返す。

- MADD (0x9)

Multiply and ADD 演算を行う。6 ビット目 (SUB OP) が 1 の場合減算となる。

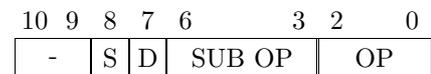
- DIV (0xf)

除算を行う。4 ビット目 (SUB OP) が 1 の場合、値を符号無しとして扱う。6 ビット目 (SUB OP) が 1 の場合、剰余演算となる。

S ビットを 1 にすると、SIMD 演算の場合にスカラ演算を行う。例えば 8bit × 4 演算の場合、S ビットを 1 にすると rt の下位 8bit を全てのフィールドで使用する。

ACC フィールドに 0x2 を指定すると、各ベクトル要素の演算結果を加算する。この場合、Rd はスカラレジスタを指定する必要がある。

以下に浮動小数点演算の場合のフォーマットを示す。



- DIV (0x7)

除算を行う。

D ビットが 1 の場合、オペランドを倍精度として扱う。D ビットが 0 の場合、オペランドを単精度として扱う。

S ビットを 1 にすると、SIMD 演算の場合にスカラ演算を行う。例えば 32bit × 2 演算の場合、S ビットを 1 にすると rt の下位 32bit を全てのフィールドで使用する。

複合演算命令は VIDCI、VFDCI 命令を用いて Compound Instruction Buffer に定義する。そして VIECI、VFECI 命令により複合命令の演算を開始する。それぞれの命令フォーマットを図 12.12 に示す。

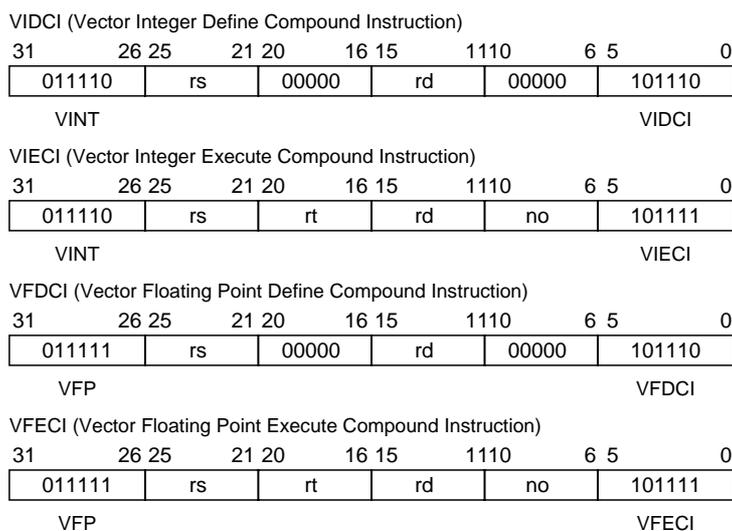


図 12.12: 複合演算命令のフォーマット

複合演算定義命令では rs に図 12.8 に従ったデータが入ったレジスタを指定し、rd に格納する Compound Instruction Buffer の ID を指定する。複合演算実行命令では rs、rt に Source Register、rd に Destination Register を指定し、no に実行を開始する Compound Instruction Buffer の位置を指定する。

複合演算実行命令が発行されると、Compound Instruction Controller は Compound Instruction Buffer から no に指定されたエントリの命令を読み出し、Register ID の変換を行ってから Vector Execution Unit へ読み出した命令を渡す。読み出した命令の Next Bit を見て 1 が立っていたら Compound Instruction Buffer の次のエントリから命令を読み出し演算を続ける。Next Bit が 0 ならばそこで複合演算を終了し、次の命令を受け付ける。

図 12.13 に複合演算命令の例を示す。例ではエントリの 0 から 1 でベクトルの加算をした後スカラレジスタの値で比較を行っている。また 2 から 9 で別の複合演算命令として、ベクトル変換命令を定義している。複合演算実行命令で no に 0 を指定するとベクトルの加算と比較を実行し、2 を指定するとベクトル変換命令を実行する。

	Next	Rd	Rt	Rs	Operation
0	1	V0	V0	V0	VADD
1	0	V1	S0	V1	VCMP
2	1	S0	V0	V0	VMAC
3	1	S1	V1	V1	VMAC
4	1	S2	V2	V2	VMAC
5	1	S3	V3	V3	VMAC
6	1	S4	V4	V4	VMAC
7	1	S5	V5	V5	VMAC
8	1	S6	V6	V6	VMAC
9	0	S7	V7	V7	VMAC

図 12.13: 複合演算の定義例

# 13

## Responsive Link

---

### 13.1 概要

*Responsive Link* は、各種ロボット、自動車、プラント、ホームオートメーション等の種々の分散制御を実現するために必要なハードリアルタイム通信、及び、画像、音声等のマルチメディアデータを滑らかに伝送するために必要なソフトリアルタイム通信の両方を同時に可能にするように設計を行っている。特に、リアルタイムの理論をそのまま応用可能なように、パケットの追い越し機能を実現している。

*Responsive Link* は柔軟なリアルタイム通信を実現するために、

- 通信パケットに優先度を付け、高い優先度の通信パケットが低い優先度の通信パケットを通信ノード毎に追い越し
- ハードリアルタイム通信（データリンク）とソフトリアルタイム通信（イベントリンク）の分離
- 全く同じネットワークアドレス（送信元アドレス及び送信先アドレス）を持つ通信パケットの経路を優先度によって別の経路に設定することによって専用回線や迂回路を設け実時間通信を制御
- 通信パケットの優先度を通信ノード毎に付け替え可能にすることによってパケットの加減速を分散管理で制御
- ハードウェアによるフレーム単位のエラー訂正

という方法を組み合わせることによって、分散管理を用いて大規模かつ量子時間の小さい実時間通信を実現する。さらに、

- 通信速度を動的に変更可能
- トポロジーフリー、
- Hot-Plug&Play

等の様々な機能を実現する。

*Responsive Link* は国内では情報処理学会試行標準 (IPSI-TS 2003:0006) として標準化されており、国際的には ISO/IEC JTC1 SC25 WG4 において標準化作業が行われている。

## 13.2 *Responsive Link* のインタフェース

ソフトリアルタイム通信（以下，単にデータと呼ぶ）のデータサイズ（画像データ，音声データ等）は大きく，それに対してハードリアルタイム通信（以下，単にイベントと呼ぶ）のデータサイズ（制御コマンド，同期信号等）は非常に小さい．従って，従来型の 1 系統の通信路で全ての通信を行う方法では，同時に通信すべき通信データとして，大量のデータパケットと，ごくわずかではあるが分散リアルタイム制御用途には非常に重要なイベントパケットが同一種類のパケットとして存在する．データとイベントを，共有された同一の通信線を通して時分割に通信を行う従来方式ではイベント伝達の時間が正確にバウンドできないので，ハードリアルタイムシステムは実現困難であると考えられる．

また，複数のモジュールでひとつの通信チャンネルを共有するシリアルバスでは，同時に何台のモジュールが通信するかによってバンド幅が動的に変化し時間をバウンドすることが困難であり，実効速度も出にくい．

さらに，リアルタイム通信におけるトレードオフとして，ソフトリアルタイム通信は主にバルク的なマルチメディアデータの通信等に用いられ，ハードリアルタイム通信は主に制御等に用いられるので，

- ソフトリアルタイム：バンド幅保証 ⇒ スループットをできるだけ上げたい
- ハードリアルタイム：レイテンシ保証 ⇒ レイテンシをできるだけ小さくしたい

という要求がある．しかしながらパケットサイズを大きくするとスループットは高くなるが，同時にレイテンシも長くなる．逆にパケットサイズを小さくするとレイテンシは短くなるが，オーバーヘッドが大きくなりスループットが低くなる．

従って，*Responsive Link* では，データラインとイベントラインを分離し，かつ各ラインの結合形態を point-to-point の双方向シリアル通信として設計されている（図 13.1 参照）．以下，それぞれをデータリンク，イベントリンクと呼ぶ．データリンクではパケットサイズを固定長かつ大きめにしてソフトリアルタイム通信に使用し，イベントリンクではパケットサイズを固定長かつ小さめにしてハードリアルタイム通信に使用する．

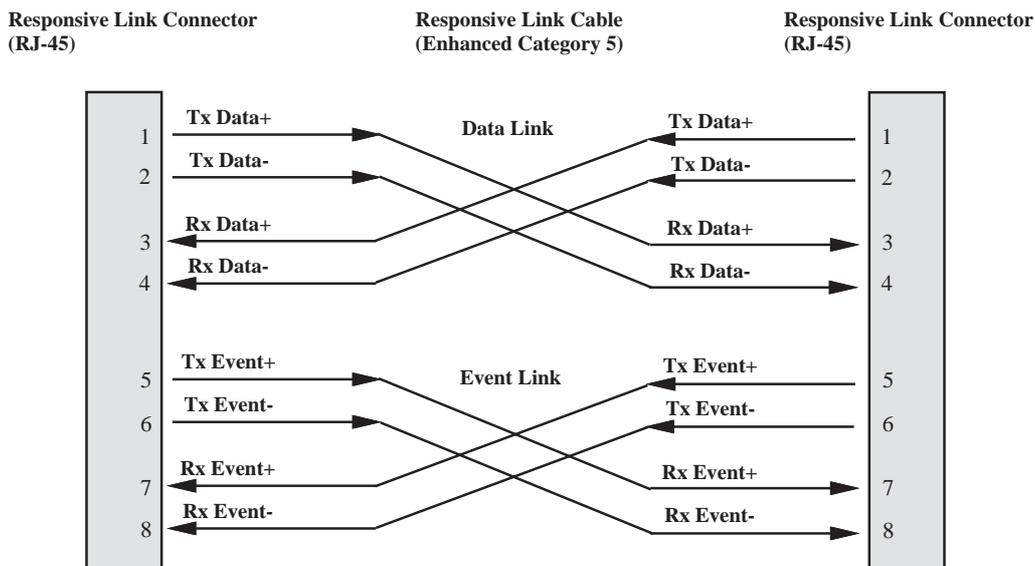


図 13.1: *Responsive Link* インタフェース



図 13.2 の通信パケットのヘッダ部に対して、図 13.3 に示すようにネットワークアドレスに優先度を付加する。256 レベル (8bit) の優先度を有し、優先度は 0 が一番低く、数字が大きくなるにしたがって高くなる。

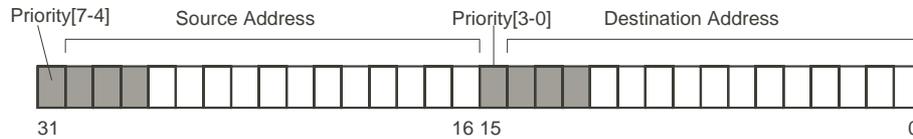


図 13.3: *Responsive Link* のヘッダフォーマット

*Responsive Link* の最大通信ノード数は、ネットワークアドレス長に制限され、優先度を使用しない場合、理論的には  $2^{32}$  ノードとなる (図 13.3 参照)。 *Responsive Link* の規格で推奨している使用法 (ノード毎にノードアドレスを割り当て、12bit の送信元アドレス、12bit の送信先アドレス、8bit の優先度を用いてルーティングを行う) の場合には、 $2^{12} = 4096$  ノードとなる。4096 よりノード数が大きなシステムを構築する際には、経路にアドレスを割り当てる (24bit のネットワークアドレスと 8bit の優先度を用いてルーティングを行う) ことにより  $2^{24} = 16M$  ノードまでのノード数をサポートする。

### 13.3.1 固定長 (64B) のデータパケット

レスポンスリンクのスイッチ部はカットスルー型のスイッチを採用している。データパケットは固定長 (64byte) で、パケットに優先度が付加されている。データパケットはアドレス (ソースとデスティネーション)、ペイロード、ステータスから構成される。カットスルー型のスイッチなので、衝突が起きない限りデータはノードを経由して転送されるが、あるノードで衝突が起こった場合は、優先度の高いパケットが低いパケットを追い越すことができるようになっている。この機能によって従来までの集中管理型ではなく分散管理型のリアルタイム通信を実現している。

データパケットは、2byte の送信元アドレス・2byte の送信先アドレス・56byte のペイロード・4byte の制御・状態データの計 64byte より構成される。4byte の制御・状態データは以下のフォーマットをとる。

UD	ユーザ定義フラグ (任意に設定可能)
Full	ペイロード 56byte がすべて有効データで埋められているとき 1, それ以外は 0
Data Length	ペイロードの有効データ長。1 から 56 の値をとる。
Dirty0-15	パケットのどのワード (4byte) にエラーが存在するかを示すビット。パケットの 2 ワード目にエラーがある場合は Dirty1 が 1 となる (ハードウェアによりセットされる)
Start	このパケットがスタートパケットであるとき 1, それ以外は 0
End	このパケットがエンドパケットであるとき 1, それ以外は 0
Int	このパケットを受け取る際に割り込みを生じるときは 1, それ以外は 0
Fatal	このパケットに致命的なエラーが存在するときは 1, それ以外は 0 (ハードウェアによりセットされる)
Correct	このパケットの一部分にエラーが存在し、それが修復されたときは 1, それ以外は 0 (ハードウェアによりセットされる)
Serial Number	パケットのシリアルナンバ。スタートパケットが 0, 以降 0 から 7 までを繰り返す。

### 13.3.2 固定長 (16B) のイベントパケット

イベントパケットも固定長 (16byte) で、送信元アドレス、送信先アドレス、ペイロード、ステータスから構成される。イベントの場合もノードで衝突がない限り、直接ノードを経由してルーティングされるが、衝突が生じた場合はデータの場合と同様に、優先順位に従ってパケットの追い越しを行なう。

4byte の制御・状態データは以下のフォーマットをとる。

UD	ユーザ定義フラグ (任意に設定可能)
Full	ペイロード 8byte がすべて有効データで埋められているとき 1, それ以外は 0
Data Length	ペイロードの有効データ長. 1 から 8 の値をとる.
Dirty0-15	パケットのどのバイトにエラーが存在するかを示すビット. パケットの 2 バイト目にエラーがある場合は Dirty1 が 1 となる (ハードウェアによりセットされる)
Start	このパケットがスタートパケットであるとき 1, それ以外は 0
End	このパケットがエンドパケットであるとき 1, それ以外は 0
Int	このパケットを受け取る際に割り込みを生じるときは 1, それ以外は 0
Fatal	このパケットに致命的なエラーが存在するときは 1, それ以外は 0 (ハードウェアによりセットされる)
Correct	このパケットの一部分にエラーが存在し, それが修復されたときは 1, それ以外は 0 (ハードウェアによりセットされる)
Serial Number	パケットのシリアルナンバ. スタートパケットが 0, 以降 0 から 7 までを繰り返す.

### 13.3.3 優先度による追い越し機構

優先度を用いたパケットの追い越し機構を実現するために、追い越し用バッファと退避用外部記憶を有したネットワークスイッチを搭載している。図 13.4 は 5 入力 5 出力で一つの入力部当たり追い越し用バッファが 4 パケット分あるネットワークスイッチの構成を示している。(実際に RMTTP に実装されている *Responsive Link* には 8 パケット分の追い越し用バッファが実装されている。) 図 13.4 において、最後の数字はポート番号を示している。入力ポート (In0~4) から入力された通信パケットは、通信ノードで衝突しない場合、そのまま出力ポート (Out0~4) へ出力を行う。異なる入力ポートから入力された通信パケットが同じ出力ポートに出力を行なう場合、通信パケットに付加された優先度に従い、低い優先度の通信パケットを追い越し用バッファ (意味的には追い越され用バッファ) に貯めて出力を待たし、高い優先度の通信パケットを先に出力させる。高い優先度の通信パケットの出力の後に低い優先度の通信パケットを追い越し用バッファから出力ポートに出力し、優先度に従った通信パケットの追い越しを行う。

この際、内部のスイッチングは、ヘッダ部受信のオーバヘッド及びルーティングテーブルの参照時間を隠蔽するために図 13.4 のように 8bit パラレル (byte 単位) で行うように設計されている。

上記の通信パケットの追い越しを実現するために通信パケットの大きさと等しい追い越し用バッファを 8 本入力ポート側に搭載している。さらに、出力が待たされ続けている時に入力が入り続けバッファが溢れそうになった場合に、追い越し用バッファの内容を一時的に退避するための退避用外部記憶 (DDR SDRAM) を設けることができるようになっている。

図 13.5 は図 13.4 のネットワークスイッチのひとつの入力部の詳細を示している。図 13.5 において、最後の数字はポート番号を示している。通信パケットの追い越しを行うために、まず、入力ポート (In) から入力された通信パケットを、入力ポインタ (In-Pointer) で指し示されている追い越し用バッファ 0 から追い越し用バッファ 3 のうち使用されていない空バッファに書き込む。入力パケットのヘッダ部分は必ず全て受信し追い越し用バッファに書き込み、その受信されたヘッダを元に図 13.6 のようなルーティングテーブルを参照し出力ポート番号と優先度を得る。得られた出力ポート番号は図 13.5 のリンクストロブ (L0~

L4) に書き込む．例えば L2 ビットが有効であればその入力パケットの出力先は出力ポート 2 であることを示す．

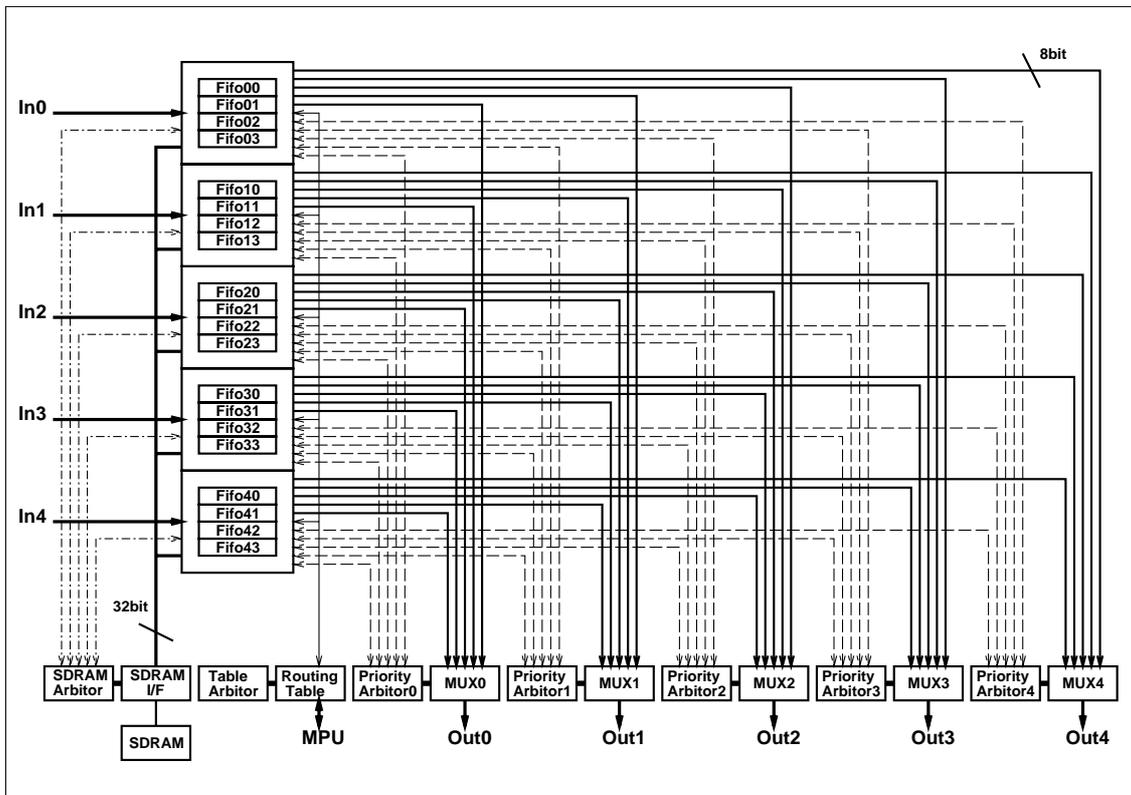


図 13.4: Responsive Link のネットワークスイッチ

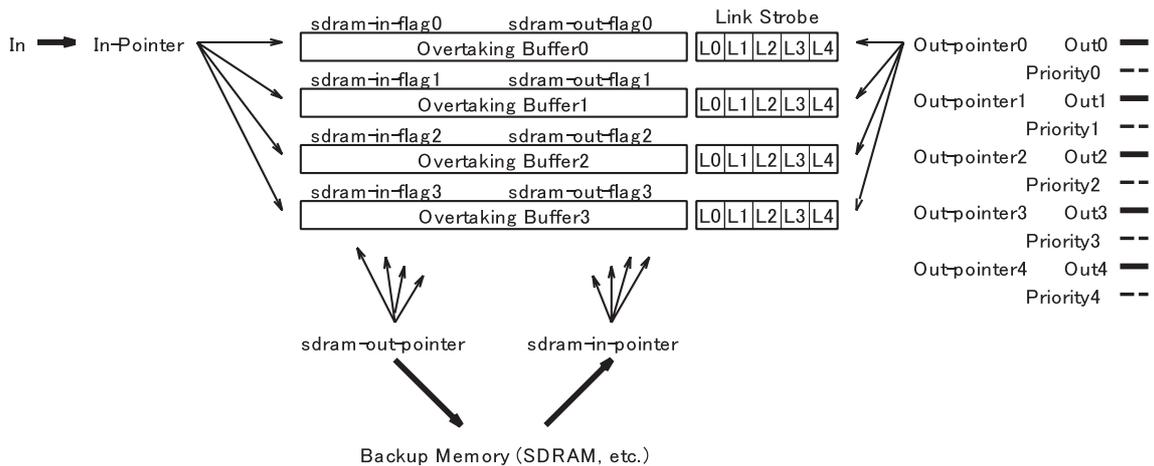


図 13.5: Responsive Link の追い越し用バッファ

図 13.5 において L0 から L4 までの複数ビットが有効であればマルチキャストを意味し，全て有効であればブロードキャストを意味する．入力部の出力側は出力ポート毎 (Out0 ~ Out4) にそれぞれ独立に各追

い越し用バッファのリンクストロブを参照し、自出力ポートのリンクストロブが有効な場合、出力側ポート側に配置された当該優先度調停器 (図 13.4 の Priority ArbitorN) に対して優先度と共に出力要求を行なう。図 13.5 の PriorityN は図 13.4 の Priority ArbitorN に接続されている。優先度調停器は、出力要求が一つの入力ポートからだけある場合はただちに出力許可を与え、出力要求が複数ある場合は優先度の一番高いものに出力許可を与えるようにする。一番優先度の高い要求が複数ある場合は、ラウンドロビン方式で出力許可を与える。

通信パケットの衝突がない場合や、衝突があってもその時点での最高優先度の通信パケットの場合は、ヘッダの受信とルーティングテーブル参照の遅延時間後に直ちに出力を開始する。入力部の各出力ポート側ではパケットの送信終了直後に対応するリンクストロブを無効にし、全てのリンクストロブが無効になったらそのバッファが空であることを意味する。

例えば、In-pointer が追い越し用バッファ1 を指している場合、入力ポート In から入力されたパケットは、まずヘッダ部が追い越し用バッファ1 に入る。次にそのヘッダを元にルーティングテーブルを引き、リンクストロブと優先度を得る。例えば、L1 と L3 が有効だった場合、Out-pointer1 と Out-pointer3 は共にその追い越し用バッファ1 を指し、Out1 と Out3 側が出力要求と共にその優先度をそれぞれ Priority1 と Priority3 に出力する。例えば、Out3 にすぐに出力可能であれば、出力許可が Priority Arbitor3 から与えられるので、直ちに追い越し用バッファ1 から Out3 に出力を開始する。出力が終われば、Out3 側が追い越し用バッファ1 の L3 をクリアする。また、Out1 には直ちに出力許可がおりなかったとすると、出力許可が得られるまで出力要求と優先度を Priority1 に出力し続ける。ここで、Out1 への出力待ちの状態で、同じく Out1 へ出力したい高優先度パケットが新たに追い越し用バッファ2 に入ってきた場合、Out-pointer1 はより優先度の高いパケットの入っている追い越し用バッファ2 を指すようになり、その高優先度パケットの出力要求と優先度を Priority1 に出力するようになる。後から到着した高優先度パケットの出力が終わると、他に Out1 に出力したい高優先度パケットがない場合、Out-pointer1 は再び追い越し用バッファ1 を指して、同様に出力を継続しようとする。このように、同一系路上の先行する低優先度パケットが待たされている際にも、後続の高優先度パケットが追い越していくことを可能にする。

図 13.5 において、空バッファが少なくなっていく残り 1 本になってしまった場合、次の入力パケットは退避用外部記憶 (DDR SDRAM) に退避を行うようになっていく。出力が進んで空バッファの残りが多くなり 2 本以上になると、退避用外部記憶に退避されていた入力パケットを優先度を考慮して追い越し用バッファに書き戻すことにより、出力を継続する。

また、退避用外部記憶が溢れそうになると、そのノードのプロセッシングコアに対して割り込みをかけられるようになっていく。退避用外部記憶が溢れる場合は、アドミッションコントロールを行ってパケットの破棄を行ったり、送信元に送信データの一時停止を行うように制御する等の方法が考えられるが、そのプロトコル自身は *Responsive Link* の規格では定めていない。それらは上位のプロトコルで行うことになるので、上記割り込みをかける閾値を設定可能にするように設計している。

リアルタイム通信を実現するために、優先度によるパケットの追い越しをこのように再送を行なわなくてもよいように設計されている。

## 13.4 フレームフォーマット

1byte は、図 13.2 の Frame Format ような冗長ビットを含めたフレームとしてシリアルに送受信される。詳細は低レベル通信の節を参照。

**Data bits** 8bit のデータ

**Redundancy bits** byte 毎に Redundancy bits (冗長ビット) を付加することで、CRC 等とは異なり、パケット全てを受信し終わらなくても byte 毎にエラー訂正が可能

### 13.5 ルーティング・テーブル

*Responsive Link* の経路制御は、図 13.6 に示すようなルーティングテーブル（経路制御表）を設定することによって行う。ルーティングテーブルは、*Responsive Link* コントローラ内に置き、そのノードのローカルプロセッサから読み書きできるようになっている。図 13.6 において、Reference 部はパケットのヘッダと同一であり、Referent 部に当該パケットに関する設定を行う。EE ビット及び DE ビットは、それぞれそのラインがイベントリンク用の設定かデータリンク用の設定かを示す。両方とも設定されていれば、両リンクとも同様の設定になる。L[4-0] は、前述のリンクストロービットであり、出力ポート（複数可）を指し示す。

ルーティングテーブルの大きさ（エントリ数）は実装依存で有限となるため、非常に大きな分散システムを構築する際には溢れてしまう可能性がある。ルーティングテーブルに入りきらない大規模なシステムを構築する際には、ローカルノードプロセッサの主記憶上に完全なルーティングテーブルを用意し、*Responsive Link* コントローラ上のルーティングテーブルはキャッシュとして用いるようにする。つまり、TLB 付きの MMU とページテーブルを用いたメモリ管理と同様な管理手法を行うようにする。

そのために、ルーティングテーブルにヒットしないエントリがあった際には、ローカルノードのプロセッサに対して割り込みをかけると同時に、該当パケットを一時的に前述の退避用外部記憶に退避する。割り込みをかけられたプロセッサは、主記憶上の完全なルーティングテーブルをソフトウェアでエントリを検索し、そのエントリを *Responsive Link* コントローラ上のルーティングテーブルの適切なエントリとスワップするようにする（多くの場合、最近使用されていないエントリとスワップすると考えられるが、それは RT-OS のポリシ依存である。）*Responsive Link* コントローラ側は、イベントリンクとデータリンクそれぞれについて、LRU エントリアドレスが分かるように設計し、RT-OS に対してヒントを与えるようにする。その後、退避していたパケットを追い越しバッファに書き戻すことによって継続的にルーティングを実現する。

上記のような機構により、大規模な分散リアルタイムシステムが構築可能である。ただし、コントローラ内のルーティングテーブルに収まる範囲の規模でないと、厳密にハードリアルタイム性を維持するのは困難となる。

また、分散リアルタイムシステムの規模が大きくなればなるほど（つまりルーティングテーブルのサイズが大きくなればなるほど）通信のジッタは大きくなり、リアルタイム性の時間粒度も大きくなるが、近傍で激しく通信している経路をキャッシュに置き、そうでないものは主記憶上のルーティングテーブルに置く等の方法をとることにより、運用が可能であると考えられる。

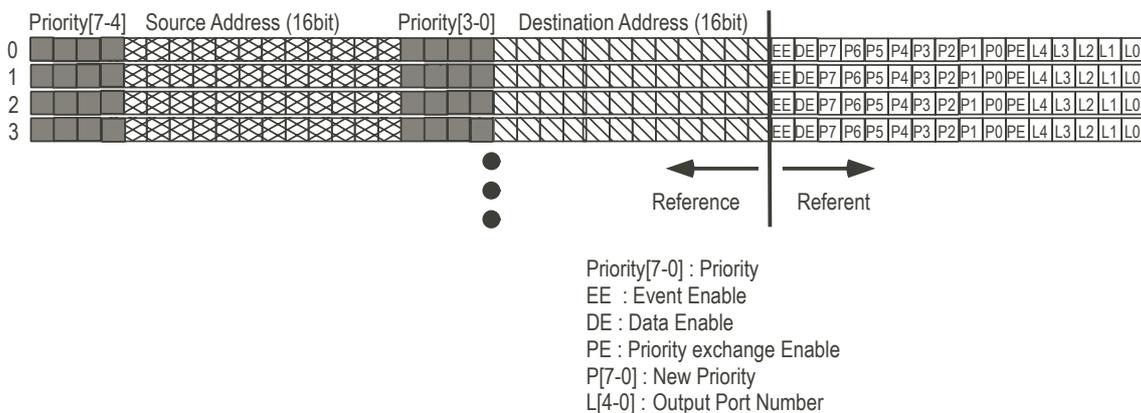


図 13.6: *Responsive Link* のルーティングテーブル

## 13.6 パケットの加減速制御

リアルタイム通信パケットの制御を外部から行うことができるようにするために、通信ノード毎にパケットの優先度の付け替えができるようにして、分散管理型でのリアルタイム通信の制御を実現している。

優先度の付け替えは、図 13.6 のルーティングテーブルを用いることによって行なう。図 13.6 において、ネットワークアドレスと優先度を元にルーティングテーブルを参照し出力ポート番号を決定する際に、優先度を付け替えないモード（図 13.6 の優先度付替ビット PE が無効）の場合は優先度はそのままであるが、優先度を付け替えるモード（優先度付替ビット PE が有効）の場合、出力ポートから出力する際に優先度 (Priority[7-0]) を新優先度 (P7~P0) に置き換える。つまり、現ノードでの通信パケットの優先度は入力パケットのヘッダに付加されている優先度で決定され、その優先度に従って追い抜きやルーティングが決定されるが、次ノード以降での通信パケットの優先度を制御することができる。ルーティングテーブルの設定はソフトウェア（分散リアルタイムオペレーティングシステム等）で行ない、ルーティング（経路制御）自身はハードウェアで行なうようになっている。

このパケットの加減速制御機構により、例えば、リアルタイム通信の流量やレイテンシを監視するミドルウェアを用いて、リアルタイム通信の制御を可能とする。リアルタイム性の低い通信パケットがバルク的に流れていて、そのパケットが他のリアルタイム性の高いパケットの通信のリアルタイム性を阻害していたとしたら、通信監視ミドルウェアが当該パケットの優先度を下げることによって、リアルタイム性の制御を行うことができる。あるいは、あるノードでデッドラインミスが発生してしまった場合、その通信パケットの優先度を途中の経路で上げることにより（特にホットスポットで優先度を上げると効果的）、次回からのデッドラインミスを防ぐことが可能となる。

## 13.7 優先度に従った経路制御

優先度に従って専用回線や迂回路を設けたり、データの流量の制御を行なうことができるように、全く同じネットワークアドレスを持つ通信パケットの経路を優先度によって別の経路に設定することができるようにしている。そのために、基本的にはネットワークアドレスと優先度の組でルーティングテーブルを参照する。

優先度ごとに必ずルーティングテーブルを設定しなければならないと煩雑であるので、デフォルトルートを設定することができる。ネットワークアドレスは同じであるが優先度が一致する組み合わせ（経路）がルーティングテーブル上に無い場合には、最も優先度の低い優先度 0 の経路がデフォルト経路となるようになっている。つまり、

1. ネットワークアドレスと優先度の両方が一致すればその経路が第一優先
2. ネットワークアドレスは一致するが優先度が一致しない場合、優先度 0 の経路

となる。ここで、優先度 0 の経路はデフォルト経路となるので、途中で経路が消滅してしまわないようにルーティングテーブルに必ず登録する必要がある。

図 13.7 は、2次元格子の交点に通信ノードがあるとし、全く同じ送信元から送信先に対して異なる優先度の通信パケットを同時に通信している状態を示す。例えば、優先度 0 のイベントリンクの経路上は別の通信ノードからの通信パケットも同じ経路を通過して送信先に通るように設定しておき、優先度 3 の経路は送信元と送信先の優先度 3 の通信パケットしか通らないように設定しておくことにより、他の通信パケットと衝突が起きない専用回線を実現することができる。*Responsive Link* には優先度による追い越し機構があるが、衝突があると追い越しのために多少のオーバーヘッドが生じてしまうので、このように優先度を用いてパケットの衝突が全くない専用回線を設定することにより、非常にレイテンシ及びジッタが小さいリアルタイム経路の実現を可能とする。また、優先度が異なる経路を複数設定することによってマルチリンクを実現し、バンド幅を広げることも同時に可能とする。

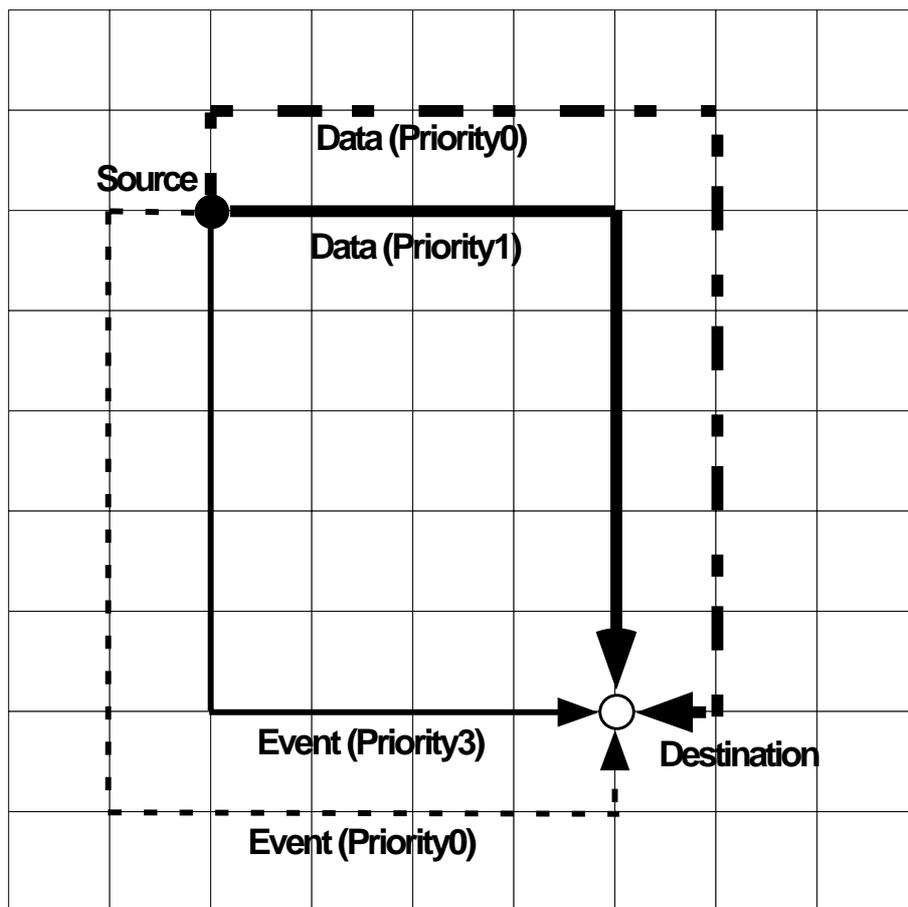


図 13.7: *Responsive Link* の優先度付経路

制御用の分散システムでは図 13.8 のような木構造を採る場合が多い。図 13.8 において通信ノード 0 から通信ノード 5 に通信する場合、優先度 0 の通信パケットは途中で通信ノード 1 と通信ノード 2 という中間ノードを経由して通信を行なうが、優先度 1 の通信パケットは通信ノード 0 から直接通信ノード 5 へ通信を行なうことができる。これは、例えばヒューマノイドロボットを開発した際に、当初は頭モジュール、肩モジュール、肘モジュール、指モジュールと接続しそれらの経路をホップして通信を行っていたが、設計後にどうしても頭モジュールと指モジュール間の通信レイテンシが間に合わないと判明した場合、後付で頭モジュールと指モジュールを直接接続し優先度を変えて通信することにより、容易に通信経路（この場合は専用回線）の増設を可能とする。この機能は、実システムを構築する際に手助けとなる。

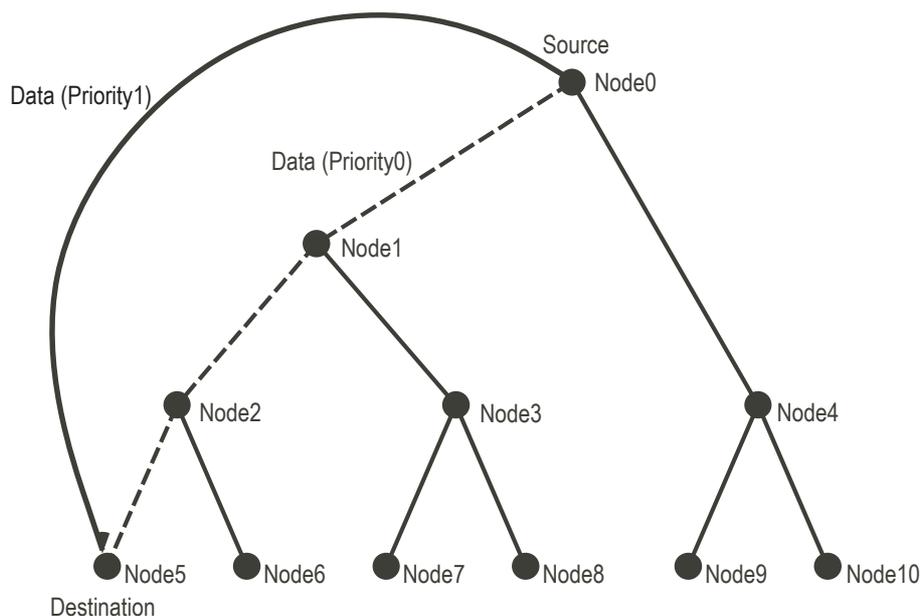


図 13.8: *Responsive Link* の優先度付木構造経路

## 13.8 低レベル通信

*Responsive Link* は分散制御用途であるので、必ずエラー訂正を行わなければならない。その際、できるだけエラー訂正によってリアルタイム性が損なわれないようにする必要がある。

ここで、パケット単位で CRC を付加しエラー訂正を行う方法では、パケット全体を受信しないとエラー訂正できない。その場合、ホップ毎にレイテンシが積算されていくので、リアルタイム通信用のエラー訂正としては好ましくない。そこで、レスポンスリンクでは 1 ホップごとにフレーム（図 13.2 参照）単位でエラー訂正を行い、1 フレーム（8bit データ+4bit 冗長符号）につき 1bit のエラーであれば、再送することなしにハードウェアで誤り訂正を行うようにする。

### 13.8.1 CODEC

*Responsive Link* の CODEC は、8bit の情報ビット列に、誤り訂正用の 4bit の冗長ビット列を加えた 12bit を 1 フレームとして通信を行う。本 CODEC で行われる符合化は、以下のような流れとなる。

1. 巡回組織ハミング符合化（冗長ビット列を加える誤り訂正符合化）
2. Bit Stuffing（連続した 1 の符合に 0 を挿入）
3. NRZI 符合化

以下、各符合化について説明を行う。

#### 13.8.2 巡回組織ハミング符号化

誤り訂正符合として、生成多項式が  $x^4 + x + 1$  の巡回組織ハミング符合を採用する。この符合化では、8bit データの下位 (LSB) 側に 4bit の冗長ビット列を付加することで、12bit 中の任意の 1bit の誤りを受信

側で訂正することを可能にし、表 13.1 より誤りの位置を特定できる。送信時には、これら 12bit のビット列は、MSB 側から 1bit ずつ送信を行う。

表 13.1: シンドロームとエラーの位置

Syndrome	Error Position (4 redundancy bits)	Meaning
0000	00000000 0000	No error
0001	00000000 0001	Redundancy-bit error
0010	00000000 0010	Redundancy-bit error
0100	00000000 0100	Redundancy-bit error
1000	00000000 1000	Redundancy-bit error
0011	00000001 0000	0bit error
0110	00000010 0000	1bit error
1100	00000100 0000	2bit error
1011	00001000 0000	3bit error
0101	00010000 0000	4bit error
1010	00100000 0000	5bit error
0111	01000000 0000	6bit error
1110	10000000 0000	7bit error

### 13.8.3 Bit Stuffing

1 が長時間連続することによって引き起こされるリンクへの直流成分が発生や、受信側のビット同期への支障を回避するために、通信データ中に 5 つの連続した 1 が現れた場合には、その後ろに 0 を挿入する。

### 13.8.4 NRZI 符合化

最終的に送信される際に NRZI(Non Return to Zero Inverted) 符合化を行う。NRZI 符合化は、0 を送る場合にはリンクのデータビットを反転し、1 を送る場合にはデータビットの状態を前のまま保持する。

### 13.8.5 セットアップパターン

電源投入直後や、予期できないバースト的なリンクエラーなどの後は、送受信インタフェース間でフレーム同期がとれない場合がある。そのような場合、明示的にリンクの初期化を行うようにする。具体的には、以下に示すセットアップパターンを受信側に送信する。

セットアップパターン：000001111110

このパターンは、連続した 1 が 6 個以上は連続しないという bit stuffing の規則に反しているため、いかなる通常の packets とも区別される。受信側では、このパターンを受信するとその後、最初に認識したフレームを、新しい packets の第 1 フレームとして解釈する。

表 13.3: 通信速度とケーブル

Speed (Mbaud)	100	200	400
Maximum Length (m)	100	80	60
Recommendable Cable	Cat5e	Cat5e	Cat6

### 13.8.6 DPLL を用いたビット同期

受信側に DPLL(Digital Phase Lock Loop) 機構を設計し、受信用クロックの立ち上がりエッジに同期して受信信号をサンプリングする。1bit 転送あたりのサンプリング数はソフトウェアの設定によって可変(4,8,16,32,64,128,256)にする。DPLL では設定された周期ごとに受信用クロックを生成し、受信信号のエッジを検出することにより、信号のエッジ間の中央で受信用クロックが立ち上がるように、受信用クロックの周期を微調整を行う。表 13.2 に DPLL のモードを示す。

モード名	p_mode2	p_mode1	p_mode0	d_clk 周期/1bit 転送
Mode2	1	1	1	2
Mode4	0	0	0	4
Mode8	0	0	1	8
Mode16	0	1	0	16
Mode32	0	1	1	32

表 13.2: DPLL モードの設定

### 13.8.7 エラーの取扱い

*Responsive Link* では、誤り訂正符合化によって 1[bit/frame] の誤りまでは自動的にエラー訂正を行うことができる。エラーの箇所を受信側で特定するために、図 13.2 のトレイラ部の Dirty ビットを立てる。具体的には、データリンクの場合ワード (4byte) 単位で、イベントリンクの場合バイト単位で、エラーのあった場所の Dirty ビットを立てる。エラーがハードウェアによって訂正されても、訂正しきれなくても Dirty ビットは立てるようにする。また、そのパケット中に 1 箇所でもエラー訂正が行われハードウェアで訂正しきれなかった場合、トレイラ部の Correct ビットを立てる。エラー訂正不可能だった場合、Fatal ビットを立てる。受信側のアプリケーションでは、これらを参考にし、例えば、受信データを本当に制御に使用してよいかどうか等を判断することを可能にする。

### 13.8.8 通信速度

*Responsive Link* の通信 (変調) 速度は、様々な環境 (コンフィギュレーション, アプリケーション) を想定し、400, 200, 100, 50, 12.5, 6.25 [Mbaud] の範囲で段階的に可変とする。

表 13.3 に、通信速度と最大通信距離、推奨ケーブルの関係を示す。例えば、最大変調速度 400[Mbaud] で通信する場合、ケーブルには Category6 を使用し、最大通信距離は 60[m] 以内である。この場合、DPLL の基準周波数にはデューティ比が 1 対 1 の 800[MHz] のアップダウンエッジを使用し、サンプリング数 4 で DPLL を行うことによって実現する。

レスポンスプロセッサは組み込み用途を想定しているため、消費電力が大きな問題となる。一般に通信速度（動作周波数）を速くすれば消費電力が大きくなり、遅くすれば小さくなる。通信速度の変更は、受信クロックを変更するのではなく DPLL のサンプリング数を変更することによって行う。従って、通信速度が遅い場合の通信は、通信速度が遅くなることによる安定性の増加と DPLL のサンプリング数が増加することによる安定性の増加という 2 重の恩恵を受ける。

## 13.9 メモリマップ

レスポンスリンク部のアドレスマップは以下の通りである。

デコードアドレス	接続される I/O
0xFFFE_0xxx	レスポンスリンク内部レジスタ
0xFFFE_1xxx	レスポンスリンク用 IRC (r/w)
0xFFFE_2xxx	ルーティングテーブルアドレス部 (r/w)
0xFFFE_3xxx	ルーティングテーブルリンク部 (r/w)
0xC0xx_xxxx	イベント入力用 DPM (r)
0xC4xx_xxxx	イベント出力用 DPM (r/w)
0xC8xx_xxxx	データ入力用 DPM (r)
0xCCxx_xxxx	データ出力用 DPM (r/w)

初期アドレス: 0xfffe0000

## 13.10 レジスタマップ

### 13.10.1 SDRAM モードレジスタ

オフセット: 0x0000

31	30'h0	2 1 0
		SDMODE

*Responsive Link* は、パケット追い越し用に外付けの SDRAM を付けることができる。SDMODE(SDram MODE) レジスタは、パケット追い越し用外付け DDR SDRAM の有無と大きさを示す。外付け SDRAM を搭載しない場合は、内蔵の追い越しバッファ（各リンク 8 パケット分）のみで優先度付きパケットの追越を行う。

bit 名	機能
29'h0	0
SDMODE	Default 000 000 : 外付け SDRAM なし 001 : 外付け SDRAM あり, 容量 : 8MB 010 : 外付け SDRAM あり, 容量 : 16MB 011 : 外付け SDRAM あり, 容量 : 32MB 100 : 外付け SDRAM あり, 容量 : 64MB 101 : 外付け SDRAM あり, 容量 : 128MB 110 : 外付け SDRAM あり, 容量 : 256MB 111 : 外付け SDRAM あり, 容量 : 512MB

### 13.10.2 レスポンシブリンク速度設定レジスタ

オフセット: 0xFFFE\_0004

属性 リード/ライト

31	28	27	25	24	22	21	19	18	16	15	12	11	9	8	6	5	3	2	0
-	Data4	Data3	Data2	Data1	-	Event4	Event3	Event2	Event1										

RSL(*Responsive Link Speed*): Default 000

本レジスタはレスポンシブリンクの変調速度を示す .

- 111 : 800 Mbaud
- 000 : 400 Mbaud
- 001 : 200 Mbaud
- 010 : 100 Mbaud
- 011 : 50 Mbaud

bit 名	機能
Data4	Data Link 4 用 RSL
Data3	Data Link 3 用 RSL
Data2	Data Link 2 用 RSL
Data1	Data Link 1 用 RSL
Event4	Event Link 4 用 RSL
Event3	Event Link 3 用 RSL
Event2	Event Link 2 用 RSL
Event1	Event Link 1 用 RSL

### 13.10.3 レスポンシブリンク初期化レジスタ

オフセット: 0xFFFE\_0008

属性 リード/ライト

31	21 20	17 16 15	5 4	1 0
-	DLINIT	D_s	-	ELINIT
				E_s

RLINIT(*Responsive Link* INITialization) レジスタはレスポンスリンクのスイッチの初期化およびエンコーダ/デコーダ部分の初期化を行なう。

- 0: 通常動作
- 1: 初期化

bit 名	機能
DLINIT	Data link の各エンコーダ/デコーダの初期化 DLINIT[4]: RLINIT[20]: Data link4 の初期化 DLINIT[3]: RLINIT[19]: Data link3 の初期化 DLINIT[2]: RLINIT[18]: Data link2 の初期化 DLINIT[1]: RLINIT[17]: Data link1 の初期化
D_s	Data link switch の初期化
ELINIT	Event link の各エンコーダ/デコーダの初期化 ELINIT[4]: RLINIT[4]: Event link4 の初期化 ELINIT[3]: RLINIT[3]: Event link3 の初期化 ELINIT[2]: RLINIT[2]: Event link2 の初期化 ELINIT[1]: RLINIT[1]: Event link1 の初期化
E_s	Event link switch の初期化

#### 13.10.4 レスポンスリンク割り込みクリアレジスタ

オフセット: 0xFFFE\_000C 属性 リード/ライト

31	7 6	1 0
-	RLIC	-

RLIC(*Responsive Link* Irq Clear) レジスタはイベントリンクの割り込み要求のクリアを行なう。

Default 0

- 0: 通常動作
- 1: クリア

bit 名	機能
RLIC[1]	Data-Out EOP(End Of Packet) IRQ Clear: データパケットが DPM の設定した範囲から送信された場合に生じる割り込みのクリア
RLIC[2]	Event-Out EOP IRQ Clear: イベントパケットが DPM の設定した範囲から送信された場合に生じる割り込みのクリア
RLIC[3]	Data-In EOP IRQ Clear: データパケットが DPM の設定した範囲に受信された場合に生じる割り込みのクリア
RLIC[4]	Event-In EOP IRQ Clear: イベントパケットが DPM の設定した範囲に受信された場合に生じる割り込みのクリア
RLIC[5]	Data Packet-In IRQ Clear: 割り込みビットの設定されたデータパケットが到着した場合に生じる割り込みのクリア
RLIC[6]	Event Packet-In IRQ Clear: 割り込みビットの設定されたイベントパケットが到着した場合に生じる割り込みのクリア

### 13.10.5 レスポンシブリンク送信停止割り込みクリアレジスタ

オフセット: 0xFFFE\_0010 属性 リード/ライト

31	21 20	16 15	5 4	0
-	DWIRQC	-	EWIRQC	

*Responsive Link* はパケット追い越し用 SDRAM を使用している際には追い越し用 SDRAM が溢れそうになると送信停止割り込みを自動生成する。同様に、追い越し用 SDRAM を使用していない際には、追い越し用バッファが溢れそうになると送信停止割り込みを自動生成する。本 WIRQC(Wait IRQ Clear) レジスタはレスポンシブリンク送信停止割り込み要求のクリアを行なう。

Default 0

- 0: 通常動作
- 1: クリア

bit 名	機能
DWIRQC	Data link WIRQC DWIRQC[4]: WIRQC[20]: Data link4 DWIRQC[3]: WIRQC[19]: Data link3 DWIRQC[2]: WIRQC[18]: Data link2 DWIRQC[1]: WIRQC[17]: Data link1 DWIRQC[0]: WIRQC[16]: Data link0(CPU)
EWIRQC	Event link WIRQC EWIRQC[4]: WIRQC[4]: Event link4 EWIRQC[3]: WIRQC[3]: Event link3 EWIRQC[2]: WIRQC[2]: Event link2 EWIRQC[1]: WIRQC[1]: Event link1 EWIRQC[0]: WIRQC[0]: Event link0(CPU)

### 13.10.6 レスポンシブリンク継続割り込みクリアレジスタ

オフセット: 0xFFFE\_0014 属性 リード/ライト

31	21 20	16 15	5 4	0
-	DCIC	-	ECIC	

*Responsive Link* は、SDRAM に退避されたパケットがスイッチに書き戻された（再度送信された）際にレスポンシブリンク継続割り込み CI(Continuous Irq) を発生する。CIC(Continuous Irq Clear) レジスタはその割り込み要求 CI のクリアを行なう。

Default 0

0: 通常動作

1: クリア

bit 名	機能
DCIC	Data CIC DCIC[4]: CIC[20]: Data link4 DCIC[3]: CIC[19]: Data link3 DCIC[2]: CIC[18]: Data link2 DCIC[1]: CIC[17]: Data link1 DCIC[0]: CIC[16]: Data link0(CPU)
ECIC	Event CIC ECIC[4]: CIC[4]: Event link4 ECIC[3]: CIC[3]: Event link3 ECIC[2]: CIC[2]: Event link2 ECIC[1]: CIC[1]: Event link1 ECIC[0]: CIC[0]: Event link0(CPU)

### 13.10.7 レスポンシブリンク致命的エラー割り込みクリアレジスタ

オフセット: 0xFFFE\_0018 属性 リード/ライト

31	21 20	16 15	5 4	0
-	DFIC	-	EFIC	

Responsive Link は、各リンクの受信パケットにハードウェアで回復不可能なエラーがあった場合にレスポンシブリンク致命的エラー割り込み FI(Fatal Irq) を発生する。FIC(Fatal Irq Clear) レジスタは、その割り込み要求 FI のクリアを行なう。

Default 0

- 0: 通常動作
- 1: クリア

bit 名	機能
DFIC	Data FIC DFIC[4]: FIC[20]: Data link4 DFIC[3]: FIC[19]: Data link3 DFIC[2]: FIC[18]: Data link2 DFIC[1]: FIC[17]: Data link1 DFIC[0]: FIC[16]: Data link0(CPU)
EFIC	Event FIC EFIC[4]: FIC[4]: Event link4 EFIC[3]: FIC[3]: Event link3 EFIC[2]: FIC[2]: Event link2 EFIC[1]: FIC[1]: Event link1 EFIC[0]: FIC[0]: Event link0(CPU)

### 13.10.8 レスポンシブリンクルーティングテーブル割り込みクリアレジスタ

オフセット: 0xFFFE\_001C 属性 リード/ライト

31	2 1 0
-	RTIRQC

*Responsive Link* は、ルーティングテーブルにマッチするエントリが無かった場合にレスポンシブリンクルーティングテーブル割り込み (RTIRQ) を発生する。RTIRQC(Routing Table IRQ Clear) レジスタは、その割り込み要求 RTIRQ のクリアを行なう。

Default 0

- 0: 通常動作 (r) / 割り込みクリア (w)
- 1: 割り込み状態 (r) / 割り込み発生 (デバッグ用) (w)

bit 名	機能
RTIC[0]	Event Routing Table IRQ Clear
RTIC[1]	Data Routing Table IRQ Clear

### 13.10.9 レスポンシブリンク SDRAM バスリクエストレジスタ

オフセット: 0xFFFE\_0020 属性 リード/ライト

31	1 0
-	RLSDBREQ

*Responsive Link* の追い越し用 SDRAM のバスには、*Responsive Link* とプロセッサバスの 2 つのバスマスタが接続されている。通常、プロセッサ側から追い越し用 SDRAM にアクセスする際には、データのトランザクション毎に、バス権の調停が行われている。プロセッサ側からバースト的に追い越し用 SDRAM をアクセスしたい場合には、本ビットを有効にすることで、追い越し用 SDRAM バスのバス権をプロセッサ側（プロセッサや DMAC 等）が常に得ることができる（本ビットを設定しなくてもアクセス可能である。）*Responsive Link* 側が追い越し用 SDRAM バスを参照できなくなる（パケットの退避・復帰ができなくなる）という副作用がある。

bit 名	機能
RLSDBREQ	RLSDBREQ ( <i>Responsive Link</i> SDram-Bus REQuest) : Default 1 本ビットはレスポンスリンクの SDRAM バスへの明示的なバスリクエストを行なう。 0: バスリクエストイネーブル 1: バスリクエストディスエーブル

### 13.10.10 レスポンスリンク SDRAM バスグラントレジスタ

オフセット: 0xFFFE\_0024 属性 リード/ライト

31	30	21	20	16	15	5	4	0
MSG	-			DSG		-		ESG

RLSDBGNT(*Responsive Link* SDram Bus GRaNT) レジスタは、追い越し用 SDRAM バスのバスグラント（どのバスマスタがバス権を有しているか）を示す。

- 0: バス権獲得
- 1: バス権開放

bit 名	機能
MSG	Mpu Sdram bus Grant: MPU がバス権を得ている
DSG	Data link Sdram bus Grant: Data Link がバス権を得ている DSG[4]: RLSDBGRNT[20]: Data link4 DSG[3]: RLSDBGRNT[19]: Data link3 DSG[2]: RLSDBGRNT[18]: Data link2 DSG[1]: RLSDBGRNT[17]: Data link1 DSG[0]: RLSDBGRNT[16]: Data link0(CPU)
ES	Event link Sdram bus Grant: Event Link がバス権を得ている ESG[4]: RLSDBGRNT[4]: Event link4 ESG[3]: RLSDBGRNT[3]: Event link3 ESG[2]: RLSDBGRNT[2]: Event link2 ESG[1]: RLSDBGRNT[1]: Event link1 ESG[0]: RLSDBGRNT[0]: Event link0(CPU)

### 13.10.11 レスポンシブリンクルーティングテーブルバスリクエストレジスタ

オフセット: 0xFFFE\_0028 属性 ライト

31	-	1 0
		BRQ

*Responsive Link* のルーティングテーブルのバスには、*Responsive Link* とプロセッサバスの 2 つのバスマスタが接続されているが、デフォルトのバスマスタは *Responsive Link* である。プロセッサ側からルーティングテーブルをアクセスしたい場合には、本ビットを有効にすることで、ルーティングテーブルバスのバス権をプロセッサ側（プロセッサや DMAC 等）が得ることができる。*Responsive Link* 側がルーティングテーブルを参照できなくなる（パケットのルーティングができなくなる）という副作用がある。

bit 名	機能
BRQ	RLTBLBREQ ( <i>Responsive Link</i> routing TaBLe Bus REQuest): Default 1 本ビットはレスポンシブリンクのルーティングテーブルバスへのバスリクエストを行なう。 0: バスリクエストイネーブル 1: バスリクエストディスエーブル

オフセット: 0xFFFE\_0028 属性 リード

31 30	21 20	16 15	5 4	0
MRR	-	DRR	-	ERR

本ビットはプロセッサバス側からレスポンスリンクのルーティングテーブルバスへのバスリクエストを示す。  
 0: バスリクエスト有  
 1: バスリクエスト無

bit 名	機能
MRR	Mpu Routing table bus Request
DRR	Data link Routing table bus Request DRR[4]: RLTBLBREQ[20]: Data link4 DRR[3]: RLTBLBREQ[19]: Data link3 DRR[2]: RLTBLBREQ[18]: Data link2 DRR[1]: RLTBLBREQ[17]: Data link1 DRR[0]: RLTBLBREQ[16]: Data link0(CPU)
ER	Event link Routing table bus Request ERR[4]: RLTBLBREQ[4]: Event link4 ERR[3]: RLTBLBREQ[3]: Event link3 ERR[2]: RLTBLBREQ[2]: Event link2 ERR[1]: RLTBLBREQ[1]: Event link1 ERR[0]: RLTBLBREQ[0]: Event link0(CPU)

### 13.10.12 レスポンスリンクルーティングテーブルバスグラントレジスタ

オフセット: 0xFFFE\_002C 属性 リード

31 30	21 20	16 15	5 4	0
MRRG	-	DRG	-	ERG

RLTBLBGRNT (*Responsive Link routing TaBLE Bus GRaNT*) レジスタは、レスポンスリンクのルーティングテーブルバスのバスグラント(どのバスマスタがバス権を有しているか)を示す。

- 0: バス権獲得
- 1: バス権開放

bit 名	機能
MRG	Mpu Routing table bus Grant: MPU がバス権を得ている
DRG	Data link Routing table bus Grant: Data Link がバス権を得ている DRG[4]: RLTBLBGRNT[20]: Data link4 DRG[3]: RLTBLBGRNT[19]: Data link3 DRG[2]: RLTBLBGRNT[18]: Data link2 DRG[1]: RLTBLBGRNT[17]: Data link1 DRG[0]: RLTBLBGRNT[16]: Data link0(CPU)
ERG	Event link Routing table bus Grant: Event Link がバス権を得ている ERG[4]: RLTBLBGRNT[4]: Event link4 ERG[3]: RLTBLBGRNT[3]: Event link3 ERG[2]: RLTBLBGRNT[2]: Event link2 ERG[1]: RLTBLBGRNT[1]: Event link1 ERG[0]: RLTBLBGRNT[0]: Event link0(CPU)

### 13.10.13 イベントリンク LRU アドレスレジスタ

オフセット: 0xFFFE\_0030 属性 リード

31	-	10 9	0
		ELLRUA	

bit 名	機能
ELLRUA	ELLRUA (Event Link LRU Address) レジスタはイベントリンクのルーティングテーブル中で、最も近くに使用されたテーブルの格納されているアドレスを示す。

### 13.10.14 データリンク LRU アドレスレジスタ

オフセット: 0xFFFE\_0034 属性 リード

31	-	10 9	0
		DLLRUA	

bit 名	機能
DLLRUA	DLLRUA (Data Link LRU Address) レジスタはデータリンクのルーティングテーブル中で、最も近くに使用されたテーブルの格納されているアドレスを示す。

## 13.10.15 レスポンシブリンク用割り込みコントローライネーブルレジスタ

オフセット: 0xFFFE\_0038 属性 リード

31	-	1 0
		RLICE

bit 名	機能
RLICE	RLICE ( <i>Responsive Link</i> Interrupt Controller Enable) レジスタはレスポンシブリンク用割り込みコントローラ RLIRC のイネーブルビットを示す．1 のとき，RLIRC は出力を行っている．

## 13.10.16 イベントリンク用 SDRAM ループカウントレジスタ

オフセット: 0xFFFE\_0040 属性 リード/ライト

31	-	8 7	0
		ELSDCNT	

追い越し用 SDRAM に退避されたイベントパケットを *Responsive Link* イベントスイッチに再度送信してよいかどうかを調べる間隔を指定する．短すぎると消費電力が大きくなり，長すぎるとリアルタイム性が損なわれる．

bit 名	機能
ELSDCNT	ELSDCNT (Event Link SDram loop CouNTer) レジスタの設定により，追い越し用 SDRAM に退避されているイベントパケットをイベントスイッチに再送しようとするリトライの間隔を 1 パケット分の送信時間を単位として指定する．(1 - 40) Default: 32

## 13.10.17 データリンク用 SDRAM ループカウントレジスタ

オフセット: 0xFFFE\_0044 属性 リード/ライト

31	-	4 3	0
		DLSDCNT	

追い越し用 SDRAM に退避されたデータパケットを *Responsive Link* データスイッチに再度送信してよいかどうかを調べる間隔を指定する．短すぎると消費電力が大きくなり，長すぎるとリアルタイム性が損なわれる．

bit 名	機能
DLSDCNT	DLSDCNT (Data Link SDram loop CouNter) レジスタの設定により、追い越し用 SDRAM に退避されているデータパケットをデータスイッチに再送しようとするリトライの間隔を 1 パケット分の送信時間を単位として指定する。(1 - 95) Default: 4

### 13.10.18 レスポンスリンクスイッチモードレジスタ

オフセット: 0xFFFE\_0048 属性 リード/ライト

31	2 1 0
-	RLSM

RLSM(*Responsive Link* Switch Mode) レジスタの設定により、レスポンスリンクのスイッチの動作を変更する。

- 0: Cut Through Mode      レイテンシ的に有利であるがパケットの追い越しをしにくい
  - 1: Store and Forward Mode      レイテンシ的に不利であるがパケットの追越ししやすい
- Default: 0

bit 名	機能
RLSM[0]	Event Link Switch の設定
RSLM[1]	Data Link Switch の設定

### 13.10.19 レスポンスリンク用オフラインレジスタ

オフセット: 0xFFFE\_004c 属性 リード

31	21 20	16 15	5 4	0
-	DRLOL	-	ERLOL	

*Responsive Link* は Plug&Play をサポートするために、リンクアップしていたリンクがリンクダウンするとオフライン割り込みを発生し、リンクダウンしていたリンクがリンクアップするとオンライン割り込みを発生する。

RLOL(*Responsive Link* OffLine) レジスタをリードすることにより、どのリンクがオフライン/オンラインかを調べることができる。

- 1: Offline
- 0: Online

bit 名	機能
DRLOL	Data link の RLOL レジスタ DRLOL[4]: RLOL[20]: Data link4 DRLOL[3]: RLOL[19]: Data link3 DRLOL[2]: RLOL[18]: Data link2 DRLOL[1]: RLOL[17]: Data link1 DRLOL[0]: RLOL[16]: Data link0(CPU)
ERLOL	Event link の RLOL レジスタ ERLOL[4]: RLOL[4]: Event link4 ERLOL[3]: RLOL[3]: Event link3 ERLOL[2]: RLOL[2]: Event link2 ERLOL[1]: RLOL[1]: Event link1 ERLOL[0]: RLOL[0]: Event link0(CPU)

オフセット: 0xFFFE\_004c 属性 ライト

31	2 1 0
-	RLOL

本ビットの設定により、レスポンスリンクのオフライン割り込み及びオンライン割り込みをクリアできる。

- 1: 割り込みクリアを行わない
- 0: 割り込みクリア

bit 名	機能
RLOL[0]	<i>Responsive Link</i> Down IRQ Clear: オフライン割り込みのクリア
RLOL[1]	<i>Responsive Link</i> Wakeup IRQ Clear: オンライン割り込みのクリア

## 13.11 DPM (Dual Port Memory)

*Responsive Link* とプロセッサは基本的に DPM を介してデータの送受信を行う。DPM はその名の通り 2port を有しており、片方はプロセッサバスに接続され、もう片方は *Responsive Link* の Link0 に接続されている。

Data in/out control register および、Event in/out control register を設定することでパケットの送信/受信方法を決定することができる。イベントパケットの送信および、受信には Event packet in/out 専用の DPM を用い、データパケットの送信および、受信には Data packet in/out 専用の DPM を使用する。

以下に Event in/out , Data in/out それぞれの DPM について説明する。

### 13.11.1 Event Output

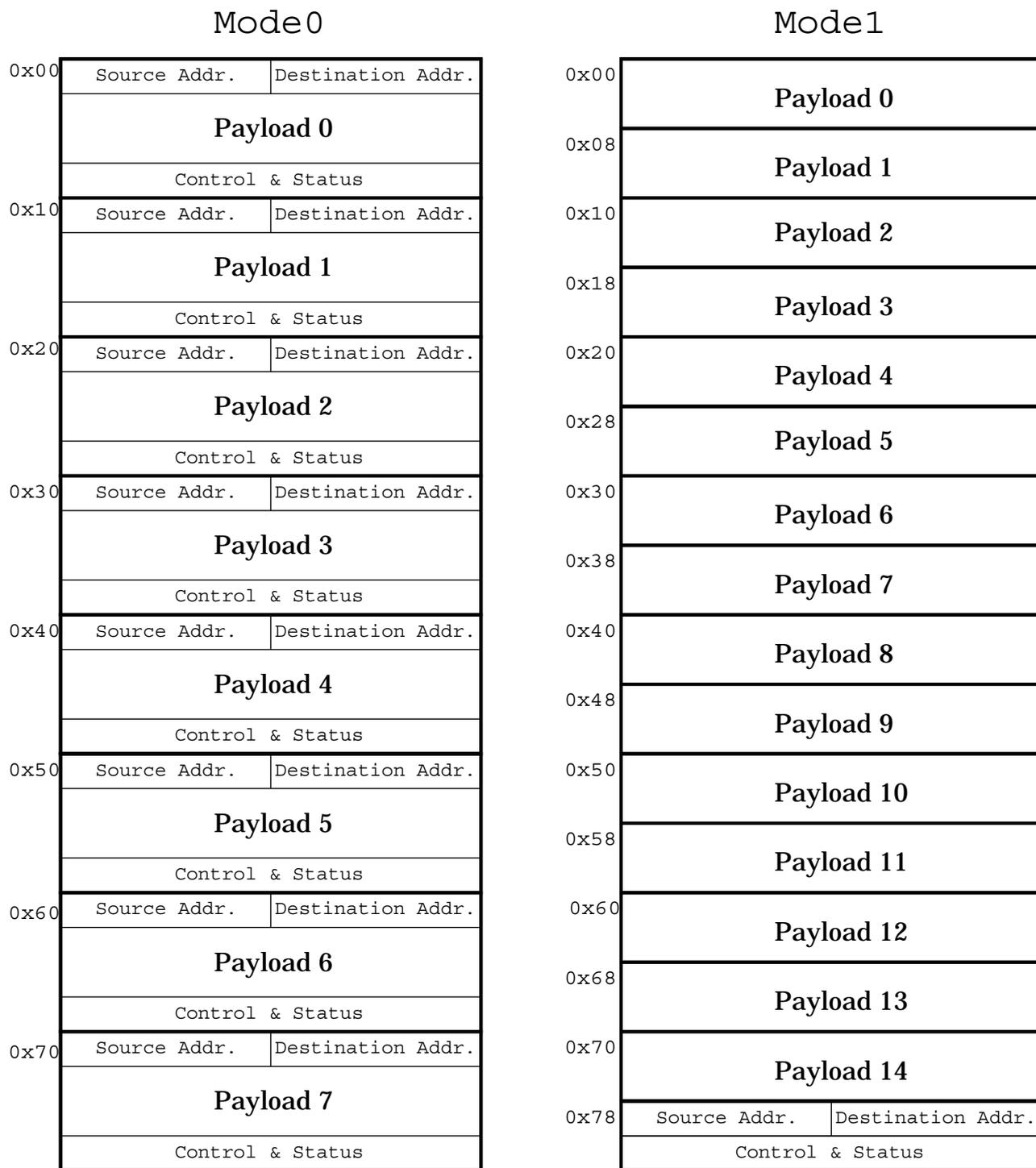
図 13.9 にイベントリンク出力用 DPM の構成を示す。Event out control register (図 13.10 参照) に対して、開始アドレス From\_Addr (byte address ではなく word address) と終了アドレス To\_Addr (word address) を設定することにより、複数パケットを一度に送信できる。From\_Addr と To\_Addr は人間に分かりやすいようにこのような名前を付けられているが、実際には全く同じ機能のレジスタが二つ用意されている。From\_Addr, To\_Addr 共に、設定された word address - 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に、DPM から Link0 に対して出力を開始する。

例えば、Mode0 を使用し、From\_Addr を 0x00 に設定し To\_Addr を 0x07 (byte address: 0x1c) に設定したとする。プロセッサバス側から DMAC もしくはプロセッサによって Payload0, Payload1 の順に DPM にデータが書かれたとすると、DPM のプロセッサ側から 0x06 番地にデータが書かれた瞬間に DPM から *Responsive Link* の Link0 に出力を開始する (この場合、実際には From\_Addr には意味がない。)

あるいは、Mode0 を使用し、From\_Addr を 0x1f (byte address 0x3c) に設定し To\_Addr を 0x2f (byte address: 0x7c) に設定し、さらに DMAC を continuous mode で使用すると、Payload0 ~ 3 の領域と Payload4 ~ 7 の領域を使用して、主記憶等に用意した DPM よりも大きな連続データをハードウェアのみで自動送信することができる (DPM のアドレスデコードの範囲内では、シャドウアドレスでも CS が生成され DPM にアクセスできるように設計しているため。)

offset address  
0xC400\_00XX

### DPM for Event Output



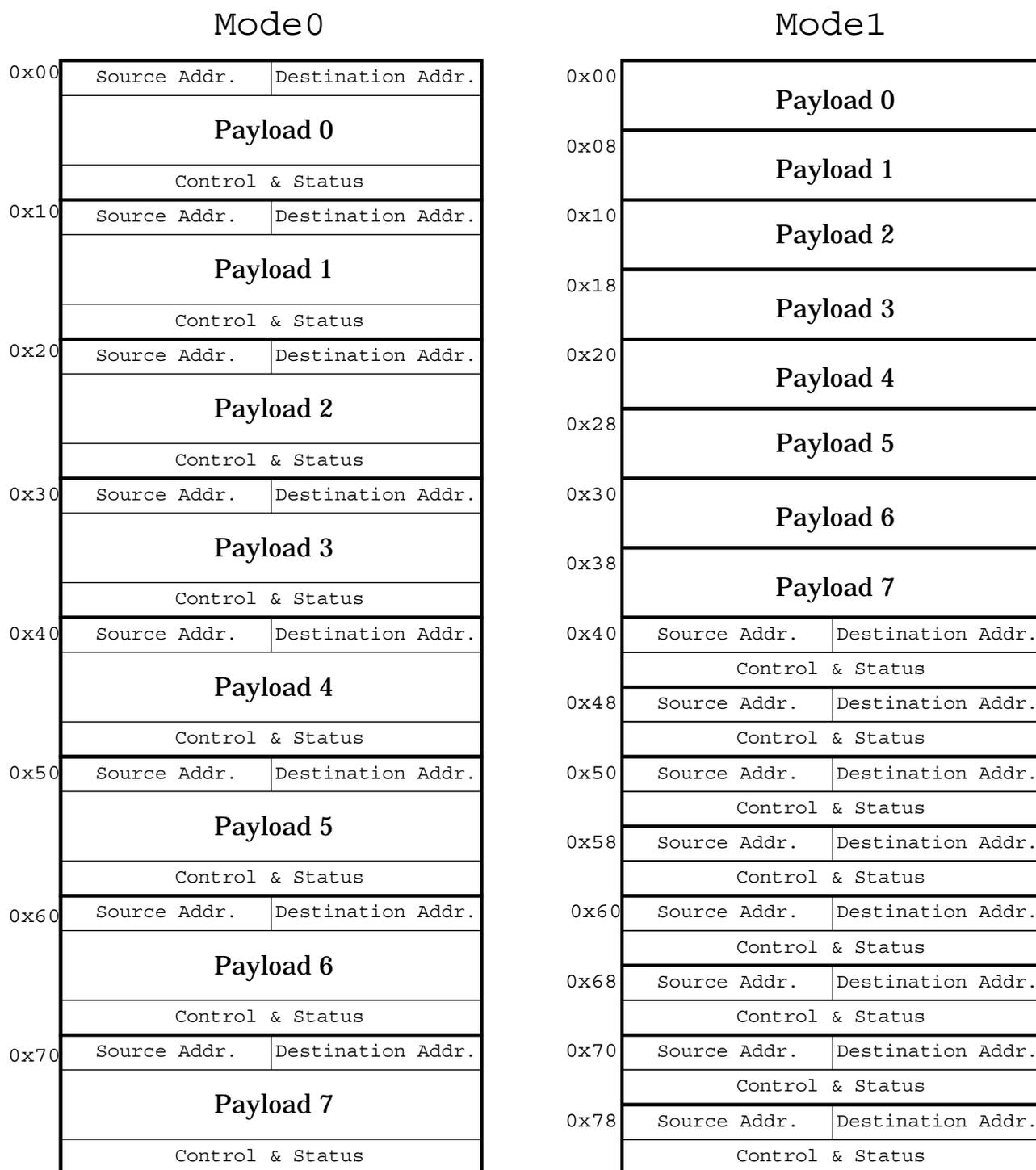
☒ 13.9: DPM for Event Output



あるいは、Mode0 を使用し、From\_Addr を 0x1f(byte address 0x3c) に設定し To\_Addr を 0x2f(byte address: 0x7c) に設定し、さらにDMACを continuous mode で使用すると、Payload0～3の領域とPayload4～7の領域を使用して、主記憶等に用意したDPMよりも大きなメモリ領域(サイクリックバッファ等)に対して、受信データをハードウェアのみで連続的に自動受信することができる。(DPMのアドレスデコードの範囲内では、シャドウアドレスでもCSが生成されDPMにアクセスできるように設計しているため。)

offset address  
0xC000\_00XX

## DPM for Event Input



☒ 13.11: DPM for Event Input



DPM にデータが書かれたとすると、DPM のプロセッサ側から word address 0x01e 番地にデータが書かれた瞬間に DPM から *Responsive Link* の Link0 に出力を開始する（この場合、実際には From\_Addr には意味がない。）

あるいは、Mode0 を使用し、From\_Addr を 0x0ff(byte address 0x3fc) に設定し To\_Addr を 0x1ff(byte address: 0x7fc) に設定し、さらに DMAC を continuous mode で使用すると、Payload0~15 の領域と Payload16~31 の領域を使用して、主記憶等に用意した DPM よりも大きな連続データをハードウェアのみで自動送信することができる（DPM のアドレスデコードの範囲内では、シャドウアドレスでも CS が生成され DPM にアクセスできるように設計しているため。）

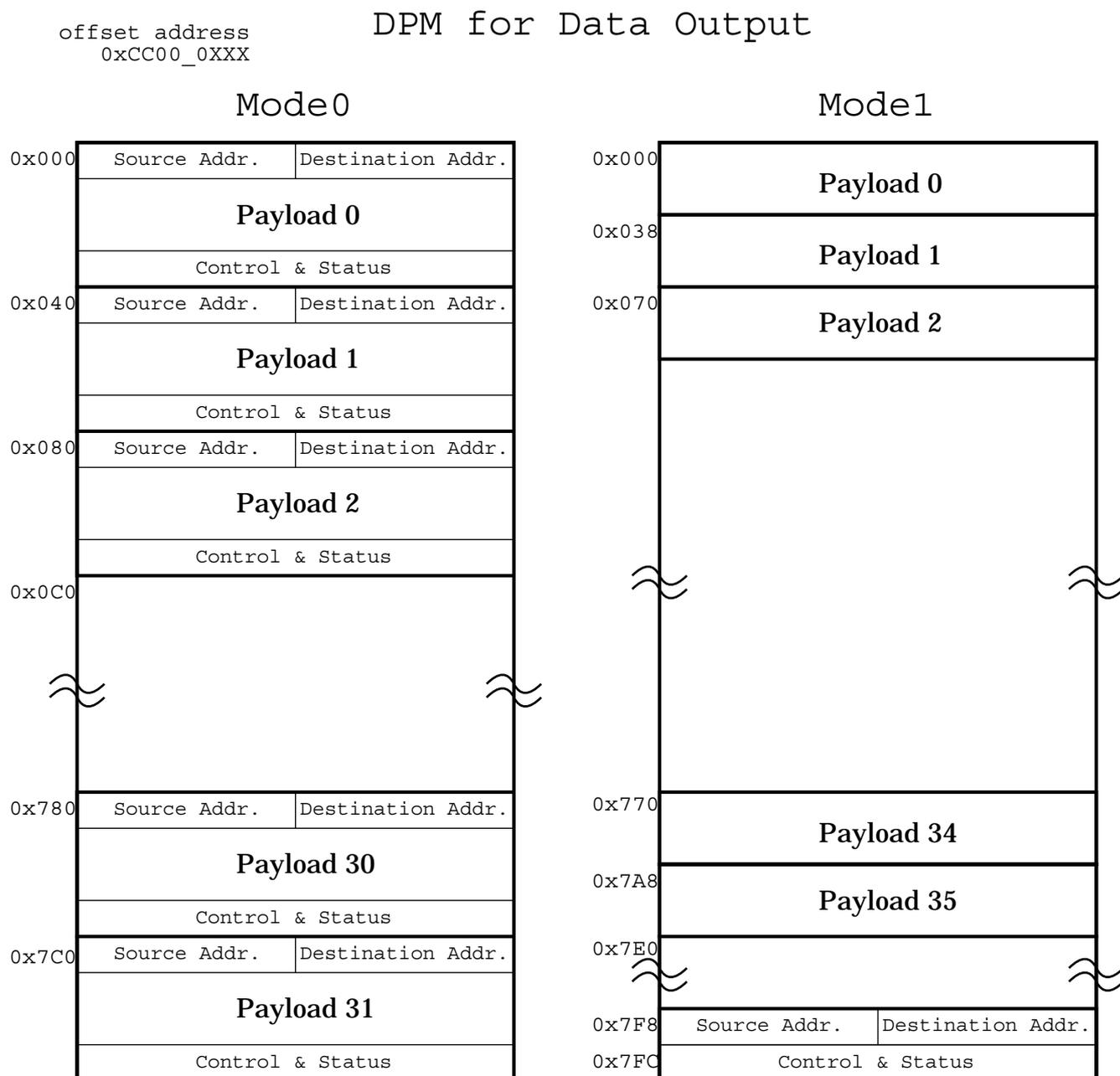


図 13.13: DPM for Data Output

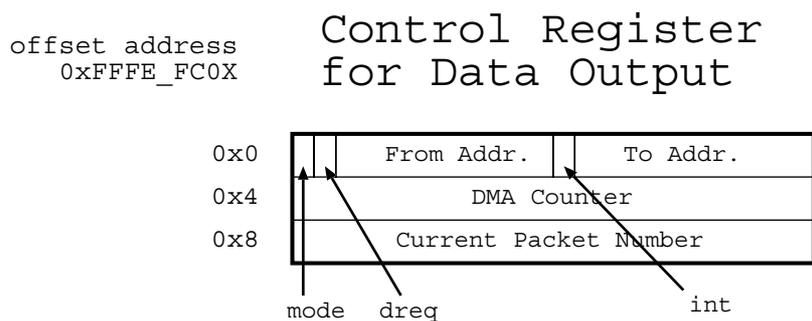


図 13.14: Data Out Control Register

## DPM 制御レジスタ

DPM の制御レジスタ (図 13.14 参照) に以下を設定することで、送信の制御を行う。

### 制御レジスタ (r/w)

- Mode0: (r/w) mode bit に 0 を設定。すべてのパケットに headr と trailer を付加する。
- Mode1: (r/w) mode bit に 1 を設定。最後に共通の header と trailer を付加する (すべてのパケットの宛先が同一となる)。
- Int: (r/w) 本ビットを 1 に設定すると、終了時に EOP(End Of Packet) 割り込みを生成する。
- Dreq: (r/w) 本ビットを 1 に設定すると、DMA Counter に設定した回数分だけ DMA を行う。
- From\_Addr: (r/w) 設定された word address - 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に DPM から Link0 に対して出力を開始する。
- To\_Addr: (r/w) 設定された word address - 1 のアドレスに DPM のプロセッサバス側からデータが書かれた瞬間に DPM から Link0 に対して出力を開始する。

### DMA Counter (r/w) DMA の回数を指定する

Current Packet Number (r) 現在送信されているパケット番号 (図 13.13 の payload 番号に相当) を示す

## 13.11.4 Data Input

図 13.15 にデータリンク入力用 DPM の構成を示す。Data in control register (図 13.16 参照) に対して、開始アドレス From\_Addr (byte address ではなく word address) と終了アドレス To\_Addr (word address) を設定することにより、複数パケットを一度に受信できる。From\_Addr と To\_Addr は人間に分かりやすいようにこのような名前を付けられているが、実際には全く同じ機能のレジスタが二つ用意されている。From\_Addr, To\_Addr 共に、設定された word address - 1 のアドレスに DPM の *Responsive Link* 側からデータが書かれた瞬間に、DPM からプロセッサバス側に対して出力 (DMA 転送) を開始する (dreq bit が設定されている場合)。int bit の割り込みを利用して、ソフトウェアで受信することもできる。

例えば、Mode0 を使用し、From\_Addr を 0x000 に設定し To\_Addr を 0x01f (byte address: 0x07c) に設定したとする。*Responsive Link* 側から Payload0, Payload1,... の順に DPM に受信データが書かれていく。

Responsive Link 側から DPM の word address 0x1e 番地にデータが書かれた瞬間に DPM からプロセッサバス側へ出力 (DMA 転送) を開始する (この場合, 実際には From\_Addr には意味がない.)

あるいは, Mode0 を使用し, From\_Addr を 0x0ff(byte address 0x3fc) に設定し To\_Addr を 0x1ff(byte address: 0x7fc) に設定し, さらに DMAC を continuous mode で使用すると, Payload0~15 の領域と Payload16~31 の領域を使用して, 主記憶等に用意した DPM よりも大きなメモリ領域 (サイクリックバッファ等) に対して, 受信データをハードウェアのみで連続的に自動受信することができる (DPM のアドレスデコードの範囲内では, シャドウアドレスでも CS が生成され DPM にアクセスできるように設計しているため.)

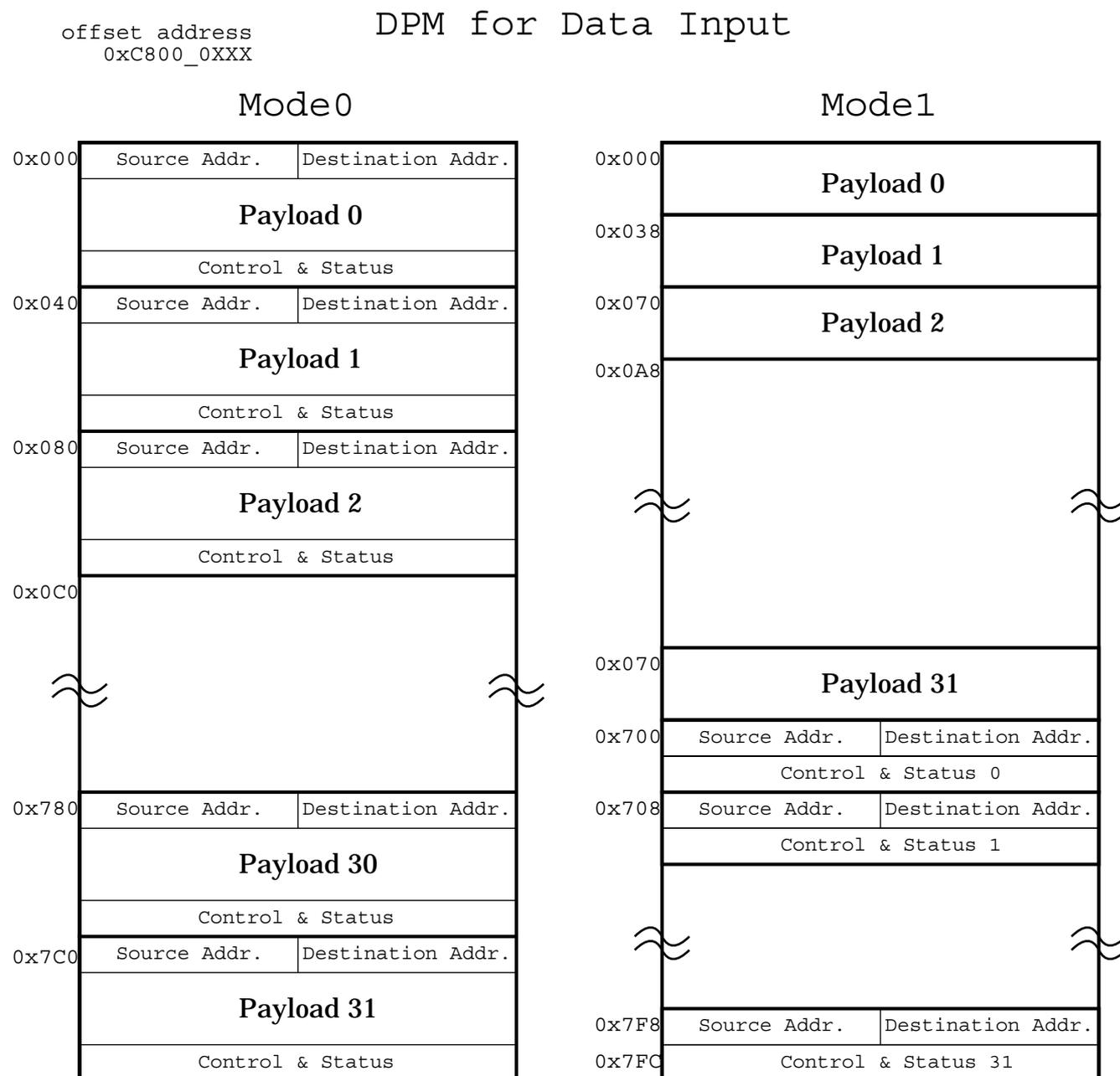


図 13.15: DPM for Data Input



## DMA を用いた送信

DPM の容量には当然限界がある。しかし、レスポンスリンクでは、DMA と DPM が協調して動作することで、DPM の容量を越えるような大きなデータを一度に送信することが可能である。その際の手順は以下の通りである。ただし、総データ量は  $N$  packet 分であることを仮定する。

### DPM の設定

1.  $N$  の約数のうち最大のものを  $f$  とする。ただし、データリンクでは  $f_{i36}$ 、イベントリンクでは  $f_{i15}$  であるとする。
2. DPM の DMA Counter を  $(N/f)-1$  に設定する。
3. DPM の MODEL\_HEADER 及び MODEL\_TRAILER に宛先およびパケットの持つべき性質 (受信側での割込みなど) を設定する。
4. DPM のコントロールレジスタを mode 1, from(0), to( $f*0xe+0xd$ ), DREQ に設定する。(mode 0 で送信する場合、DMA の転送元にはパケットの形でデータが存在している必要がある。ただし、この場合手順 3 は必要無い。)

### DMA の設定

1. DMA の送信元を送信したいデータの格納されているメモリの先頭アドレスに設定する。
2. DMA の送信先を送信用 DPM の先頭のアドレスにする。
3. DMA の送信先を送信用 DPM の先頭のアドレスにする。
4. DMA のコントロールレジスタは SAU, RL, MTM, ST を ON にする。(この ST による起動が DMA Counter の設定時に差し引いた 1 回に相当する)

## 13.12.2 相互通信の際の注意点

相互通信をする際に注意すべきは、二つのボードをつなげてから、まずそれぞれのボードにおいて「通信速度の設定」を行い、さらに、それぞれのボードにおいて「リンクの初期化」を行う。

当然、通信速度は同じでなければならない。また、リンクの初期化はボードをつなげてから行わないと相互通信ができないので注意すること。その他の設定は個別に各ボードで行う (モニタでの相互通信には、リンクの初期化モジュールを作成し、ボードをつなげたあと、そのモジュールを実行することにより相互通信を行う。)

# 14

## DMAC

- 32/16/8 bit I/F
- 入力チャネル：4
- 優先順位：固定優先度及びラウンドロビン
- Memory to memory 転送機能
- Bus sizing 機能 (8, 16bit I/O 用)
- Bus swapping 機能 (8, 16bit I/O 用)

### 14.1 レジスタマップ

DMAC	初期アドレス
DMAC0	FFFF0000
DMAC1	FFFF1000
DMAC2	FFFF2000

offset	31	24	23	16	15	8	7	0
0x800	-							PRI
0x804	-							IC
0x40*(x)+0x04	PSA<31:0>							
0x40*(x)+0x08	MDA<31:0>							
0x40*(x)+0x18	LN<31:0>							
0x40*(x)+0x0c	ID<31:0>							
0x40*(x)+0x10	-							DA S AUBM RL PCMTMR32P16P8P S16 S8 IERIED ST
0x40*(x)+0x14	L0	L1	L2	L3	-			ER ED

## 14.1.1 DMA 制御レジスタ

ライト/リード  
オフセット: 0x800

31	-	1 0
		PRI

bit 名	機能
PRI	PRiority :Default 0 本ビットは DMA チャンネルのプライオリティを示す 0:プライオリティはラウンドロビン 1:プライオリティは ch0>ch1>ch2>ch3

## 14.1.2 DMA 割り込みクリアレジスタ

オフセット: 0x804

31	-	1 0
		IC

bit 名	機能
IC	Interrupt Clear 本ビットは DMA 割り込みのクリアを行う。0:割り込みクリア

## 14.1.3 ポート/ソースアドレスレジスタ

オフセット: 0x40\*(x) +0x04

31	PSA<31:0>	0
----	-----------	---

bit 名	機能
PSA<31:0>	Port/Source Address :Default X チャンネル x の DMA に対し、本ビットはメモリから I/O への転送の時 (MODE レジスタの MTM ビットが 0) ポートアドレスを示し、メモリからメモリへの転送の時 (MODE レジスタの MTM ビットが 1) ソースアドレスを示す。

## 14.1.4 メモリ/デスティネーションアドレスレジスタ

オフセット:  $0x40*(x) + 0x08$ 

31	MDA<31:0>	0
----	-----------	---

bit 名	機能
MDA<31:0>	Memory/Destination Address :Default X チャンネル x の DMA に対し, 本ビットはメモリから I/O への転送の時 (MODE レジスタの MTM ビットが 0) メモリアドレスを示し, メモリからメモリへの転送の時 (MODE レジスタの MTM ビットが 1) デスティネーションアドレスを示す.

## 14.1.5 転送レングスレジスタ

オフセット:  $0x40*(x) + 0x18$ 

31	LN<31:0>	0
----	----------	---

bit 名	機能
LN<31:0>	transfer LeNght :Default X チャンネル x の DMA に対し, 本レジスタは転送レングスを示す. 単位はバイトである.

## 14.1.6 データバッファレジスタ

オフセット:  $0x40*(x) + 0x0c$ 

31	ID<31:0>	0
----	----------	---

bit 名	機能
ID<31:0>	Internal Data :Default X チャンネル x の DMA に対し, DMA 転送時一度内部のデータバッファにてメモリにライトするデータを組み立てるが, そのデータバッファの値が読める. どのロケーションに有効なデータがあるかはステータスレジスタを見る必要がある.



bit 名	機能
DAS	Destination Address Update :Default X 0:メモリアドレスレジスタで設定したアドレスが次の転送にも使用される． 1:メモリアドレスレジスタの値は，最後に転送を行ったアドレスより1ワード先を示す．
SAU	Source Address Update :Default X 0:ポートアドレスレジスタで設定したアドレスが次の転送にも使用される． 1:ポートアドレスレジスタの値は，最後に転送を行ったアドレスより1ワード先を示す．
BM	Burst Mode :Default X 0:バースト転送しない． 1:バースト転送する．
RL	Responsive Link :Default X 1:レスポンスリンク用 DPM に対する DMA 転送を行う．
PCI	PCI :Default X 1:PCI に対して DMA 転送を行う．
MTM	Memory To Memory transfer :Default X 0:ポートアドレスレジスタで設定した I/O とメモリ間の DMA 転送であることを示す．転送方向は MR ビットにて指定する． 1:ソースアドレスからデスティネーションアドレスへ，レンジレジスタで設定したバイト数のデータを DMA 転送を行う．アドレスカウンタは UP 方向のみのカウントとする．また，4 バイトバウンダリでない転送領域及びレンジの DMA 転送は Memory To Memory ではサポートしない．
MR	MR Memory Read :Default X 0:I/O からメモリへの転送であることを示す． 1:メモリから I/O への転送であることを示す．
32P	32bit I/O Port :Default X 0:don't care 1:MTM ビットが 0 の時 32bit の I/O ポートとの転送であることを示す．この時，ポートアドレスのビット 1, 0 は無視される．
16P	16P 16bit I/O Port :Default X 0:don't care 1:MTM ビットが 0 の時 16bit の I/O ポートとの転送であることを示す．この時，ポートアドレスのビット 0 は無視され，ビット 1 によりどのデータバスに接続されるか (D31-16 or D15-0) を示す．
8P	8P 8bit I/O Port :Default X 0:don't care 1:MTM ビットが 0 の時 8bit の I/O ポートとの転送であることを示す．この時，ポートアドレスのビット 1, 0 によりどのデータバスに接続されるか (D31-24 or D23-16 or D15-8 or D7-0) を示す．
S16	Swap at 16bit :Default X 0:don't care 1: 16bit 単位でデータのスワップを行う． 31   A   B   C   D   0    31   C   D   A   B   0
S8	Swap at 8bit :Default X 0:don't care 1: 8bit 単位でデータのスワップを行う． 31   A   B   C   D   0    31   B   A   D   C   0 S16=1,S8=1 をセットすると以下のようにスワップされる． 31   A   B   C   D   0    31   D   C   B   A   0
IER	Interrupt enable of ER-bit :Default 0 0:割り込みを発生しない． 1:割り込み発生を許可する．
IED	Interrupt enable of ED-bit :Default 0 0:割り込みを発生しない． 1:割り込み発生を許可する．
ST	Start :Default 0 0:DMA 転送を停止させる．0 をライト後 DMAC は初期化される． 1:DMA 転送を起動する．

本レジスタで設定できるモードは次ページの通りであり，それ以外の設定では動作の保証はしない．

転送モード		スワップなし (S16=0,S8=0)	スワップあり (S16=0,S8=1)	スワップあり (S16=1,S8=0)	リトルエンディアン (S16=1,S8=1)
メモリ (32bit)	メモリ (32bit)		×	×	
メモリ (32bit)	I/O 32bit(D31-0)				
	I/O 16bit(D31-16)		×	×	
	I/O 16bit(D15-0)		×	×	
	I/O 8bit(D31-24)		×	×	
	I/O 8bit(D23-16)		×	×	
	I/O 8bit(D15-8)		×	×	
	I/O 8bit(D7-0)		×	×	
メモリ (D31-16)	I/O 8bit(D31-24)		×	×	

### 14.1.8 ステータスレジスタ

オフセット:  $0x40*(x)+0x14$  ライト/リード

31	30	29	28	27	2	1	0
L0	L1	L2	L3	-	ER	ED	

bit 名	機能
L0	Location 0 :Default 0 0:内部データレジスタの D31-24 は有効なデータでない . 1:内部データレジスタの D31-24 は有効なデータである .
L1	Location 1 :Default 0 0:内部データレジスタの D23-16 は有効なデータでない . 1:内部データレジスタの D23-16 は有効なデータである .
L2	Location 2 :Default 0 0:内部データレジスタの D15-8 は有効なデータでない . 1:内部データレジスタの D15-8 は有効なデータである .
L3	Location 3 :Default 0 0:内部データレジスタの D7-0 は有効なデータでない . 1:内部データレジスタの D7-0 は有効なデータである .
ER	Error :Default 0 0:don't care 1:DMA 転送中にエラーが発生して DMA 転送が停止したことを示す . 本ビットは 0 をライトするとクリアされる .
ED	END :Default 0 0:don't care 1:DMA 転送が終了すると 1 に設定される . 本ビットは 0 をライトするとクリアされる .

# 15

## バスサイジング機能付き DMA

### 15.1 本 DMA の特徴

256 bit ⇔ 32 bit のバスサイジングをしながら転送する。

### 15.2 制御レジスタ

表 15.1: 制御レジスタ一覧

アドレス	レジスタ名	用途
0xFFFFD000	PSA	転送元アドレスを指定 ( 32 bit )
0xFFFFD004	MDA	転送先アドレスを指定 ( 32 bit )
0xFFFFD008	LENGTH	転送データ数を指定 ( byte )
0xFFFFD00C	MODE	転送モード指定、転送開始指定

### 15.3 制御レジスタ詳細

#### 15.3.1 PSA レジスタ

アドレス: 0xFFFFD000

DMA 転送先のアドレスを 32 bit で指定する。Read/Write 可能。

31

0

PSA

### 15.3.2 MDA レジスタ

アドレス: 0xFFFFD004

DMA 転送元のアドレスを 32 bit で指定する。Read/Write 可能。



### 15.3.3 LENGTH レジスタ

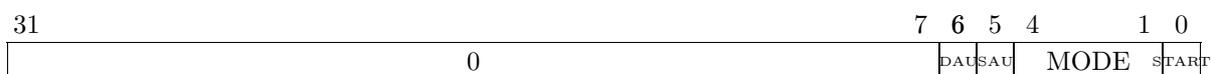
アドレス: 0xFFFFD008

DMA 転送するデータ数を byte 単位で指定する。Read/Write 可能。



### 15.3.4 MODE レジスタ

アドレス: 0xFFFFD00C



bit 名	機能																														
DAU	DMA 転送開始時に転送先として設定されているアドレス。Write のみ可能。 <table border="1" data-bbox="667 369 1109 492"> <thead> <tr> <th>設定値</th> <th>動作</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>MDA に設定したアドレス</td> </tr> <tr> <td>1</td> <td>前回の転送先の続き</td> </tr> </tbody> </table>	設定値	動作	0	MDA に設定したアドレス	1	前回の転送先の続き																								
設定値	動作																														
0	MDA に設定したアドレス																														
1	前回の転送先の続き																														
SAU	DMA 転送開始時に転送元として設定されているアドレス。Write のみ可能。 <table border="1" data-bbox="667 622 1109 745"> <thead> <tr> <th>設定値</th> <th>動作</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>MDA に設定したアドレス</td> </tr> <tr> <td>1</td> <td>前回の転送元の続き</td> </tr> </tbody> </table>	設定値	動作	0	MDA に設定したアドレス	1	前回の転送元の続き																								
設定値	動作																														
0	MDA に設定したアドレス																														
1	前回の転送元の続き																														
MODE	DMA 転送元と転送先の対象を指定する。Read/Write 可能。 <table border="1" data-bbox="702 875 1074 1283"> <thead> <tr> <th>設定値</th> <th>転送先</th> <th>転送元</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>I/O</td> <td>I/O</td> </tr> <tr> <td>0100</td> <td>Memory</td> <td>I/O</td> </tr> <tr> <td>0001</td> <td>I/O</td> <td>Memory</td> </tr> <tr> <td>0101</td> <td>Memory</td> <td>Memory</td> </tr> <tr> <td>1100</td> <td>SDRAM</td> <td>I/O</td> </tr> <tr> <td>1101</td> <td>SDRAM</td> <td>Memory</td> </tr> <tr> <td>0011</td> <td>I/O</td> <td>SDRAM</td> </tr> <tr> <td>0111</td> <td>Memory</td> <td>SDRAM</td> </tr> <tr> <td>1111</td> <td>SDRAM</td> <td>SDRAM</td> </tr> </tbody> </table> <ul style="list-style-type: none"> <li>● SDRAM : DDR SDRAM I/F (256 bit bus)</li> <li>● Memory : 32 bit bus に接続されたメモリ</li> <li>● I/O : 32 bit bus に接続された I/O</li> </ul>	設定値	転送先	転送元	0000	I/O	I/O	0100	Memory	I/O	0001	I/O	Memory	0101	Memory	Memory	1100	SDRAM	I/O	1101	SDRAM	Memory	0011	I/O	SDRAM	0111	Memory	SDRAM	1111	SDRAM	SDRAM
設定値	転送先	転送元																													
0000	I/O	I/O																													
0100	Memory	I/O																													
0001	I/O	Memory																													
0101	Memory	Memory																													
1100	SDRAM	I/O																													
1101	SDRAM	Memory																													
0011	I/O	SDRAM																													
0111	Memory	SDRAM																													
1111	SDRAM	SDRAM																													
START	DMA の転送の開始を指定する。Read/Write 可能。 <table border="1" data-bbox="686 1597 1090 1720"> <thead> <tr> <th>設定値</th> <th>動作</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>DMA 転送終了状態</td> </tr> <tr> <td>1</td> <td>DMA 転送開始/転送中</td> </tr> </tbody> </table>	設定値	動作	0	DMA 転送終了状態	1	DMA 転送開始/転送中																								
設定値	動作																														
0	DMA 転送終了状態																														
1	DMA 転送開始/転送中																														

## 15.4 注意事項

- データ転送に指定するアドレスは 8word(32Byte) アラインにそろえる必要がある。



# 16

## パルスカウンタ

---

### 16.1 パルスカウンタ概要

- 位相（2入力）による Up-Down Counter（いわゆるマウスカウンタ）
- Z相によるリセット/割り込み機能（ソフトウェアで選択可能）
- bit幅：32bit
- パルスカウント機能：カウント数がコンペアレジスタにあらかじめ設定されている数になるとパルス（割り込み）を発生
- 上記パルス発生の許可レジスタ及びステータスレジスタ
- 外部入力
- チャンネル数：9

### 16.2 レジスタインタフェース

#### 16.2.1 パルスカウンタ制御レジスタ

アドレス	パルスカウンタ制御レジスタ
0xFFFF7000	PLSCTRL[0]
0xFFFF7020	PLSCTRL[1]
0xFFFF7040	PLSCTRL[2]
0xFFFF7060	PLSCTRL[3]
0xFFFF7080	PLSCTRL[4]
0xFFFF70a0	PLSCTRL[5]
0xFFFF70c0	PLSCTRL[6]
0xFFFF70e0	PLSCTRL[7]
0xFFFF7100	PLSCTRL[8]



## 16.2.2 コンペアデータレジスタ

アドレス	コンペアデータレジスタ
0xFFFF7004	CMP[0]
0xFFFF7024	CMP[1]
0xFFFF7044	CMP[2]
0xFFFF7064	CMP[3]
0xFFFF7084	CMP[4]
0xFFFF70A4	CMP[5]
0xFFFF70C4	CMP[6]
0xFFFF70E4	CMP[7]
0xFFFF7104	CMP[8]

リード/ライト時

31	CMP<31:0>	0
----	-----------	---

bit 名	機能
CMP<31:0>	Compare Data :Default X カウンタ値と比較す比較データを格納する．SEL bit が 0 の場合，カウンタがこの値と等しくなると割込みを発生する．

## 16.2.3 カウンタレジスタ

アドレス	カウンタレジスタ
0xFFFF7008	CNT[0]
0xFFFF7028	CNT[1]
0xFFFF7048	CNT[2]
0xFFFF7068	CNT[3]
0xFFFF7088	CNT[4]
0xFFFF70A8	CNT[5]
0xFFFF70C8	CNT[6]
0xFFFF70E8	CNT[7]
0xFFFF7108	CNT[8]

リード時

31	CNT<31:0>	0
----	-----------	---

bit 名	機能
CNT<31:0>	Count Data :Default X ラッチパルスが入力された時にカウンタの値が本レジスタにラッチされる .

#### 16.2.4 タイマレジスタ

アドレス	タイマレジスタ
0xFFFF700C	TIMER[0]
0xFFFF702C	TIMER[1]
0xFFFF704C	TIMER[2]
0xFFFF706C	TIMER[3]
0xFFFF708C	TIMER[4]
0xFFFF70AC	TIMER[5]
0xFFFF70CC	TIMER[6]
0xFFFF70EC	TIMER[7]
0xFFFF710C	TIMER[8]

リード/ライト時

31	0
TIMER<31:0>	

bit 名	機能
TIMER<31:0>	Timer Data :Default X 周期割り込みに使用するタイマ値を設定する . カウンタクロックをカウントし本タイマ値と等しくなると , SEL bit が 1 の場合 , 割り込みを発生させる .

# 17

## PWM 発生器

---

### 17.1 PWM 発生器概要

- PWM 出力：内部レジスタの設定によってデューティ比の異なる矩形波を出力
- Bit 幅：32bit
- ノコギリ波を用いて PWM を発生するノコギリ波モードと三角波を用いた三角波モード
- デッドタイムの設定
- 正論理、負論理の設定
- 汎用出力としても利用可能
- チャンネル数：6

図 17.1 にノコギリ波モード、図 17.2 に三角波モードの PWM 波形を示す。

デッドタイム付反転出力は、隣の PWM 発生器から出力することができるようにサイクリックにカスケード接続されている。具体的には、PWM 発生器 N のデッドタイム付反転出力は、REV bit を立てることにより、PWM 発生器 N+1 で利用することができる。

## 17.2 PWM コントロールレジスタ

アドレス	CTRL レジスタ
0xFFFF7200	PWMCTRL[0]
0xFFFF7220	PWMCTRL[1]
0xFFFF7240	PWMCTRL[2]
0xFFFF7260	PWMCTRL[3]
0xFFFF7280	PWMCTRL[4]
0xFFFF72A0	PWMCTRL[5]

リード/ライト

31	0	7	6	5	4	3	2	1	0
		M	REV	DEN	D	P	CLR	CEN	

bit 名	機能
M	Mode: Default 0 0: ノコギリ波で PWM を生成するノコギリ波モード 1: 三角波で PWM を生成する三角波モード
REV	Reverse mode enable: Default 0 0: 本 PWM 発生器で生成された PWM 波を出力する通常モード 1: 本 PWM 発生器より一つ若い番号の PWM 発生器で生成された PWM 波のデッドタイム付反転出力を出力するモード (本 PWM 発生器内のカウンタは使用しない) DEN より優先度が高い
DEN	Data Enable: Default 0 0: 生成した PWM 波を出力する 1: D bit に設定された値 (一定値) を出力する REV より優先度が低い
D	Data: Default 0 DEN が 1 の時、本 D bit に設定された値 (一定値) を出力する
P	Positive: Default 0 PWM 波の論理を決定する (図 17.1, 17.2 参照)。 0: 負論理 1: 正論理
CLR	Counter clear: Default 0 0: 通常動作 1: カウンタをクリアする
CEN	Count Enable: Default 0 0: カウンタを停止する 1: カウンタを起動する

## 17.3 PWM 周期制御レジスタ

アドレス	PWM 正転制御レジスタ
0xFFFF7204	FWCNT[0]
0xFFFF7224	FWCNT[1]
0xFFFF7244	FWCNT[2]
0xFFFF7264	FWCNT[3]
0xFFFF7284	FWCNT[4]
0xFFFF72A4	FWCNT[5]

リード/ライト

31

0

FWCNT<31:0>

bit 名	機能
FWCNT	<p>Forward Counter: Default 0</p> <p>PWM の周期を決定するカウンタレジスタである。</p> <p>Mode が 0(ノコギリ波モード) の時には、PWM の周期を決定する。PWM 用カウンタが 0 から FWCNT までカウントアップすると、次のクロックで 0 に戻るようなノコギリ波を生成する (図 17.1 参照)。</p> <p>Mode が 1(三角波モード) の時には、PWM の半周期を決定する。PWM 用カウンタが 0 から FWCNT までカウントアップすると、次のクロックから 0 にカウントダウンするような三角波を生成する (図 17.2 参照)。</p>

## 17.4 PWM 反転制御レジスタ

アドレス	PWM 反転制御レジスタ
0xFFFF7208	REVCNT[0]
0xFFFF7228	REVCNT[1]
0xFFFF7248	REVCNT[2]
0xFFFF7268	REVCNT[3]
0xFFFF7288	REVCNT[4]
0xFFFF72A8	REVCNT[5]

リード/ライト

31	REVCNT<31:0>	0
----	--------------	---

bit 名	機能
REVCNT	Reverse Counter: Default 0 PWM 出力を反転する時間を決定するレジスタである。カウンタ値が本レジスタ値と同じになったら PWM 出力は反転する (図 17.1, 17.2 参照)。

## 17.5 デッドタイムレジスタ

アドレス	デッドタイムレジスタ
0xFFFF720C	DT[0]
0xFFFF722C	DT[1]
0xFFFF724C	DT[2]
0xFFFF726C	DT[3]
0xFFFF728C	DT[4]
0xFFFF72AC	DT[5]

リード/ライト

31	DT<15:0>	0
----	----------	---

bit 名	機能
DT<15:0>	Reverse Counter :Default 0 デッドタイムを指定するレジスタである。カウンタ値が本レジスタ値と同じになったら PWM 出力は反転する (図 17.1, 17.2 参照)。

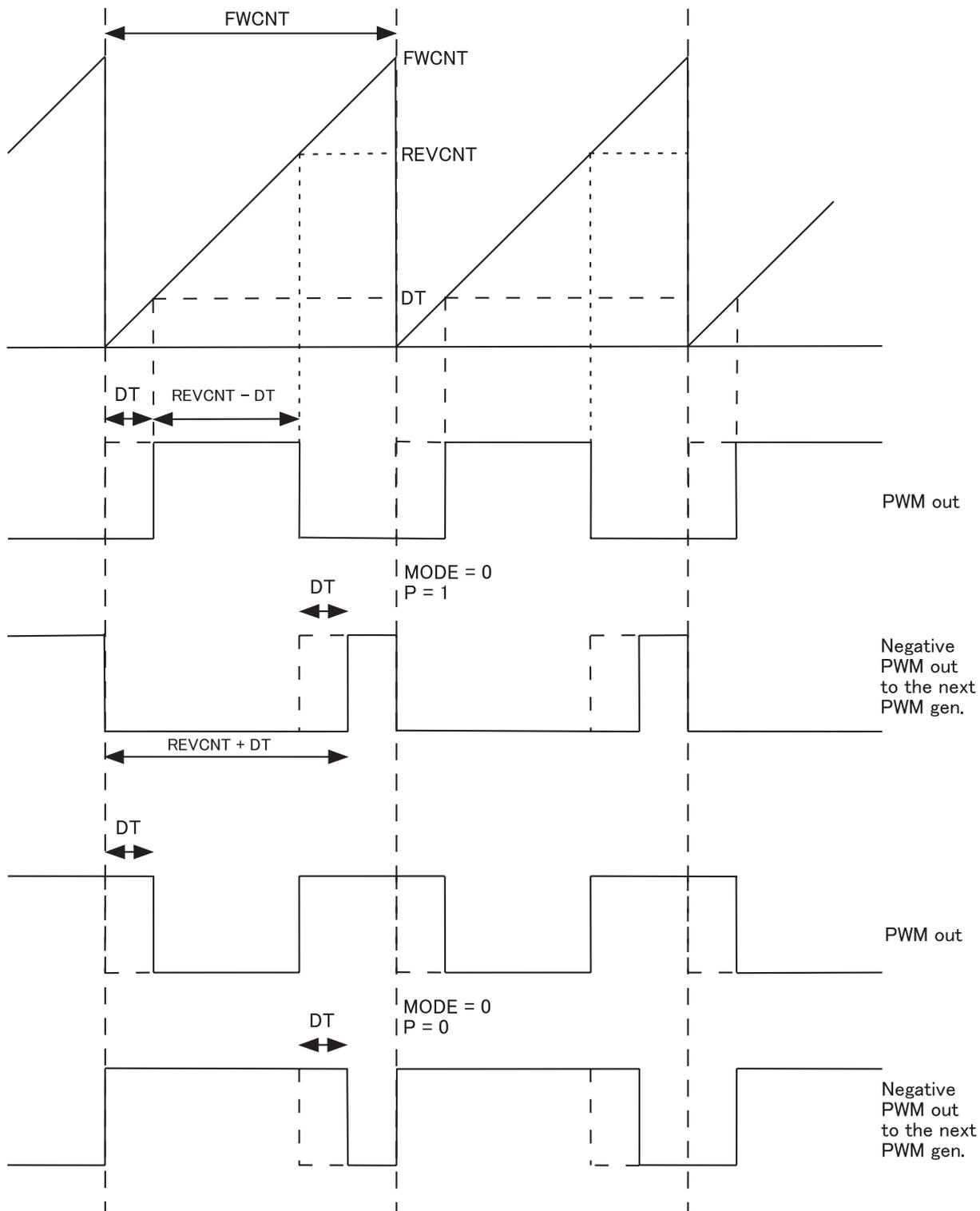


図 17.1: ノコギリ波モード

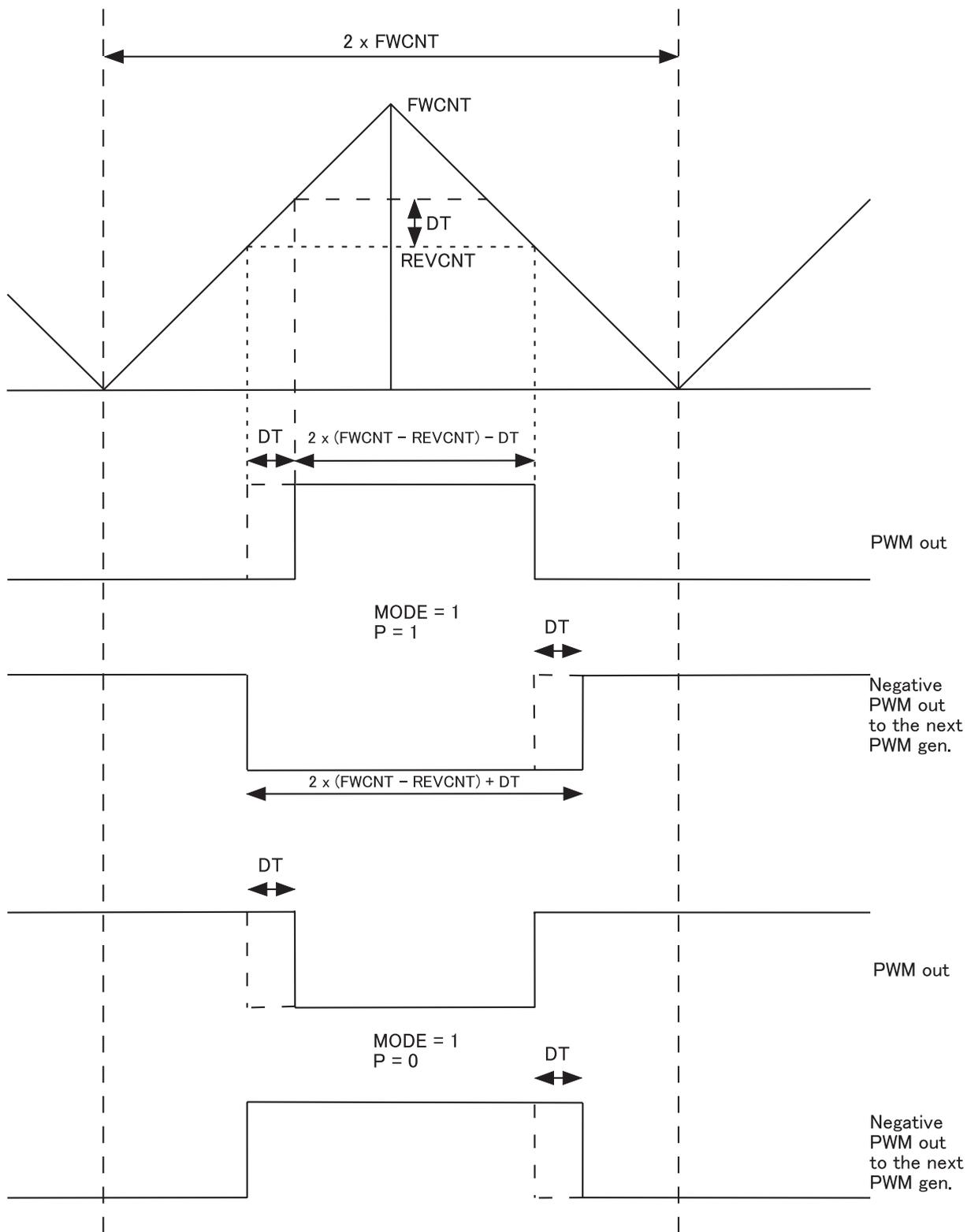


図 17.2: 三角波モード

# 18

## PWM入力器

### 18.1 PWM入力器概要

- PWM入力のデューティ比を High カウンタと Low カウンタの比に数値化
- Bit 幅 : 32bit
- チャンネル数 : 2
- クロックジェネレータ (9 章参照) で生成した基準クロックによってカウント
- 複数周期のデューティ比を求めて平均化する機構
- 割り込み発生機能

### 18.2 PWMIN コントロールレジスタ

アドレス	PWMIN コントロールレジスタ
0xFFFF7400	PWMINCTRL[0]
0xFFFF7420	PWMINCTRL[1]

リード/ライト

31	-	10 9	6 5	2 1 0
		LP	LPO	CLR IEN

bit 名	機能
IEN	Interrupt Enable :Default 0 r/w 0: 割り込みを発生しない . 1: 設定した周期分のデューティ比をカウント後に毎回割り込みを発生させる .
CLR	Interrupt Clear :Default 0 r/w 0:割り込みをクリアしない . 1:割り込みをクリアする . 割り込みクリア完了後に 0 にリセットされる .
LPO	Loop Original :Default 1 r/w 何周期分のデューティ比を平均化するか , その周期を設定する ( 1 から 15 まで , 0 は禁止 ) .
LP	Loop :Default 1 ro 現在実行している周期を示す .

### 18.3 PWMIN HIGH レジスタ

アドレス	PWMIN HIGH レジスタ
0xFFFF7404	HIGH[0]
0xFFFF7424	HIGH[1]

リード

31	0
HIGH<31:0>	

bit 名	機能
HIGH<31:0>	High :Default X 指定した PWM 周期分合計の High の期間 . 単位は , クロックジェネレータでプログラマブルに設定した PWMIN 用クロックのサイクル数 .

### 18.4 PWMIN LOW レジスタ

アドレス	PWMIN LOW レジスタ
0xFFFF7408	LOW[0]
0xFFFF7428	LOW[1]

リード/ライト

31	0
LOW<31:0>	

bit 名	機能
LOW<31:0>	Low :Default X 指定した PWM 周期分合計の Low の期間 . 単位は , クロックジェネレータでプログラマブルに設定した PWMIN 用クロックのサイクル数 .

# 19

## PCI I/F

初期アドレス: 0xffff3000

### 19.1 アドレスマップ

#### 19.1.1 Local Bus

PCI I/F

offset	31	24	23	16	15	8	7	0										
0x00	0x0000				te	me	ee	ite	lts	lme	lms	0	pe	ps	le	ls	id	0
0x04	data	4'b0000		direq	bst	rest		data	4'b0000		direq	bst	rest					
0x08	MailboxA_Higher ( PCI → Local )																	
0x0c	MailboxA_Lower ( PCI → Local )																	
0x10	MailboxB ( Local → PCI )																	
0x14	Reserved																	
0x18	Local AD			Mode		Reserved										Local AD		
0x1c	Reserved																	
0x20	Local Bus Access Port(Local-BAP)																	
0x24	Reserved																	
0x28	PCI Bus Access Port ( PCI-BAP )																	
0x2c	Reserved																	
0x30	Current Local AD																	
0x34	Reserved																	
0x38	Reserved																	
0x3c	Reserved																	

DMA(Channel0)

offset	31	24 23	16 15	8 7	0
0x40	B/C AR	Mode	Base/Current Address Register(B/C AR)		
0x44	B/C DCR	00	Base/Current Data Count Register(B/C DCR)		
0x48	dsr		00000	dsmr	dmr
0x4c	00000000		dcr	dmcr	damr
0x50	16'h0000		fdcr	00	fdcr
0x54	rqpr	00	rqpr	wqpr	wqpr
0x58	fsr		mrber	fsmr	8'h00
0x5c	8'h00		fcrr	fmcr	famr

## DMA(Channel0)

offset	31	24 23	16 15	8 7	0
0xc0	B/C AR	Mode	Base/Current Address Register(B/C AR)		
0xc4	B/C DCR	00	Base/Current Data Count Register(B/C DCR)		
0xc8	dsr		00000	dsmr	dmr
0xcc	00000000		dcr	dmcr	damr
0xd0	16'h0000		fdcr	00	fdcr
0xd4	rqpr	00	rqpr	wqpr	wqpr
0xd8	fsr		mrber	fsmr	8'h00
0xdc	8'h00		fcrr	fmcr	famr

## PCI Configuration Register

offset	31	24 23	16 15	8 7	0
0x100	VendorID			DeviceID	
0x104	Command			Status	
0x108	RevisionID			ClassCode	
0x10c	CacheLineSize	LatencyTimer	HeaderType	BIST	
0x110	Base Address Register				
0x114	Reserved				
0x118	Reserved				
0x11c	Reserved				
0x120	Reserved				
0x124	Reserved				
0x12c	SubsystemVendorID			SubsystemID	
0x130	Expansion ROM Base Address				
0x134	Cap_Ptr	Reserved			
0x138	Reserved				
0x13c	InterruptLine	InterruptPin	Min_Gnt	Max_Lat	

## 19.1.2 PCI Bus

PCI I/F

offset	31	24	23	16	15	8	7	0
0x00	pe	ps	le	ls	id	0	te	me
0x04	direc	bst	rest	data	4'b0000	direc	bst	rest
0x08	MailboxA_Lower ( PCI → Local )							
0x0c	MailboxA_Higher ( PCI → Local )							
0x10	MailboxB ( Local → PCI )							
0x14	Reserved							
0x18	Local AD							Mode
0x1c	Reserved							
0x20	Local Bus Access Port(Local-BAP)							
0x24	Reserved							
0x28	PCI Bus Access Port ( PCI-BAP )							
0x2c	Reserved							
0x30	Current Local AD							
0x34	Reserved							
0x38	Reserved							
0x3c	Reserved							

DMA(Channel0)

offset	31	24	23	16	15	8	7	0
0x40	Base/Current Address Register(B/C AR)							Mode
0x44	Reserved							
0x48	Base/Current Data Count Register(B/C DCR)							00
0x4c	Reserved							
0x50	dmr	dsmr	00000	edcc	ca	has	32	dcr
0x54	Reserved							
0x58	damr	dmcr	dcr	00000000				
0x5c	Reserved							
0x60	focr	00	16'h0000					
0x64	Reserved							
0x68	wqpr	00	rqpr	00				
0x6c	Reserved							
0x70	8'h00	fsmr	mrber	fcr				
0x74	Reserved							
0x78	famr	fmcr	fcr	8'h00				
0x7c	Reserved							

DMA(Channel1)

offset	31	24	23	16	15	8	7	0
0xc0	Base/Current Address Register(B/C AR)							Mode
0xc4	Reserved							
0xc8	Base/Current Data Count Register(B/C DCR)							00
0xcc	Reserved							
0xd0	dmr	dsmr			00000	edccca	has32	dsr
0xd4	Reserved							
0xd8	damr	dmcr			dcr		00000000	
0xdc	Reserved							
0xe0	fdcr			00	16'h0000			
0xe4	Reserved							
0xe8	wqpr			00	rqpr			00
0xec	Reserved							
0xf0	8'h00	fsmr			mrber		fsr	
0xf4	Reserved							
0xf8	famr			fmcr		fcr		8'h00
0xfc	Reserved							

## 19.2 PCI I/F レジスタマップ

PCI BUS 側は little endian . 8bit の memory space .

### 19.2.1 割り込み制御レジスタ

オフセット: 0x0000(Local)

31															16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0000														te	me	ee	lte	lts	lme	lms	0	pe	ps	le	ls	id	0				

オフセット: 0x0000(PCI)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15												0
pe	ps	le	ls	id	0	te	me	ee	lte	lts	lme	lms	0	0x0000														

bit 名	機能
te	Target abort interrupt Enable ( Default:'b1 ) Local:R, PCI R/W Master 動作時に Target Abort を受信した際の PCI BUS への割り込み制御 1:割り込み許可 0:不許可
me	Master Abort Interrupt Enable ( Default:'b0 ) Local:R, PCI R/W Master 動作時に Master Abort を受信した際の PCI Bus への割り込み制御 1: 割り込み許可 0:不許可
ee	End of Process Interrupt Enable ( Default:'b0 ) Local: R/W, PCI: R DMA レジスタの CDCR が”0”になったときの Local Bus への割り込み制御 1: 割り込み許可 0:不許可
lte	Target Abort Interrupt Enable for Local side ( Default:'b0 ) Local: R/W, PCI: R Master 動作時に Target Abort を受信した際の Local Bus への割り込み制御 1: 割り込み許可 0:不許可
lts	Target Abort Interrupt Status for Local side ( Default:'b0 ) Local: R/W, PCI: R Master 動作の Target Abort 受信用の割り込みレジスタ (Local Bus 側) 受信時に”1”にセットされ, Local Bus 側から”1”を書込むことでリセット
lme	Master Abort Interrupt Enable for Master side ( Default:'b0 ) Local: R/W, PCI: R Master 動作時に Master Abort を受信した際の Local Bus への割り込み制御 1: 割り込み許可 0:不許可
lms	Master Abort Interrupt Status for Master side ( Default:'b0 ) Local: R/W, PCI: R Master 動作の Master Abort 受信用の割り込みレジスタ (Local Bus 側) 受信時に”1”にセットされ, Local Bus 側から”1”を書込むことでリセット

bit 名	機能
pe	PCI Bus Interrupt Enable ( Default:'b1 ) Local: R, PCI: R/W MailboxB への書込み時の PCI Bus への割り込み制御 . 1: 割り込み許可 0: 不許可
ps	PCI Bus Interrupt Status ( Default:'b0 ) Local: R, PCI: R/W DoorbellB 用の割り込みレジスタ, 割り込み発生時に"1"にセット PCI BUS から"1"を書込むことでリセット
le	Local Bus Interrupt Enable ( Default:'b1 ) Local: R/W, PCI: R MailboxA への書込み時の LocalBus への割り込み制御 . High,Low 一括で行なう . 1: 割り込み許可 0: 不許可
ls	Local Bus Interrupt Status ( Default:'b00 ) Local: R/W, PCI: R DoorbellA 用の割り込みレジスタ, 割り込み発生時に"1"にセット MailBoxA(Lower) - LIS(Lower),bit3 MailBoxA(Higher) - LIS(Higher), bit4 Local Bus から"1"を書込むことでそれぞれリセット . 割り込み解除には双方リセット する必要あり .
id	ID0,ID1 ( Default:'b00 ) Local: R/W, PCI: R LocalBus から任意の値を書込み可 , PCI から読み込み可 . bit3: ID1, bit2: ID0

## 19.2.2 プログラム制御レジスタ

オフセット: 0x0004(Local)

31	28	27	24	23	22	21	20	16	15	12	11	8	7	6	5	4	0
data	4'b0000			dreq	bst	rest			data	4'b0000			dreq	bst	rest		

オフセット: 0x0004(PCI)

31	30	29	28	24	23	20	19	16	15	14	13	12	8	7	4	3	0
dreq	bst	rest			data	4'b0000			dreq	bst	rest			data	4'b0000		

Channel1: 0x0007, 0x0006

Channel0: 0x0005, 0x0004

bit 名	機能
data	FIFO data number for Master Transaction Local: R, PCI: R Master 転送用の FIFO 内のデータ数をバイト単位で出力
dreq	DMA Request Local: R, PCI: R PCI I/F からの dreq_を出力
bst	Burst Number Local: R, PCI: R PCI I/F からの burst_ack_を出力 'b00: 8, 'b01: 4, 'b10: 2, 'b00: single
rest	Remaining Number of Master Transaction Local: R, PCI: R Master 転送時の残り転送数 'b10000 - 8 以上, 'b01000 - 4-7 'b00100 - 2-3, 'b00010 - 1 'b00001 - 0

### 19.2.3 MailboxA

オフセット: 0x000c(Local), 0x0008(PCI)

31	MailboxA.Lower ( PCI → Local )	0
----	--------------------------------	---

オフセット: 0x0008(Local), 0x000c(PCI)

31	MailboxA.Higher ( PCI → Local )	0
----	---------------------------------	---

bit 名	機能
MailboxA	MailBoxA ( PCI → Local )( Default: 64'h0 ) Local: R, PCI: R/W PCI Bus から任意の値を書くことができ, Local Bus から読み出すことが可能 32Bit PCI - 'h0c のみ, 64Bit PCI - 'h08, 'h0c

### 19.2.4 MailboxB

オフセット: 0x0010(Local), 0x0010(PCI)

31	0
MailboxB ( Local → PCI )	

bit 名	機能
MailboxB	MailBoxB ( Local → PCI )( Default: 64'h0 ) Local: R/W, PCI: R Local Bus から任意の値を書くことができ, PCI Bus から読み出すことが可能

### 19.2.5 Local AD

オフセット: 0x0018(Local)

31	26 25 24 23	0
Local AD	Mode	Local AD

オフセット: 0x0018(PCI)

31	2 1 0
Local AD	Mode

bit 名	機能
Local AD	Address for Local Bus ( Default: 32'h0 ) Local: R, PCI: R/W Target Transaction 時に Local Bus の address bus へ出力するアドレスを指定 . 転送開始時にこの値を Current Local AD に読み込む . 実際へ出力する値は [Local AD, 2'b00] .
Mode	Address Update Mode in Target Transaction ( Default: 2'b0 ) Local: R, PCI: R/W Local AD の更新モードを指定 . 2'b00: リニアインクリメント, その他: 固定

### 19.2.6 Local Bus Access Port

オフセット: 0x0020(Local), 0x0020(PCI)

31	0
Local Bus Access Port(Local-BAP)	

bit 名	機能
Local-BAP	Local Bus Access Port Local: 不可, PCI: R/W PCI Bus から Local Bus へのアクセスポート . このポートへアクセスすると Local Bus の権利を取得して transaction を行う .

### 19.2.7 PCI Bus Access Port

オフセット: 0x0028, 0x00a8(Local), 0x0028, 0x00a8(PCI)

31	PCI Bus Access Port ( PCI-BAP )	0
----	---------------------------------	---

bit 名	機能
PCI-BAP	PCI Bus Access Port Local: R/W, PCI: 不可 Local Bus から PCI Bus へのアクセスポート . このポートへアクセスすると Master Transaction 用の FIFO との転送を行う . 'h28: Channel0, 'ha8: Channel1

### 19.2.8 Current Local AD

オフセット: 0x0030(Local), 0x0030(PCI)

31	Current Local AD	0
----	------------------	---

bit 名	機能
C-Local AD	Current Local AD Local: R, PCI: R Target Transaction 時に Local Bus 転送に用いられている現在のアドレスを出力 . 転送開始時に Local AD に変更がある場合は Local AD の値を取り込む .

## 19.3 Master Transaction 用 DMA レジスタマップ

Channel ごとに独立して用意 . bit7 = Channel No.

### 19.3.1 Address Register

オフセット: 0x0040, 0x00c0(Local)

31	26 25 24 23	0
B/C AR	Mode	Base/Current Address Register(B/C AR)

オフセット: 0x0040, 0x00c0(PCI)

31	2 1 0
Base/Current Address Register(B/C AR)	Mode

bit 名	機能
B/C AR	Base/Current Address Register ( Default: 30'h0 ) Local: R/W, PCI: R/W PCI Master 動作時に PCI Bus に出力するアドレスの指定 . 実際に出力されるアドレスは [AR,2'b00] . このポートへの書き込みは BAR と CAR を同時に変更するので , 転送中に明示的に BAR を変更することは推奨しない .
Mode	Address Update Mode in Master Transaction ( Default: 2'b00 ) Local: R/W, PCI: R/W Master Transaction 時のアドレス更新モードの指定 2'b00: リニアインクリメント, その他: 固定

### 19.3.2 Data Count Register

オフセット: 0x0044, 0x00c4(Local)

31	26 25 24 23	0
B/C DCR	00	Base/Current Data Count Register(B/C DCR)

オフセット: 0x0048, 0x00c8(PCI)

31	2 1 0
Base/Current Data Count Register(B/C DCR)	00

bit 名	機能
B/C DCR	Base/Current Data Count Register ( Default: 30'h0 ) Local R/W, PCI R/W Master Transaction 時の転送バイト数の指定 . 64bit PCI では 8, 32bit PCI では 4 ずつ減少 . Byte Enable に関わらず一定 . このポートへの書き込みは BDCR, CDCR 同時に値を変更するので , 転送中に値を明示的に変更することは推奨しない .

### 19.3.3 DMA Control Register

オフセット: 0x0048, 0x00c8(Local)

31	24	23	19	18	17	16	15	8	7	0
dsr		00000		cdce	cau	mas	32	dsmr		dmr

オフセット: 0x0050, 0x00d0(PCI)

31	24	23	16	15	11	10	9	8	7	0
dmr		dsmr		00000		cdce	cau	mas	32	dsr

bit 名	機能
dsr	DMA Status Register ( Default: 8'h10 ) Local: R, PCI: R DMA の状態出力ポート . [4]: pci_req_の値を出力 . Core への Master 転送要求時に"0" . [0]: tc(terminal count) の値を出力 . cdcr が 0 のとき"1" .
cdce	Circulate Data Count Enable ( Default: 1'b0 ) Local: R/W, PCI: R/W CDCR が 0 に Local Bus 側の DMA が転送を終了した際に自動的に転送を開始するかを指定 . 開始する際には CDCR には BDCR の値がロードされる . 1'b1:自動開始, 1'b0: 待機
cau	Current Address Update ( Default: 1'b0 ) Local: R/W, PCI: R/W CDCE=1 の場合の自動転送開始時における CAR の設定方法の指定 . 1'b1:最後の転送の次のアドレス (アドレス更新モードが固定ならば変更なし) 1'b0:BAR
mas32	Master 32Bit Mode ( Default: 1'b0 ) Local: R/W, PCI: R/W Core-I/F 間の転送モードの指定 . PCI-BUS 幅と同じ値の設定を推奨 . PCI-32bit, 64Bit Mode の場合は Disconnect や Retry の際にデータが消える恐れあり .
dsmr	DMA Single Mask Register ( Default: 8'h04 ) Local: R/W, PCI: R/W Master Transaction における PCI DMA 転送開始シグナル (pci_req_) のマスク . DSMR,DAMR の両方のマスクを解除する必要がある . bit[2] がマスク .
dmr	DMA Mode Register ( Default:8'h00 ) Local: R/W, PCI R/W Master Transaction 時のコマンドの指定 . 下位 4bit が app_cmd に出力 .

### 19.3.4 DMA Stop/Reset Register

オフセット: 0x004c, 0x00cc (Local)

31	24 23	16 15	8 7	0
00000000	dcr	dmc	damr	

オフセット: 0x0058, 0x00d8 (PCI)

31	24 23	16 15	8 7	0
damr	dmc	dcr	00000000	

bit 名	機能
dcr	DMA Clear Register Local: W, PCI: W ソフトウェアで DMA を初期化するポート . このポートへ書き込むと DMA(内部レジスタと status) が初期化される .
dmcr	DMA Mask Clear Register Local: W, PCI: W このポートへ書き込むと DMA Mask(DSMR,DAMR) がクリアされる .
damr	DMA All Mask Register ( Default:8'h01 ) Local: R/W, PCI: R/W Channel0,1 双方に効果のあるマスクレジスタ . どちらの Channel のポートへ書き込んでも同じ意味 .

### 19.3.5 FIFO Data Register

オフセット: 0x0050, 0x00d0(Local)

31	16 15	10 9 8 7	0
16'h0000	fdcr	00	fdc

オフセット: 0x0060, 0x00e0(PCI)

31	18 17 16 15	0
fdc	00	16'h0000

bit 名	機能
fdc	FIFO Data Count Register Local: R, PCI: R Master Transaction 用 FIFO のデータ数 (byte count) .

### 19.3.6 FIFO Request Parameter Register

オフセット: 0x0054,0x00d4(Local)

31	26 25 24 23	16 15	10 9 8 7	0	
rqpr	00	rqpr	wqpr	00	wqpr

オフセット: 0x0068,0x00e8(PCI)

31	18 17 16 15	2 1 0
wqpr	00	rqpr 00

bit 名	機能
wqpr	Write Request Parameter Register ( Default: 14'h0 ) Local: R/W, PCI: R/W PCI bus に write request を出す閾値の設定 . byte count で設定し , FIFO のデータ数が上回ると要求を出す .
rqpr	Read Request Parameter Register ( Default: 14'0 ) Local: R/W, PCI: R/W PCI bus に read request を出す閾値の設定 . byte count で設定し , FIFO の空きデータ数が上回ると要求を出す .

### 19.3.7 FIFO Control Register

オフセット: 0x0058, 0x00d8(Local)

31	24 23	16 15	8 7	0
fsr	mrber	fsmr	8'h00	

オフセット: 0x0070, 0x00f0(PCI)

31	24 23	16 15	8 7	0
8'h00	fsmr	mrber	fsr	

bit 名	機能
fsr	FIFO Status Register Local: R, PCI: R FIFO の Status 出力ポート . [7] - Full, [6] - Empty, [5] - R/W-, [4] - MREQ-, [3] - MACK-, [0] - EOP
mrber	Master Read Byte Enable Register ( Default: 8'h00 ) Local: R, PCI: R/W Master Read Transaction 時に出力する Byte Enable の設定 .
fsmr	FIFO Single Mask Register ( Default: 8'h04 ) Local: R/W, PCI: R/W 対応する Channel の pci_req_ の mask register . bit[2] がマスク . DMA と同様に FSMR と FAMR の両方を解除する必要がある .

### 19.3.8 FIFO Stop/Reset Register

オフセット: 0x005c, 0x00dc(Local)

31	24 23	16 15	8 7	0
8'h00	fcr	fmcrr	famr	

オフセット: 0x0078, 0x00f8(Local)

31	24 23	16 15	8 7	0
famr	fmcrr	fcr	8'h00	

bit 名	機能
fcr	FIFO Clear Register Local: W, PCI: W このポートへの書き込みは FIFO 内のデータをクリアする。内部レジスタはクリアされない。
fmcrr	FIFO Mask Clear Register Local: W, PCI: W このポートへの書き込みは対応する Channel の FSMR,FAMR を解除する。
famr	FIFO All Mask Register ( Default: 8'h01 ) Local: R/W, PCI: R/W 両 Channel にマスクをかける。bit[0] がマスク。

## 19.4 動作/使用方法

### 19.4.1 Target Transaction (PCI → Local)

PCI Bus 側から Local BUS をアクセスする際には転送前に Local AD を設定する必要があります。Local AD の設定後、Local Bus Access Port にアクセスすると、PCI I/F が Local Bus の権利を要求し、CPU bus のバスマスタとして動作します。Local Bus 側へ出力されるアドレスは Local AD です。各転送ごとにアドレスは更新され、そのモードは Local AD の下位 2bit で決定されます。

データの転送の際には一時的に PCI I/F 内にある FIFO にデータが格納されますが、転送自体は Local Bus の権利を獲得してから行なわれます。Local Bus の権利を獲得するのに時間がかかるので、PCI Bus の仕様にある 16clock rule は無視し、初めのデータ転送に限り 255clock で retry をかけます。次のデータ転送からは 8clock rule を守ります。

- Target Write Transaction

Local Bus の権利を取り、FIFO に空きがある場合のみ trdy\_ がアサートされます。FIFO に空きがなくなった場合は trdy\_ をディアサートし続けます。ただし、8clock たった場合は disconnect します。Local Bus 側への転送は FIFO にデータが格納された時点で開始します。

- Target Read Transaction  
Local Bus の権利を取るとすぐに Local Bus の転送を始めます。この転送は必ず Burst Access になり、複数データを一度に取ってこようとします。FIFO にデータが格納されると `trdy_` がアサートされます。FIFO にデータがない場合は `8clock` たった時点で disconnect されます。常にデータを先に取りに行くので、transaction 終了時に FIFO はリセットされます。
- target initiated termination  
termination が起こる要因は次の 2 つです。stop がアサートされるとともに configuration register の status register がセットされます。
  1. Clock Rule (Time up) 初めのデータについては `255clock`(仕様では `16clock`)、2 つ目以降では `8clock` 経過した場合に disconnect します。この場合は PCI 仕様に従ってただちに転送を再開してください。
  2. Local Bus Error Local Bus 側の転送で Error が起きた場合 (Address が間違っているなど) に target abort が発生します。master への通知は PCI 仕様に従って起こるため、master 側で対処を行なってください。ただし、`te` をセットし、PCI Bus に割り込みをかけてブリッジ側で対処することも可能です。

#### 19.4.2 Master Transaction( Local → PCI )

CPU Bus から PCI Bus にアクセスする場合は Local Bus の DMA を使用します (しなくても可)。PCI DMA のレジスタの設定の前には `pci_req_` と `dreq_` にマスクをかけます。かならず動作停止時にレジスタの変更を行なってください。設定するのは PCI Bus 上での相手アドレス (BAR)、転送データ数 (BDCR)、転送モード (DMR) が必須です。レジスタ設定後に `mask` を解除して Master 転送を開始します。

- Write  
BDCR が 1 以上で、FIFO に 1 つでも空きがあると `dreq_` がアサートされます。`dreq_` がアサートされている状態で、PCI Bus Access Port へ data が書込まれると FIFO に格納されます。  
FIFO のデータ数が `wqpr` で設定された閾値を上回ると `pci_req_` がアサートされます。PCI Bus の権利を獲得すると PCI Master となり、転送を始めます。1 回の転送は FIFO が空になるか、`CDCR` が 0 になるまで行なわれます。FIFO が空になって転送が中断した場合は再度 FIFO のデータ数が、`wqpr` を越えるか `CDCR` と一致するまで `pci_req_` をアサートしません。
- Read  
BDCR が 1 以上で、FIFO の空き数が `rqpr` で設定された閾値を上回ると `pci_req_` がアサートされます。PCI Bus の権利を獲得すると PCI Master になり転送を始めます。この転送は FIFO が一杯になるか、`CDCR` が 0 になるまで行なわれます。FIFO が一杯になって転送が中断した場合は再び空き領域が `rqpr` の閾値を越すか、`CDCR` と一致するまで `pci_req_` をアサートしません。FIFO にデータが 1 つでもあると `dreq_` がアサートされ、`dreq_` がアサートされている状態で、PCI Bus Access Port へ read access が来ると、FIFO から data が読み込まれます。
- Master Initiated Termination  
Master 動作時の Termination は通常終了以外は Master Abort のみです。PCI 仕様通りに Configuration Register の Master Abort 受信 bit がセットされます。Master での対処を行なう場合は、`lme` をセットして Local Bus 側へ割り込みがかかるようにしてください。

# 20

## IEEE1394

---

---

### 20.1 概要

本コアは、IEEE1394 の標準 IEEE1394A-2000 に準拠した、ハイスピードシリアルパスのリンク層コントローラです。

#### 20.1.1 特徴

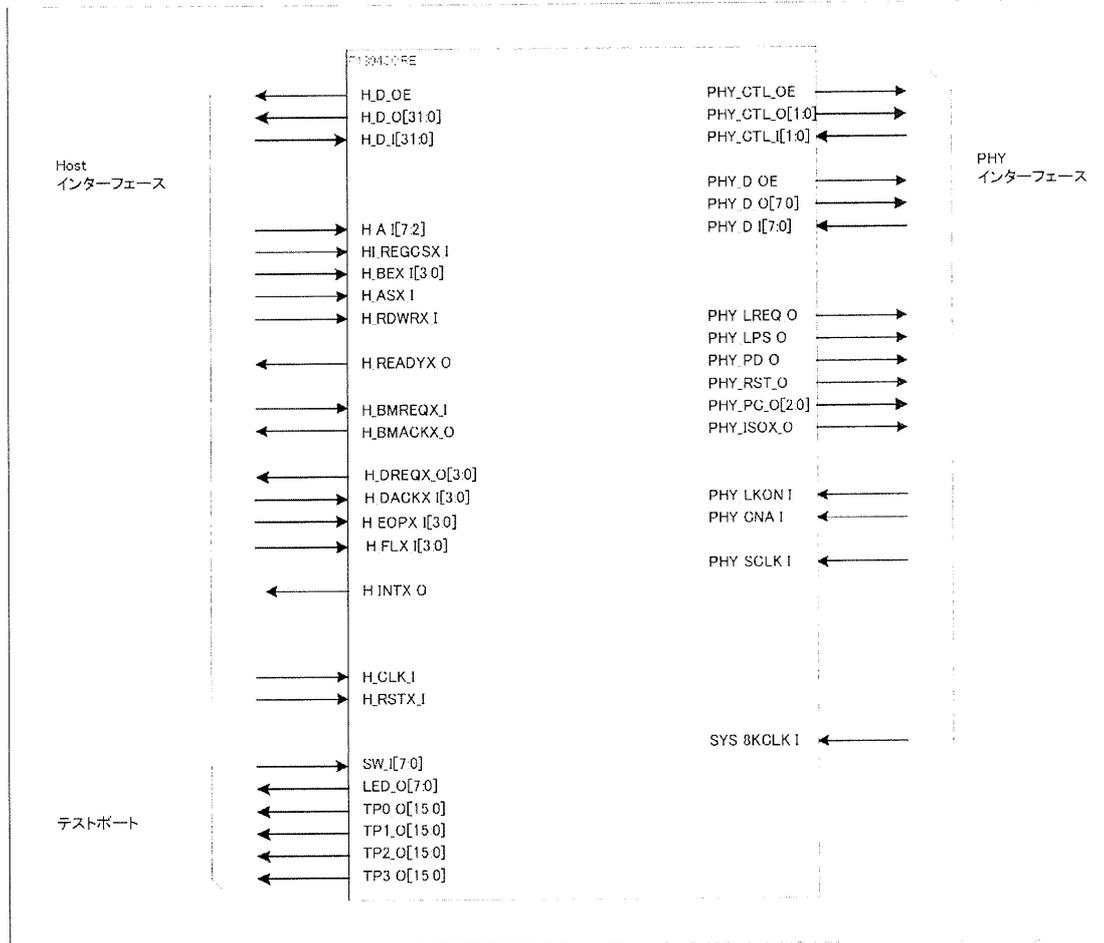
- IEEE1394-1995 及び IEEE1394A-2000 に準じた送信時のパッキング、受信時のアンパッキング
- サイクルマスターのサポート
- 32-bitCRC によるパリティ生成とエラー検出
- PHY チップとの DC 接続のインターフェースをサポート
- 100/200/400Mb/sec の 3 スピードのサポート
- バスタイムレジスタのサポート
- ドロップしたサイクルスタートメッセージの検出
- Asynchronous Stream パケットのサポート
- ホスト側のローカルアクセスと DMA 転送のサポート

#### 20.1.2 関連資料

- IEEE Standard for a High Performance Serial Bus(IEEE1394-1995)
- IEEE Standard for a High Performance Serial Bus-Amendment1(IEEE1394A-2000)
- IEEE1212-1991 Command and Status Register Architecture

## 20.2 構成

### 20.2.1 全体構成



## 20.2.2 入出力端子説明

&lt;ホストインターフェース&gt;

端子名	入出力	本数	内容
H_D_D[31:0]	出力	32	ホストデータ出力バス: レジスタ、データアクセス用データバス。
H_D_I[31:0]	入力	32	ホストデータ入力バス: レジスタ、データアクセス用データバス。
H_D_OE	出力	1	ホストデータ出力カインープル: レジスタ、データアクセス用データバス出力カインープル。
H_A_I[7:2]	入力	6	ホストアドレス: レジスタ選択用ホストアドレス。
H_BEX_I[3:0]	入力	4	ホストデータバイトイネープル: バイトイネープル信号です。 32ビットバスの為、レジスタライト時に対する有効バイトを示します。レジスタリード時は、無効で、全バイト出力され ます。本信号は、バスサイクル期間中のみ有効とされます。
H_ASX_I	入力	1	ホストアドレスストロープ: アドレスストロープ信号です。 バスサイクル最初のサイクルで1サイクル期間”L”が出力さ れ、リード、ライト、アイドル期間に”H”が出力されます。
H_RDWRX_I	入力	1	ホストデータリードライトイネープル: データリードライトイネープル信号です。 ライトの場合は、バスサイクル期間中、”L”が出力されます。 リードとアイドルサイクル中は、”H”が出力されます。
H_READYX_O	出力	1	ホストレディ: レディ出力信号です。 バスサイクル終了時”L”を出力しますので、ホスト側は、こ れを受け次のサイクルに移行してください。
H_IREGCSX_I	入力	1	レジスタ領域イネープル: 本コア内部のレジスタへアクセスする際”L”を入力します。
H_DREQX_O[3:0]	出力	4	DMA リクエスト (4 チャンネル): DMA アクセスリクエストイネープルです。 本コアからの DMA 転送要求時、”L”を出力します。 DMA 転送要求がある限り”L”を出力します。 Bit0: Asynchronous 送信 DMA 転送チャンネル Bit1: Asynchronous 受信 DMA 転送チャンネル Bit2: Isochronous 送信 DMA 転送チャンネル Bit3: Isochronous 受信 DMA 転送チャンネル
H_DACKX_I[3:0]	入力	4	DMA アクノリッジ (4 チャンネル): DMA アクセスアクノリッジです。 DMAC からの DMA アクノリッジ信号で、DMA 転送時 ”L”を入力します。 DMA 転送要求がある限り”L”を出力します。

			Bit0: Asynchronous 送信 DMA 転送チャンネル Bit1: Asynchronous 受信 DMA 転送チャンネル Bit2: Isochronous 送信 DMA 転送チャンネル Bit3: Isochronous 受信 DMA 転送チャンネル
H_EOPX_I[3:0]	入力	4	DMA 転送エンドオフパッケージ (4 チャンネル): DMA 転送エンドオフパッケージです。 DMAC からの DMA 転送エンドオフパッケージ信号で、DMAC の転送動作終了時”L”を入力します。 Bit0: Asynchronous 送信 DMA 転送チャンネル Bit1: Asynchronous 受信 DMA 転送チャンネル Bit2: Isochronous 送信 DMA 転送チャンネル Bit3: Isochronous 受信 DMA 転送チャンネル
H_FLX_I[3:0]	入力	4	DMA バッファフラッシュ (4 チャンネル): DMA バッファフラッシュです。 DMAC からの DMA バッファフラッシュ信号で、DMAC 内 のバッファフラッシュ終了時”L”を入力します。 Bit0: Asynchronous 送信 DMA 転送チャンネル Bit1: Asynchronous 受信 DMA 転送チャンネル Bit2: Isochronous 送信 DMA 転送チャンネル Bit3: Isochronous 受信 DMA 転送チャンネル
H_BMACKX_O	出力	1	パーストアクノリッジ: パーストアクセスアクノリッジです。 H_BMREQX_I に対して、パーストアクセスが可能な際に”L” を出力します。
H_BMREQX_I	入力	1	パーストリクエスト: パーストリクエストです。 ホストからパーストアクセス要求時、”L”が入力されます。内 部レジスタでパースト設定されており、かつパースト転送可 能な場合、H_BMACKX_ を”L”にして応答します。
H_INIT_O	出力	1	インターラプト: 割り込み出力です。 割り込み発生時”L”を出力します。
H_CLK_I	入力	1	ホストクロック: ホストからのクロックを入力します。 本信号に同期してホストバスを制御します。
H_RSTX_I	入力	1	ホストリセット: 本信号が”L”の時、本コアを初期化します。

## &lt;PHY インターフェース&gt;

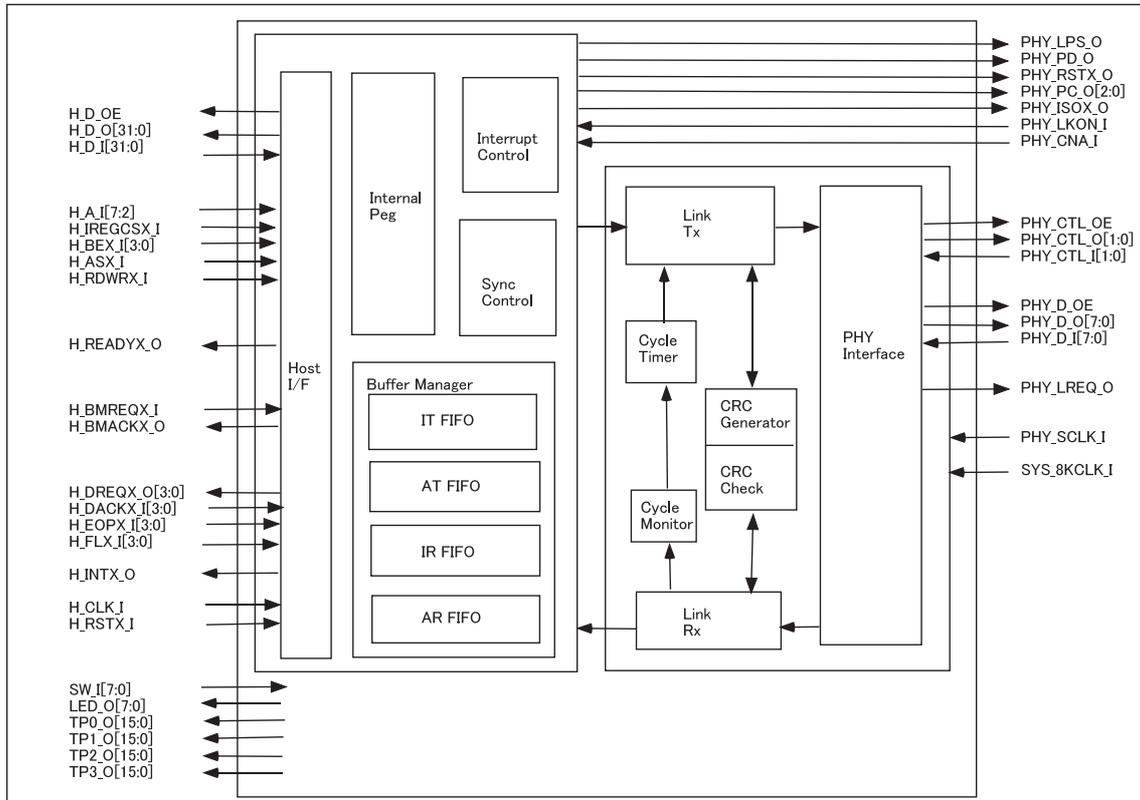
端子名	入出力	本数	内容
PHY_CTL_O[1:0]	出力	2	PHY-LINK コントロール出力: PHY チップとのインターフェース制御信号。 LLC がバスを駆動している時: PHY_CTL_O[0:1]=00b: アイドル、バスは LLC によって開放されている。 PHY_CTL_O[0:1]=01b: ホールド、LLC はバスをホールドしている。 PHY_CTL_O[0:1]=10b: 送信、LLC はデータ線経由で PHY にパケットを送っている。 PHY_CTL_O[0:1]=11b: 未使用
PHY_CTL_I[1:0]	入力	2	PHY-LINK コントロール入力: PHY チップとのインターフェース制御信号。 PHY がバスを駆動している時: PHY_CTL_I[0:1]=00b: アイドル、動作なし PHY_CTL_I[0:1]=01b: ステータス、PHY は LLC にステータス情報を送出している。 PHY_CTL_I[0:1]=10b: 受信、PHY は入ってきたパケットを LLC にデータ線経由で転送している。 PHY_CTL_I[0:1]=11b: 許可、PHY は LLC にバスの使用を許可している。
PHY_CTL_OE	出力	1	PHY-LINK コントロール出力イネーブル: PHY チップとのインターフェース出力イネーブル。
PHY_D_O[7:0]	出力	8	PHY-LINK データバス出力: PHY チップとのデータ送信用データバス。 100Mbps では [0..1]、200Mbps では [0..3]、400Mbps では [0..7] を使用します。
PHY_D_I[7:0]	入力	8	PHY-LINK データバス入力: PHY チップとのデータ送信用データバス。 100Mbps では [0..1]、200Mbps では [0..3]、400Mbps では [0..7] を使用します。
PHY_D_OE	出力	1	PHY-LINK データバス出力イネーブル: PHY チップとのデータ送信用データバス出力イネーブル。
PHY_LKON_I	入力	1	PHY リンクオン: PHY からのリンクパワーオンリクエスト
PHY_LPS_O	出力	1	PHY リンクパワーステータス: リンクを起動時 "H" を出力し、PHY へリンク状態を送ります。
PHY_LREQ_O	出力	1	PHY リンクリクエスト: PHY チップないのレジスタアクセス、シリアルバス使用の為に使用します。
PHY_SCLK_I	入力	1	PHY クロック: PHY チップから出力される 49.152MHz のクロック

PHY_CNA_I	入力	1	PHY ケーブルノットアクティブ: ”H”時、ケーブル接続がされていないことを示します。
PHY_PD_O	出力	1	PHY パワーダウン: ”H”時 PHY チップをパワーダウンします。
PHY_RSTX_O	出力	1	PHY リセット: ”L”時 PHY チップをリセットします。
PHY_PC_O	出力	1	アイソクロナスクロック: アイソレーション端子を制御します。
SYS_8KCLK_I	入力	1	アイソクロナスクロック: アイソクロナス外部ソースクロックです。

## &lt;テストポート&gt;

端子名	入出力	本数	内容
SW_[7:0]	入力	8	テスト用入力端子 0Ch: 通常モード Others: テストモード
LED_O[7:0]	出力	8	テスト用出力端子
TP0_O[15:0]	出力	16	テスト用出力端子
TP1_O[15:0]	出力	16	テスト用出力端子
TP2_O[15:0]	出力	16	テスト用出力端子
TP3_O[15:0]	出力	16	テスト用出力端子

20.2.3 内部構成図



### 20.2.4 構成図説明

- Host I/F  
ホストインターフェースは、PARC lite バスライクな同期バスとなっております。バッファ状態に応じて DMA 転送要求を行い高速にデータ転送を行うことができます。
- Application Block
  - Control(Interrupt Control, Sync Control)
  - Internal Register
  - Buffer Manager
- Link Core Block
  - Link Tx  
Asynchronous、Isochronous の両送信バッファからデータを読み出し、IEEE1394 で定義される各パケットフォーマットにして PHY インターフェースへパケットを送出します。  
ノードがルートのときは、サイクルスタートパケットの送出手も行います。
  - Link Rx  
PHY インターフェースからのパケットを受信し、そのパケットが自ノード宛かどうか判断します。  
Asynchronous パケットであれば、Asynchronous 受信バッファへ書き込みます。  
Isochronous パケットであれば、Isochronous 受信バッファへ書き込みます。
  - Cycle Timer  
Isochronous サイクルスタートパケットの送出管理を行います。
  - Cycle Monitor  
Isochronous サイクルスタートパケットの監視を行います。
  - CRC Generator  
送信パケットの巡回冗長チェック用コードの生成を行います。
  - CRC Check  
受信パケットの巡回冗長チェック用コードのチェックを行います。
  - PHY Interface  
PHY チップを直接接続可能なインターフェースを有しています。  
接続対象チップは、100Mbps、200Mbps、400Mbps のいずれのチップも可能です。  
本コアでは、DC 接続をサポートします。

## 20.3 レジスタ

## 20.3.1 レジスタ一覧

offset	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		
00h	Version													Revision													Version							
04h														CTL_CYCSRC	CTL_CYCMAS	CTL_CYCTIMEN																CTL_REN	CTL_TEN	Control
08h	NID_IDVAL																NID_BUSID						NID_NODEID						Node Identification					
0Ch																								RET_DMA	RET_LINK				RET_IRF	RET_ITF	RET_ARF	RST_ATF	Reset	
10h																																	Reserved	
14h																																	Reserved	
18h																				PKTCTL_WPEN						PKTCTL_PHY	PKTCTL_SID						Packet Control	
1Ch						LC_RDITBUSY	LC_RDATBUSY													LC_RDRXACKST													Diagnostic Status	





### 20.3.2 レジスタ内容

#### Version Register

offset	31	24 23	16 15	8 7	0
00h	Version			Revision	

初期値: XXXX XXXXh

本レジスタは、RTL のバージョン、レビジョンナンバーを示すリードオンリーレジスタです。

**Bit 15 ~ 0** : Revision

**Bit 31 ~ 16** : Version

## Control Register

offset: 04h

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	
													CTL_CYCSRC	CTL_CYCMAS	CTL_CYCTIMEN																CTL_REN	CTL_TEN

初期値: 0000\_0000h

本レジスタは、チップの各動作のコンフィグレーション、イネーブル等の設定を行います。通常、電源投入直後にこのレジスタの設定を行い、本コアのコンフィグレーションを決めておきます。

bit 名	機能
CTL_TEN<0>	Transmitter Enable ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>0 = トランスミッターをディスイネーブルにします。</li> <li>1 = トランスミッターをイネーブルにします。</li> </ul> <p>本レジスタのトランスミッターをイネーブルにするか否かを設定します。イネーブル時は以下の送信を行います。</p> <ul style="list-style-type: none"> <li>Asynchronous パケット</li> <li>CycleMaster ビットがイネーブル時でのサイクルスタートパケット</li> <li>サイクルスタート時での Isochronous パケット</li> </ul>
CTL_REN<1>	Receiver Enable ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>0 = レシーバーをディスイネーブルにします。</li> <li>1 = レシーバーをイネーブルにします。</li> </ul> <p>本コアのレシーバーをイネーブルにするか否かを設定します。イネーブル時は以下の受信を行います。</p> <ul style="list-style-type: none"> <li>他のノードからこのノードにアドレスされた Asynchronous パケット</li> <li>指定したチャンネルの Isochronous パケット</li> </ul>
CTL_CYCTIMEN<16>	Cycle Timer Enable ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>0 = サイクルタイマーをディスイネーブルにします。</li> <li>1 = サイクルタイマーをイネーブルにします。</li> </ul> <p>本コア 内部のサイクルタイマーをイネーブルにするか否かを設定します。</p>
CTL_CYCMAS	Cycle Master ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>0 = 他のルートノードからのサイクルスタートパケットを受信し、サイクルタイマーの管理を行います。</li> <li>1 = 自ノードがルートであり、かつこのビットが'1'の時、本コアのサイクルタイマーがキャリアするたびに、サイクルスタートパケットを生成します。</li> </ul>
CTL_CYCSRC<18>	Cycle Sourcer ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>0 = PHY チップから供給されるクロックであるマスタークロックの 49.152MHz を 2 分周して 24.576MHz でサイクルタイマーをカウントして Isochronous サイクルを管理する。</li> <li>1 = SYS_8KCLK_I 端子から入力される信号の立ち上がりでサイクルタイマーを更新して Isochronous サイクルを管理します。Isochronous の時間管理を行っている、内部のサイクルタイマーの更新元を設定します。</li> </ul>

## Node Identification Register

offset : 08h

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
NID_IDVAL																NID_BUSID										NID_NODEID					

bit 名	機能
NID_NODEID<5:0>	<p>Node Number ビット (RW - 初期値: 00h)</p> <p>この値は IEEE1212 空間で定義される 6-bit のノードナンバーを設定します。送信時 IEEE1394 パケットフォーマットのヘッダー内のソース領域にこの値を使用します。また、受信時は受信するパケットのデスティネーションのノードナンバーを見て、この値と一致する場合は受信、そうでない場合はリジェクトします。通常はバスリセット後の Self Identification フェイズの終了後、PHY チップからノードナンバーを読み出し、このレジスタに設定します。</p>
NID_BUSID<15:6>	<p>Bus Number ビット (RW - 初期値: 000h)</p> <p>この値は IEEE1212 空間で定義される 10-bit のバスナンバーを設定します。送信時 IEEE1394 パケットフォーマットのヘッダー領域のソース領域にこの値を使用します。また、受信時は受信するパケットのデスティネーションのバスナンバーを見て、この値と一致する場合は受信、そうでない場合はリジェクトします。通常は、'3FFH' に指定します。</p>
NID_IDVAL<31>	<p>ID Valid ビット (RW- 初期値: 00b)</p> <ul style="list-style-type: none"> <li>● 0 = BusNumber の値が'3FFH' がかつ NodeNumber の値が'3Fh' にアドレスされたパケットのみを受信します。それ以外のパケットはリジェクトします。</li> <li>● 1 = 以下の条件で上記レジスタで設定された、IEEE1212 アドレス空間にアドレスされたパケットのみを受信します。ブロードキャストパケットもまた受信します。 <ul style="list-style-type: none"> <li>– BusNumber と NodeNumber の両方がレジスタ設定値と一致</li> <li>– BusNumber がレジスタ設定と一致しかつ NodeNumber の値が'3FH'</li> <li>– BusNumber の値が'3FFH' がかつ NodeNumber がレジスタ設定と一致</li> <li>– BusNumber の値が'3FFH' がかつ NodeNumber の値が'3FH'</li> </ul> </li> </ul> <p>通常、バスリセット後の Self Identification フェイズの終了後にノードナンバーが決定するため、その値をホストが NodeNumber レジスタに設定後、このビットをセットします。</p>

## Reset Register

offset: 0Ch

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
																								RST_DMA	RST_LINK			RST_IRF	RST_ITF	RST_ARF	RST_ATF

bit 名	機能
RST_ATF<0>	Reset ATF ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = 通常状態</li> <li>● 1 = Asynchronous 送信用バッファ領域を初期化します。</li> </ul> Asynchronous 送信用バッファ領域のみを初期状態に戻します。この時、その領域にあるデータはすべて失われます。また、このバッファのステータスフラグもすべて初期状態に戻ります。'1' をセットすると、その後内部で初期化動作が完了すると自動的にこのビットは'0' にセットされます。
RST_ARF<1>	Reset ARF ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = 通常状態</li> <li>● 1 = Asynchronous 受信用バッファ領域を初期化</li> </ul> Asynchronous 受信用バッファ領域のみを初期状態に戻します。この時、その領域にあるデータはすべて失われます。また、このバッファのステータスフラグもすべて初期状態に戻ります。'1' をセットすると、その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされます。
RST_ITF<2>	Reset ARF ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = 通常状態</li> <li>● 1 = Isochronous 送信用バッファ領域を初期化します。</li> </ul> Isochronous 送信用バッファ領域のみを初期状態に戻します。この時、その領域にあるデータはすべて失われます。また、このバッファのステータスフラグもすべて初期状態に戻ります。'1' をセットすると、その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされます。
RST_IRF<3>	Reset IRF ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = 通常状態</li> <li>● 1 = Isochronous 受信用バッファ領域を初期化します。</li> </ul> Isochronous Configuration レジスタで指定された Isochronous のチャンネルの受信用バッファ領域のみを初期状態に戻します。この時、その領域にあるデータはすべて失われます。また、このバッファのステータスフラグもすべて初期状態に戻ります。'1' をセットすると、その後内部で初期状態が完了すると自動的にこのビットは'0' にセットされます。
RST_LINK<6>	Reset Link Core ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = 通常状態</li> <li>● 1 = Link Core をリセットします。</li> </ul> LINK 制御部を初期化します。
RST_DMA<7>	Reset DMA ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = 通常状態</li> <li>● 1 = DMA 制御をリセットする。</li> </ul> DMA 制御部を初期化します。

## Packet Control Register

offset: 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	
																			PKTCTL_WPEN							PKTCTL_PHY	PKTCTL_SID					

bit 名	機能
PKTCTL_SID<5>	<p>Receive Self ID ビット (RW - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>• 0 = SelfID パケットをバッファへ挿入しません。</li> <li>• 1 = SelfID パケットをバッファへ入力します。</li> </ul> <p>バスリセット後の Self ID フェイズ中に受信される Self ID パケットを、受信 Async 用のバッファ領域に入れるか入れないかを、このビットで設定します。</p>
PKTCTL_PHY<6>	<p>Receive Phy Packet ビット (RW - 初期値:0b)</p> <ul style="list-style-type: none"> <li>• 0 = Phy Packet をバッファへ挿入しません。</li> <li>• 1 = PhyPacket をバッファへ入力します。</li> </ul> <p>受信した PHY コントロールパケットを、受信 Async 用のバッファ領域に入れるか入れないかを、このビットで設定します。 また、PHY Control Packet の反転データが異なっていた場合でも、その PHY Controll Packet は、ARF バッファには、格納されません。</p>
PKTCTL_WPEN<12>	<p>Write Request Ack-Pending ビット (RW - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>• 0 = Write Request Packet に対する Ack コードで正常受信の場合 Ack-Complete を返します。</li> <li>• 1 = Write Requist Packet に対する Ack コードで正常受信の場合 Ack-Pending を返します。</li> </ul> <p>通常、Write Request Packet を受信した場合、正常に受信したら、Ack コードは ACK-Complete を返し、バッファの容量不足等で正常に受信できなかった場合は ACK-Busy を返します。このビットを'1' にセットすることで正常に受信したら、Ack コードは ACK-Pending を返します。つまり、Write Request の Split Transaction を実行することになります。ホストは Write Requist 処理が完了したら、Write Responce Packet を送信しなければなりません。</p>

## Diagnostic Status Register

offset: 1Ch

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	
						LC_RDITBUSY	LC_RDATBUSY											LC_RDRXACKST			LC_RDRXACKCOD									LC_RDTXBUSYST		

bit 名	機能
LC_RDTXBUSYST<2>	<p>Busy State ビット (R - 初期値: 0b) 受信時、このノードから返送したビジーステータスを示します。</p> <ul style="list-style-type: none"> <li>● 0 = ノンビジーステータスを返送しました。</li> <li>● 1 = Ack_busy_X ステータスを返送しました。</li> </ul>
LC_RDRXACKCOD<11:8>	<p>AT Ack ビット (R - 初期値: 0000b)</p> <ul style="list-style-type: none"> <li>● 0000 = No ack</li> <li>● 0001 = ack_complete</li> <li>● 0010 = ack_pending</li> <li>● 0011 = Reserved</li> <li>● 0100 = ack_busy_X</li> <li>● 0101 = ack_busy_A</li> <li>● 0110 = ack_busy_B</li> <li>● 0111 ~ 1100 = Reserved</li> <li>● 1101 = ack_data_error</li> <li>● 1110 = ack_type_error</li> <li>● 1111 = Reserved</li> </ul> <p>送信時、トランスミッターから送信されたパケットに対してのデスティネーションノードから返送されたアクノリッジ (Ack コード) の内容がこのレジスタに反映されます。反映されるタイミングは送信したい Asynchronous バッファ内のパケットの処理中を示す、ビジーフラグがネゲートされたときです。次のパケット送信でそのビジーがネゲートされるまで、この値は保持されます。</p>

bit 名	機能
LC_RDRXACKST	<p>&lt;13:12&gt;</p> <p>Ack Status ビット (R - 初期値: 00b)</p> <ul style="list-style-type: none"> <li>• 00 = 正常に受信</li> <li>• 01 = パリティエラー</li> <li>• 10 = パケットロスト (規定時間内でのアクノリッジパケットがこなかった)</li> <li>• 11 = 予約</li> </ul> <p>送信した Asynchronous パケットに対してデスティネーションノードから返送されてきたアクノリッジパケットのステータス状態を示します。</p>
LC_RDATBUSY	<p>&lt;24&gt;</p> <p>AT Busy ビット (R - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>• 0 = ATGo の発行可能を示します。</li> <li>• 1 = ATGo が発行できない状態を示します。現在直前に発行された ATGo によるパケット処理中を示します。</li> </ul> <p>Asynchronous 送信時, ATGo の発行でアサートし, そのアクノリッジの返送を ATAck レジスタに設定したらネゲートします。ホストはこのビットがアサート中は次の ATGo を発行できません。また発行しても無視されます。あるパケット送信がリトライ動作になった場合、そのリトライが終了するまで、このビットはネゲートされません。</p>
LC_RDITBUSY	<p>&lt;25&gt;</p> <p>IT Busy ビット (R - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>• 0 = ITGo の発行可能を示します。</li> <li>• 1 = ITGo が発行できない状態を示します。現在直前に発行された ITGo によるパケット処理中を示します。</li> </ul> <p>Isochronous 送信で、ITGo の発行でアサートし、パケットの送信終了でネゲートします。ホストはこのビットがアサート中は次の ITGo を発行できません。また発行しても無視されます。</p>

## Phy Control Register

offset: 20h

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	
PHY_PD	PHY_RTX	PHY_LPS		PHY_PC		PHY_ISOX											PHYCTL_RCVFLG	PHYCTL_PRREG	PHYCTL_PWREG					PHYCTL_PADR								PHYCTL_PWDT

bit 名	機能
PHYCTL_PWDT<7:0>	Register Data ビット (RW - 初期値: 00h) ライト要求で PHY へ転送されるデータを格納します。また、リード要求で PHY から転送されたデータが格納されます。このレジスタの内容を読み出す場合、Register Data の内容は直前のリード要求にて PHY から読み出された値が読み込まれます。つまり、ホストから書き込んだ値をこのレジスタから直接読み出すことはできません。読み出したい場合は PHY にリード要求して読み出す必要があります。
PHYCTL_PADR<11:8>	Register Address ビット (RW - 初期値: 00h) ライト要求でアクセスしたい PHY のレジスタのアドレス値を設定します。リード要求で PHY から転送されたレジスタアドレスが格納されます。
PHYCTL_PWREG<12>	Write Register ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>• 0 = 通常状態</li> <li>• 1 = ライト要求発行</li> </ul> PHY のレジスタへのライト要求を発行します。そのライト要求を行った後このビットをクリアします。
PHYCTL_PRREG<13>	Read Register ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>• 0 = 通常状態</li> <li>• 1 = リード要求発行</li> </ul> PHY のレジスタへのリード要求を発行します。そのリード要求を行った後このビットをクリアします。

bit 名	機能
PHYCTL_RCVFLG<14>	Register Data Received ビット (R - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = 通常状態</li> <li>● 1 = リード要求発行後、Register Data に PHY からのデータが格納されたことを示す。</li> </ul> PHY のレジスタへのリード要求を発行後、Register Data に PHY からのデータが格納されると '1' が設定されます。その後一度このレジスタを読み出すと '0' にクリアされます。
PHY_ISOX<25>	Isolation Control ビット (RW - 初期値: 1b) LINK と PHY の接続方式を制御するビットです。 <ul style="list-style-type: none"> <li>● 0 = LINK と PHY の間は、AC 手段で接続</li> <li>● 1 = LINK と PHY の間は、DC 手段で接続</li> </ul> LINK と PHY の AC 接続がサポートされていない場合に、このビットは必ず '1' に設定されます。
PHY_PC<28:26>	Power Class ビット (RW - 初期値: 000b) LINK と PHY のパワークラスを設定します。Bit26 ~ 28 の設定時の説明: <ul style="list-style-type: none"> <li>● 000 = ノードはバス電力を必要とせず電力を転送しない。</li> <li>● 001 = 自己給電でバスに (最小) 15W 供給する。</li> <li>● 010 = 自己給電でバスに (最小) 30W 供給する。</li> <li>● 011 = 自己給電でバスに (最小) 45W 供給する。</li> <li>● 100 = ノードは PHY のためだけに (最大 3W) バスから受電してよい。またバスに電力を供給してもよい。</li> <li>● 101 = ノードはバスから受電して最大 3W 消費する。またバスに電力を供給してよい。Link とそれより高位のレイヤー用の電力としてさらに最大 2W 必要。</li> <li>● 110 = ノードはバスから受電して最大 3W 消費する。またバスに電力を供給してよい。Link とそれより高位のレイヤー用の電力としてさらに最大 5W 必要。</li> <li>● 111 = ノードはバスから受電して最大 3W 消費する。またバスに電力を供給してよい。Link とそれより高位のレイヤー用の電力としてさらに最大 9W 必要。</li> </ul>
PHY_LPS<29>	Link Power Status ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = PHY の LPS へ、'0' を出力</li> <li>● 1 = PHY の LPS へ、'1' を出力</li> </ul>
PHY_RSTX<30>	PHY Reset ビット (RW - 初期値: 0b) PHY のリセット制御ビットです。 <ul style="list-style-type: none"> <li>● 0 = PHY のリセット動作をします。</li> <li>● 1 = PHY のリセット動作は行いません。</li> </ul> このビットを '0' に設定して、PHY をリセットした後、正常動作をする前に、必ずこのビットを '1' に設定します。
PHY_PD<31>	PHY Power Down ビット (RW - 初期値: 1b) PHY パワーを管理するためのビットです。 <ul style="list-style-type: none"> <li>● 0 = PHY をパワーダウンしていません。</li> <li>● 1 = PHY をパワーダウンしています。</li> </ul> PHY の動作を始める前に、必ずこのビットを '0' に設定して、PHY をパワーオンします。



## Cycle Timer Register

offset	31	24 23	16 15	8 7	0
28h	CYCT_SECOND		CYCT_COUNT		CYCT_OFFSET

bit 名	機能
CYCT_OFFSET<12:0>	Cycle Offset ビット (RW - 初期値: 00h) この領域は 24.576MHz のクロックでカウントアップします。Modulo3072 で動作します。
CYCT_COUNT<24:13>	Cycle Count ビット (RW - 初期値: 00h) この領域は CycleField レジスタがキャリーしたときにカウントアップし、Isochronous サイクルをカウントします。Modulo8000 で動作します。
CYCT_SECOND<31:25>	Cycle Seconds ビット (RW - 初期値: 00h) この領域は CycleCount レジスタがキャリーしたときにカウントアップし、秒をカウントします。Modulo128 で動作します。

### Isochronous Configuration Register

offset: 30h

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				
ICFG_IRTAG		ICFG_IRCHN																																	

bit 名	機能
ICFG_IRCHN<29:24>	Channel ビット (RW - 初期値: 00h) 受信したい Isochronous チャンネルを設定します。設定範囲は 0 から 63 です。
ICFG_IRTAG<31:30>	Tag ビット (RW - 初期値: 00h) 受信したい Tag を設定します。

### ATF Data Register

offset	31	24 23	16 15	8 7	0
40h	ATF data				

bit 名	機能
ATF Data<31:0>	ATF Data ビット (W - 初期値: XXXX_XXXXh) Asynchronous パケット送信データ書き込み用レジスタ。内部の Asynchronous 送信用バッファ内にデータが書かれます。

**ARF Data Register**

offset	31	24 23	16 15	8 7	0
44h	ARF data				

bit 名	機能
ARF Data<31:0>	ARF Data ビット (R - 初期値: XXXX_XXXXh) Asynchronous パケット受信データ読みだし用レジスタ。内部の Asynchronous 受信用バッファからデータが読み出されます。

**ITF Data Register**

offset	31	24 23	16 15	8 7	0
48h	ITF data				

bit 名	機能
ITF Data<31:0>	ITF Data ビット (W - 初期値: XXXX_XXXXh) Isochronous パケット送信データ書き込み用レジスタ。内部の Isochronous 送信用バッファ内にデータが書かれます。

**IRF Data Register**

offset	31	24 23	16 15	8 7	0
4Ch	IRF data				

bit 名	機能
IRF Data<31:0>	IRF Data ビット (R - 初期値: XXXX_XXXXh) Isochronous パケット受信データ読みだし用レジスタ。内部の Isochronous 受信用バッファからデータが読み出されます。

## Buffer Status and Control Register

offset: 50h

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
																	BSC_SELF							IRF_FULL	IRF_EMP	ITF_FULL	ITF_EMP	ARF_FULL	ARF_EMP	ATF_FULL	ATF_EMP

bit 名	機能
ATF_EMP<0>	<p>ATF Empty ビット (R - 初期値: 1b)</p> <ul style="list-style-type: none"> <li>0 = バッファが空ではない状態を示します。</li> <li>1 = バッファが空の状態を示します。</li> </ul> <p>ATF Data レジスタからアクセスする Asynchronous 送信バッファが空であることを示します。</p>
ATF_FULL<1>	<p>ATF Full ビット (R - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>0 = バッファが一杯ではない状態を示します。</li> <li>1 = バッファが一杯の状態を示します。</li> </ul> <p>ATF Data レジスタからアクセスする Asynchronous 送信バッファが一杯であることを示します。</p>
ARF_EMP<2>	<p>ARF Empty ビット (R - 初期値: 1b)</p> <ul style="list-style-type: none"> <li>0 = バッファが空ではない状態を示します。</li> <li>1 = バッファが空の状態を示します。</li> </ul> <p>ARF Data レジスタからアクセスする Asynchronous 受信バッファが空であることを示します。</p>
ARF_FULL<3>	<p>ARF Full ビット (R - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>0 = バッファが一杯ではない状態を示します。</li> <li>1 = バッファが一杯の状態を示します。</li> </ul> <p>ARF Data レジスタからアクセスする Asynchronous 受信バッファが一杯であることを示します。</p>

bit 名	機能
ITF_EMP<4>	<p>ITF Empty ビット (R - 初期値: 1b)</p> <ul style="list-style-type: none"> <li>● 0 = バッファ内が全て空ではないことを示します。</li> <li>● 1 = バッファ内が全て空であることを示します。</li> </ul> <p>ITF Data レジスタからアクセスする Isochronous 送信バッファが空であることを示します。</p>
ITF_FULL<5>	<p>ITF Full ビット (R - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>● 0 = バッファが一杯ではない状態を示します。</li> <li>● 1 = バッファが一杯の状態を示します。</li> </ul> <p>ITF Data レジスタからアクセスする Isochronous 送信バッファが一杯であることを示します。</p>
IRF_EMP<6>	<p>IRF Empty ビット (R - 初期値: 1b)</p> <ul style="list-style-type: none"> <li>● 0 = バッファが空ではない状態を示します。</li> <li>● 1 = バッファが空の状態を示します。</li> </ul> <p>IRF Data レジスタからアクセスする Isochronous 受信バッファが空であることを示します。</p>
IRF_FULL<7>	<p>IRF Full ビット (R - 初期値: 0b)</p> <ul style="list-style-type: none"> <li>● 0 = バッファが一杯ではない状態を示します。</li> <li>● 1 = バッファが一杯の状態を示します。</li> </ul> <p>IRF Data レジスタからアクセスする Isochronous 受信バッファが一杯であることを示します。</p>
BSC_SELFF<14:13>	<p>Select FIFO Count ビット (RW - 初期値: 00b)</p> <ul style="list-style-type: none"> <li>● 00 = ATF の空き容量を Quadlet 単位で FFCNT に示します。</li> <li>● 01 = ARF の有効データ量を Quadlet 単位で FFCNT に示します。</li> <li>● 10 = ITF の空き容量を Quadlet 単位で FFCNT に示します。</li> <li>● 11 = IRF の有効データ量を Quadlet 単位で FFCNT に示します。</li> </ul>
FFCNT<25:16>	<p>FIFO count ビット (R - 初期値: 00b)</p> <p>BSC_SELFF で選択された受信 FIFO のカウント値を Quadlet 単位で示します。</p> <ul style="list-style-type: none"> <li>● 送信 Fifo の場合は、FIFO の残り容量が表示されます。</li> <li>● 受信 Fifo の場合は、FIFO にあるデータ量が表示されます。</li> </ul>

## Interrupt Register

offset: 54h

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
PHY_LKON	PHY_CNA										INTCMDRST	INTCYARBFL	INTCYLOS	INTCYDON	INTCYSTART	INTCYSEC	INTSETNTRJ	INTHDERR	INTTCERR	INTACKERR	INTPHYRCV	INTBUSRST	INTBRSTF	INTPHYINT	INTITXEND	INTATXEND	INTIRXEND	INTARXEND	INTIRFFLU	INTARFFLU	INTRTYEXP

bit 名	機能
INTRTYEXP<0>	<p>Retry Expired ビット (RW - 初期値: 0b)</p> <p>Asynchronous パケット送信時、デスティネーションノードから Ack.Busy コードを返送してきたら、リトライフェーズに入ります。設定した最大リトライ回数に達しても、デスティネーションノードから Ack.Busy コードを返送してきた場合に、この割り込みビットを'1' にセットされます。このビットは、このビットに'1' を書き込むことで、クリアされます。</p>
INTARFFLU<1>	<p>ARF Flushed ビット (RW - 初期値: 0b)</p> <p>Asynchronous パケット受信を行ったとき、ARF バッファにルーティングされたパケットが以下の要因にて正常受信できなかったため、受信データを破棄した場合に、'1' にセットされます。</p> <ul style="list-style-type: none"> <li>Asynchronous 受信バッファに受信パケットのデータ数分、空き領域がない。</li> <li>パケットのヘッダーおよびペイロード領域が CRC エラー。</li> <li>パケットのヘッダー内のデータレンクス設定値と実際のペイロード領域のデータ数が一致しなかった。</li> </ul> <p>このビットは、このビットに'1' を書き込むことで、クリアされます。</p>
INTIRFFLU<2>	<p>IRF Flushed ビット (RW - 初期値: 0b)</p> <p>Isochronous パケット受信を行ったとき、ARF バッファにルーティングされたパケットが以下の要因にて正常受信できなかったため、受信データを破棄した場合に、'1' にセットされます。</p> <ul style="list-style-type: none"> <li>Isochronous 受信バッファに受信パケットのデータ数分、空き領域がない。</li> <li>パケットのヘッダーおよびペイロード領域が CRC エラー。</li> <li>パケットのヘッダー内のデータレンクス設定値と実際のペイロード領域のデータ数が一致しなかった。</li> </ul> <p>このビットは、このビットに'1' を書き込むことで、クリアされます。</p>

bit 名	機能
INTARXEND<3>	AsynRxEnd ビット (RW - 初期値: 0b) Asynchronous パケット受信を行ったとき、ARF バッファにデータが格納された場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTIRXEND<4>	IsoRxEnd ビット (RW - 初期値: 0b) Isochronous パケット受信を行ったとき、IRF バッファにデータが格納された場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTATXEND<5>	AsynTxEnd ビット (RW - 初期値: 0b) Asynchronous パケット送信を行った場合、送信動作を完了し、返送された送信先からの Ack コードを受信した時に'1'にセットされます。また、リトライ動作に入った場合はそのリトライフェーズが完了したとき、または途中 AckErr がセットされた場合も'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTITXEND<6>	IsoTxEnd ビット (RW - 初期値: 0b) Isochronous パケット送信を行った場合、転送したいパケットを転送し終わったときに'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTPHYINT<7>	PhyInterrupt ビット (RW - 初期値: 0b) 接続されている PHY から割り込み要因がきた場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTBRSTF<8>	BusResetFinish ビット (RW - 初期値: 0b) バスリセットが終了し、Subaction_Gap を検出した場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTBUSRST<9>	BusReset ビット (RW - 初期値: 0b) PHY はバスリセットモードになった場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTPHYRCV<10>	PhyRegRcvd ビット (RW - 初期値: 0b) PHY のレジスタへのリード要求を発行後、Phy_Control_Register の Register Data に PHY からのデータが格納された場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。

bit 名	機能
INTACKERR<11>	AckError ビット (RW - 初期値: 0b) 送信した Asynchronous パケットに対してデスティネーションから返送されてきたアクノリッジパケットが正常に受信できなかった場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTTCERR<12>	tCodeError ビット (RW - 初期値: 0b) パケット送信時、パケットヘッダー内の tCode 領域に、LLC がサポートしていないコードが設定された場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTHDERR<13>	HeadError ビット (RW - 初期値: 0b) パケット受信時、あるエラーを含むヘッダーを持つパケットを受信した場合に'1'にセットされます。受信データは破棄されます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTSENRJ<14>	SentReject ビット (RW - 初期値: 0b) Asynchronous 及び Isochronous パケット受信時、その受信バッファにそのパケット容量分の空き領域がなく、完全にそのパケットを受信できず、LLC がそのソースコードに対してビジーのアクノリッジパケットを返送した場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTCYSEC<15>	CycleSecond ビット (RW - 初期値: 0b) LLC が持っているサイクルタイマーが 1 秒をカウントした場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTCYSTART<16>	CycleStart ビット (RW - 初期値: 0b) LLC ノードが、サイクルマスターとなったときサイクルスタートパケットを発行して、または非サイクルマスターとなったときサイクルスタートパケットを受信して、新しい Isochronous サイクルがスタートした場合に'1'にセットされます。
INTCYDON<17>	CycleDone ビット (RW - 初期値: 0b) ある Isochronous サイクルが終了した場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。
INTCYLOS<18>	CycleLost ビット (RW - 初期値: 0b) LLC のノードがサイクルマスターではない場合に、あるサイクルスタートパケットを受信して内部のサイクルタイマーを更新した後、そのサイクルタイマーで 250 $\mu$ s の間、次のサイクルスタートパケットを受信できなかった場合に'1'にセットされます。このビットは、このビットに'1'を書き込むことで、クリアされます。

bit 名	機能
INTCYARBFL<19>	CycleArbitrationFail ビット (RW - 初期値: 0b) LLC のノードがサイクルマスターである場合に、サイクルスタートパケット発行の調停要求を出し、調停が失敗した場合に '1' にセットされます。このビットは、このビットに '1' を書き込むことで、クリアされます。
INTCMDRST<20>	CommandReset ビット (RW - 初期値: 0b) CSR 空間のリセット領域にアドレスされたパケットを受信した場合に '1' にセットされます。このビットは、このビットに '1' を書き込むことで、クリアされます。 ARF バッファにそのパケット分の空きがあれば、そのパケットは ARF に格納されます。そのパケット分の空きがなければ、そのパケットは ARF に格納されず、CMDRST 割り込みのみを処理します。
PHY_CNA<30>	CableNotActive ビット (RW - 初期値: 0b) 接続されている PHY が接続しているケーブルを検出していない場合に '1' にセットされます。このビットは、このビットに '1' を書き込むことで、クリアされます。
PHY_LKON<31>	LonkOn ビット (RW - 初期値: 0b) 接続されている PHY から LinkOn 信号がアサートした場合に '1' にセットされます。このビットは、このビットに '1' を書き込むことで、クリアされます。

## Interrupt Mask Register

offset: 58h

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
PHY_LKON	PHY_CNA										INTCMDRST	INTCYARBFL	INTCYLOS	INTCYDON	INTCYSTART	INTCYSEC	INTSETNTRJ	INTHDERR	INTTCERR	INTACKERR	INTPHYRCV	INTBUSRST	INTBRSTF	INTPHYINT	INTTXEND	INTATXEND	INTIRXEND	INTARXEND	INTIRFFLU	INTARFFLU	INTIRTYEXP

初期値: 0000\_0000h

Interrupt レジスタ内の各割込発生要因を INT 信号に反映させたくない場合、このレジスタでマスクします。このレジスタの並びは Interrupt レジスタと同様です。各ビットとも '1' の設定でマスクされます。なお、Interrupt レジスタで未定義の部分に当たる Mask ビットは必ずマスク ('1' を設定) して下さい。





## ARF Control Register

offset: 64h

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
																										ARC_DREQHOLD	ARC_CNTEOP	ARC_DREQFL	ARC_DREQEN	ARC_DACKEN	ARC_BMEN

bit 名	機能
ARC_BMEN<0>	Burst Mode Enable ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = BMACK 信号が随時ノンアクティブ状態になります。</li> <li>● 1 = BMREQ 信号に対してバースト転送が可能ならば BMACK で応答しバースト転送を行います。</li> </ul>
ARC_DACKEN<1>	Dack Detect Enable ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = DACK 信号に対して応答しません。</li> <li>● 1 = DACK 信号で FIFO へのアクセスを有効とします。</li> </ul>
ARC_DREQEN<2>	DREQ Enable ビット (Rw - 初期値: 0b) <ul style="list-style-type: none"> <li>● 0 = DREQ 信号が随時ノンアクティブ状態になります。</li> <li>● 1 = DREQ 信号を送出し DMA アクセスを要求します。</li> </ul>
ARC_DREQFL<3>	DMAC Flush Signal Detect Enable ビット (RW - 初期値: 0b)           一時的にバスを開放する為に DMAC からの FLX により DREQ をノンアクティブにします。 <ul style="list-style-type: none"> <li>● 0 = バス開放を行いません。</li> <li>● 1 = バスを開放する為に、FLX 信号により DREQ 信号を一時的にノンアクティブにします。</li> </ul>
ARC_CNTEOP<4>	DMA Continue or Stop Select ビット (RW - 初期値: 0b)           DMAC からの EOP により DREQEN ビットの制御を設定します。 <ul style="list-style-type: none"> <li>● 0 = EOP 検出時 DREQEN をノンアクティブにし DMA 転送を終了させます。(自動停止)</li> <li>● 1 = EOP 検出しても DREQEN を変化させません。(継続転送)</li> </ul>
ARC_DREQHOLD<5>	DREQ Signal Hold Enable ビット (RW - 初期値: 0b)           DMAC からの FLX を検出するまで、DREQ 信号の有効状態の制御を設定します。 <ul style="list-style-type: none"> <li>● 0 = DREQ 信号が、FLX を検出までのホールドをしません。</li> <li>● 1 = DREQ 信号が、FLX を検出するまでホールドされます。</li> </ul>



## IRF Control Register

offset: 6Ch

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
																										IRC_DREQHOLD	IRC_CNTEOP	IRC_DREQFL	IRC_DREQEN	IRC_DACKEN	IRC_BMEN

bit 名	機能
IRC_BMEN<0>	Burst Mode Enable ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>• 0 = BMACK 信号が随時ノンアクティブ状態になります。</li> <li>• 1 = BMREQ 信号に対してバースト転送が可能ならば BMACK で応答しバースト転送を行います。</li> </ul>
IRC_DACKEN<1>	Dack Detect Enable ビット (RW - 初期値: 0b) <ul style="list-style-type: none"> <li>• 0 = DACK 信号に対して応答しません。</li> <li>• 1 = DACK 信号で FIFO へのアクセスを有効とします。</li> </ul>
IRC_DREQEN<2>	DREQ Enable ビット (Rw - 初期値: 0b) <ul style="list-style-type: none"> <li>• 0 = DREQ 信号が随時ノンアクティブ状態になります。</li> <li>• 1 = DREQ 信号を送出し DMA アクセスを要求します。</li> </ul>
IRC_DREQFL<3>	DMAC Flush Signal Detect Enable ビット (RW - 初期値: 0b) 一時的にバスを開放する為に DMAC からの FLX により DREQ をノンアクティブにします。 <ul style="list-style-type: none"> <li>• 0 = バスを開放しません。</li> <li>• 1 = バスを開放する為に、FLX 信号により DREQ 信号を一時的にノンアクティブにします。</li> </ul>
IRC_CNTEOP<4>	DMA Continue or Stop Select ビット (RW - 初期値: 0b) DMAC からの EOP により DREQEN ビットの制御を設定します。 <ul style="list-style-type: none"> <li>• 0 = EOP 検出時 DREQEN をノンアクティブにし DMA 転送を終了させます。(自動停止)</li> <li>• 1 = EOP 検出しても DREQEN を変化させません。(継続転送)</li> </ul>
ARC_DREQHOLD<5>	DREQ Signal Hold Enable ビット (RW - 初期値: 0b) DMAC からの FLX を検出するまで、DREQ 信号の有効状態の制御を設定します。 <ul style="list-style-type: none"> <li>• 0 = DREQ 信号が、FLX を検出までのホールドをしません。</li> <li>• 1 = DREQ 信号が、FLX を検出するまでホールドされます。</li> </ul>

## 20.4 データフォーマット

### 20.4.1 アシンクロナス データフォーマット

#### クワドレット送信

リクエスト送信 (quadlet/block read request and quadlet write request):

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
												BusID	spd		tLabel				rt	tCode			priority								
destinationID												destinationOffset_H																			
destinationOffset_L																															
quadlet data(for quadlet write request)																															

レスポンス送信 (quadlet/block write response and quadlet read response):

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
												BusID	spd		tLabel				rt	tCode			priority								
destinationID												rCode				(reserved)															
(reserved)																															
quadlet data(for quadlet read response)																															

bit 名	機能
BusID	<p>BusID フィールド</p> <p>このフィールドは IEEE1394 パケットフォーマットのヘッダー領域内、ソース領域に設定する BusID をコンフィグレーションレジスタ設定を有効にするか、3FF 固定とするかを選択します。</p> <ul style="list-style-type: none"> <li>• 0 = BusID をコンフィグレーションレジスタの設定値を有効にします。</li> <li>• 1 = BusID を 3FF 固定とします。</li> </ul> <p>本ビットを'1' に設定した場合、コンフィグレーションレジスタの BusID レジスタ設定は無効となります。</p>
spd	<p>speed フィールド</p> <p>このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。</p>
tLabel	<p>Transaction Label フィールド</p> <p>このフィールドは転送されるそれぞれのトランザクションのためにユニークなタグを定義します。要求時に送った tLabel はただしく応答する場合のトランザクションラベルとして用います。</p>
rt	<p>retry フィールド</p> <p>このフィールドはこのパケットがリトライを行っているパケットかどうかを定義するために用います。そして、リトライプロトコルは相手先ノードのよってフォローされます。設定値はコード説明部分参照のこと。</p>
tCode	<p>Transaction Code フィールド</p> <p>このフィールドはトランザクションコードを設定します。トランザクションコードはパケットの形式、タイプを定義するのに用いられます。設定値はコード説明部分参照のこと。</p>
priority	<p>priority フィールド</p> <p>このフィールドはバックプレーン環境で有効になります。このため、ここでは必ず 0000b を設定しなければなりません。</p>
destinationID	<p>destination ID フィールド</p> <p>このフィールドは送信先 (destination) のバス、ノード ID を設定します。16 ビット長のうち上位 10 ビットは相手先のバス ID (destination_bus_ID) を、下位 6 ビットは相手先のフィジカル ID (destination_physical_ID) を設定します。設定値の中には特別な意味を持つものもあります。詳細はコード説明部分参照のこと。</p>
destination Offset	<p>destination Offset Address フィールド</p> <p>このフィールドは要求したパケットの送信先ノードの下位 48 ビットのアドレスを定義します。設定値は Quadlet データの読み込み要求 (read request for quadlet data)、Quadlet データの書き込み要求 (write request for quadlet data) の場合は Quadlet 単位に合わせた形で定義しなければなりません。</p>
rcode	<p>reply code フィールド</p> <p>このフィールドはこのトランザクションの結果を指定します。設定値はコード説明部分参照のこと。</p>

## ブロック送信

ブロックライト (ロック) リクエスト送信:

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
												BusID		spd	tLabel				rt	tCode				priority							
destinationID												destinationOffset_H																			
destinationOffset_L																															
dataLength																extended_tCode															
block data quadlet1																															
other block data quadlets																															
																padding (if necessary)															

ブロックリード (ロック) リスponse送信:

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
												BusID		spd	tLabel				rt	tCode				priority							
destinationID												rCode				(reserved)															
(reserved)																															
dataLength																extended_tCode															
block data quadlet1																															
other block data quadlets																															
																padding (if necessary)															

bit 名	機能
BusID	<p>BusID フィールド</p> <p>このフィールドは IEEE1394 パケットフォーマットのヘッダー領域内、ソース領域に設定する BusID をコンフィグレーションレジスタ設定を有効にするか、3FF 固定とするかを選択します。</p> <ul style="list-style-type: none"> <li>• 0 = BusID をコンフィグレーションレジスタの設定値を有効にします。</li> <li>• 1 = BusID を 3FF 固定とします。</li> </ul> <p>本ビットを'1' に設定した場合、コンフィグレーションレジスタの BusID レジスタ設定は無効となります。</p>
spd	<p>speed フィールド</p> <p>このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。</p>
tLabel	<p>Transaction Label フィールド</p> <p>このフィールドは転送されるそれぞれのトランザクションのためにユニークなタグを定義します。要求時に送った tLabel はただしく応答する場合のトランザクションラベルとして用います。</p>
rt	<p>retry フィールド</p> <p>このフィールドはこのパケットがリトライを行っているパケットかどうかを定義するために用います。そして、リトライプロトコルは相手先ノードのよってフォローされます。設定値はコード説明部分参照のこと。</p>
tCode	<p>Transaction Code フィールド</p> <p>このフィールドはトランザクションコードを設定します。トランザクションコードはパケットの形式、タイプを定義するのに用いられます。設定値はコード説明部分参照のこと。</p>
priority	<p>priority フィールド</p> <p>このフィールドはバックプレーン環境で有効になります。このため、ここでは必ず 0000b を設定しなければなりません。</p>
destinationID	<p>destination ID フィールド</p> <p>このフィールドは送信先 (destination) のバス、ノード ID を設定します。16 ビット長のうち上位 10 ビットは相手先のバス ID (destination_bus_ID) を、下位 6 ビットは相手先のフィジカル ID (destination_physical_ID) を設定します。設定値の中には特別な意味を持つものもあります。詳細はコード説明部分参照のこと。</p>
destination Offset	<p>destination Offset Address フィールド</p> <p>このフィールドは要求したパケットの送信先ノードの下位 48 ビットのアドレスを定義します。設定値は Quadlet データの読み込み要求 (read request for quadlet data)、Quadlet データの書き込み要求 (write request for quadlet data) の場合は Quadlet 単位に合わせた形で定義しなければなりません。</p>

bit 名	機能
dataLength	data length フィールド このフィールドは blockData フィールドのデータ長を設定します。設定値の最大値は speed フィールドの設定値に依存します。詳細はコード説明部分参照のこと。
extended_tCode	extended transaction code フィールド このフィールドは拡張 tCode を定義します。この extended_tCode は tCode が 'lock request' または 'lock response' の場合のみ有効になります。その他の tCode の場合はこのフィールドには 0000h を設定しなければなりません。設定値の詳細はコード説明部分参照のこと。
blockData	block data フィールド このフィールドは実際の転送データを設定します。もし、dataLength フィールドが 4 の倍数の値で定義されていない場合はこのフィールドでは 00h で Quadlet 単位が完結するように埋める必要があります。
rCode	reply code フィールド このフィールドはこのトランザクションの結果を指定します。設定値はコード説明部分参照のこと。

## クワドレット受信

リクエスト受信 (quadlet/block read request and quadlet write request):

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
destinationID												tLabel				rt		tCode				priority									
sourceID												destinationOffset_H																			
destinationOffset_L																															
quadlet data(for quadlet write request)																															
spd																AckSent															

レスポンス受信 (quadlet/block write response and quadlet read response):

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
destinationID												tLabel				rt		tCode				priority									
sourceID												rcode				(reserved)															
(reserved)																															
quadlet data(for quadlet read response)																															
spd																AckSent															

bit 名	機能
destinationID	destination ID フィールド このフィールドは送信先 (destination) のバス、ノード ID を設定します。16 ビット長のうち上位 10 ビットは相手先のバス ID (destination_bus_ID) を、下位 6 ビットは相手先のフィジカル ID (destination_physical_ID) を設定します。設定値の中には特別な意味を持つものもあります。詳細はコード説明部分参照のこと。
tLabel	Transaction Label フィールド このフィールドは転送されるそれぞれのトランザクションのためにユニークなタグを定義します。要求時に送った tLabel はただしく応答する場合のトランザクションラベルとして用います。
rt	retry フィールド このフィールドはこのパケットがリトライを行っているパケットかどうかを定義するために用います。そして、リトライプロトコルは相手先ノードのよってフォローされます。設定値はコード説明部分参照のこと。
tCode	Transaction Code フィールド このフィールドはトランザクションコードを設定します。トランザクションコードはパケットの形式、タイプを定義するのに用いられます。設定値はコード説明部分参照のこと。
priority	priority フィールド このフィールドはバックプレーン環境で有効になります。このため、ここでは必ず 0000b を設定しなければなりません。
sourceID	source ID フィールド このフィールドは送信元 (source) のバス、ノード ID が格納されています。16 ビット長のうち上位 10 ビットは相手先のバス ID (destination_bus_ID) を、下位 6 ビットは相手先のフィジカル ID (destination_physical_ID) になります。値の中には特別な意味を持つものもあります。詳細はコード説明部分参照のこと。
destination Offset	destination Offset Address フィールド このフィールドは要求したパケットの送信先ノードの下位 48 ビットのアドレスを定義します。設定値は Quadlet データの読み込み要求 (read request for quadlet data)、Quadlet データの書き込み要求 (write request for quadlet data) の場合は Quadlet 単位に合わせた形で定義しなければなりません。
quadlet Data	quadlet data フィールド このフィールドは転送されてきたデータが格納されています。
spd	speed フィールド このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。
AckSent	ackSent フィールド このフィールドはこのパケットを受信した後、そのアクノリッジとして送り返した Ack コードが格納されています。設定値はコード説明部分参照のこと。
rcode	reply code フィールド このフィールドはこのトランザクションの結果を指定します。設定値はコード説明部分参照のこと。

## ブロック受信

ブロックライト (ロック) リクエスト受信:

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
destinationID											tLabel				rt		tCode			priority											
sourceID											destinationOffset_H																				
destinationOffset_L																															
dataLength																extended_tCode															
block data quadlet1																															
other block data quadlets																															
spd																AckSent															

ブロックリード (ロック) リスponse受信:

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
destinationID											tLabel				rt		tCode			priority											
sourceID											rnode				(reserved)																
(reserved)																															
dataLength																extended_tCode															
block data quadlet1																															
other block data quadlets																															
spd																AckSent															

bit 名	機能
destinationID	destination ID フィールド このフィールドは送信先 (destination) のバス、ノード ID を設定します。16 ビット長のうち上位 10 ビットは相手先のバス ID (destination_bus_ID) を、下位 6 ビットは相手先のフィジカル ID (destination_physical_ID) を設定します。設定値の中には特別な意味を持つものもあります。詳細はコード説明部分参照のこと。
tLabel	Transaction Label フィールド このフィールドは転送されるそれぞれのトランザクションのためにユニークなタグを定義します。要求時に送った tLabel はただしく応答する場合のトランザクションラベルとして用います。
rt	retry フィールド このフィールドはこのパケットがリトライを行っているパケットかどうかを定義するために用います。そして、リトライプロトコルは相手先ノードのよってフォローされます。設定値はコード説明部分参照のこと。
tCode	Transaction Code フィールド このフィールドはトランザクションコードを設定します。トランザクションコードはパケットの形式、タイプを定義するのに用いられます。設定値はコード説明部分参照のこと。
priority	priority フィールド このフィールドはバックプレーン環境で有効になります。このため、ここでは必ず 0000b を設定しなければなりません。
sourceID	source ID フィールド このフィールドは送信元 (source) のバス、ノード ID が格納されています。16 ビット長のうち上位 10 ビットは相手先のバス ID (destination_bus_ID) を、下位 6 ビットは相手先のフィジカル ID (destination_physical_ID) になります。値の中には特別な意味を持つものもあります。詳細はコード説明部分参照のこと。
destination Offset	destination Offset Address フィールド このフィールドは要求したパケットの送信先ノードの下位 48 ビットのアドレスを定義します。設定値は Quadlet データの読み込み要求 (read request for quadlet data)、Quadlet データの書き込み要求 (write request for quadlet data) の場合は Quadlet 単位に合わせた形で定義しなければなりません。
dataLength	data length フィールド このフィールドは blockData フィールドのデータ長を設定します。設定値の最大値は speed フィールドの設定値に依存します。詳細はコード説明部分参照のこと。
extended_tCode	extended transaction code フィールド このフィールドは拡張 tCode を定義します。この extended_tCode は tCode が 'lock request' または 'lock response' の場合のみ有効になります。その他の tCode の場合はこのフィールドには 0000h を設定しなければなりません。設定値の詳細はコード説明部分参照のこと。
blockData	block data フィールド このフィールドは実際の転送データを設定します。もし、dataLength フィールドが 4 の倍数の値で定義されていない場合はこのフィールドでは 00h で Quadlet 単位が完結するように埋める必要があります。
spd	speed フィールド このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。
AckSent	ackSent フィールド このフィールドはこのパケットを受信した後、そのアクノリッジとして送り返した Ack コードが格納されています。設定値はコード説明部分参照のこと。
rCode	reply code フィールド このフィールドはこのトランザクションの結果を指定します。設定値はコード説明部分参照のこと。

## 20.4.2 アシクロナス ストリーム データフォーマット

### アシクロナスストリーム送信

アシクロナスストリーム送信:

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		
														spd																			
dataLength														tag		channel								sy									
Asynchronous Stream Data																																	
padding (if necessary)																																	

bit 名	機能
tCode	Transaction Code フィールド このフィールドはトランザクションコードを設定します。トランザクションコードはパケットの形式、タイプを定義するのに用いられます。値は 0xA でなければなりません。
dataLength	data length フィールド このフィールドは blockData フィールドのデータ長を設定します。
tag	tag フィールド このフィールドはアシクロナスストリーム転送の tag を格納します。
channel	channel フィールド このフィールドはアシクロナスストリーム転送チャンネルの番号を格納します。
spd	speed フィールド このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。
sy	sync フィールド このフィールドはアシクロナスストリーム転送の sync を格納します。
Asynchronous Stream Data	Asynchronous Stream Data フィールド このフィールドは実際の転送データを設定します。もし、dataLength フィールドが 4 の倍数の値で定義されていない場合はこのフィールドでは 00h で Quadlet 単位が完結するように埋める必要があります。

アシンクロナスストリーム受信:

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
dataLength												tag		channel				tCode			sy										
Asynchronous Stream Data																															
spd																errCode															

bit 名	機能
dataLength	data length フィールド このフィールドは blockData フィールドのデータ長を設定します。
tag	tag フィールド このフィールドはアシンクロナスストリーム転送の tag を格納します。
channel	channel フィールド このフィールドはアシンクロナスストリーム転送チャンネルの番号を格納します。
sy	sync フィールド このフィールドはアシンクロナスストリーム転送の sync を格納します。
tCode	Transaction Code フィールド このフィールドはトランザクションコードを設定します。トランザクションコードはパケットの形式、タイプを定義するのに用いられます。値は 0xA でなければなりません。
Asynchronous Stream Data	Asynchronous Stream Data フィールド このフィールドは実際の転送データを設定します。
spd	speed フィールド このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。
errCode	errCode フィールド このフィールドはこのパケットを受信した時のアクノリッジが格納されます。ただし、Asynchronous 転送と異なりこのフィールドの値は返送されません。設定値はコード説明部分参照のこと。

## 20.4.3 アイソクロナス データフォーマット

送信

アイソクロナス送信 (ノーマルモード):

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
dataLength														tag		channel						spd		sy							
Asynchronous Stream Data																															
sy																															

bit 名	機能
dataLength	data length フィールド このフィールドは blockData フィールドのデータ長を設定します。
tag	tag フィールド このフィールドはアイソクロナス転送の tag を格納します。
channel	channel フィールド このフィールドはアイソクロナス転送チャンネルの番号を格納します。
spd	speed フィールド このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。
sy	sync フィールド このフィールドはアシンクロナス転送の sync を格納します。
Isochronous Data	Isochronous Data フィールド このフィールドは実際の転送データを設定します。もし、dataLength フィールドが 4 の倍数の値で定義されていない場合はこのフィールドでは 00h で Quadlet 単位が完結するように埋める必要があります。

## 受信

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
dataLength																tag		channel				tCode			sy						
Isochronous Stream Data																															
																sy															

bit 名	機能
dataLength	data length フィールド このフィールドは blockData フィールドのデータ長を設定します。
tag	tag フィールド このフィールドはアイソクロナス転送の tag を格納します。
channel	channel フィールド このフィールドはアイソクロナス転送チャンネルの番号を格納します。
sy	sync フィールド このフィールドはアシンクロナス転送の sync を格納します。
tCode	Transaction Code フィールド このフィールドはトランザクションコードを設定します。トランザクションコードはパケットの形式、タイプを定義するのに用いられます。値は 0xA でなければなりません。
spd	speed フィールド このフィールドは転送する速度を設定します。設定値はコード説明部分参照のこと。
errCode	errCode フィールド このフィールドはこのパケットを受信した時のアクノリッジが格納されます。ただし、Asynchronous 転送と異なりこのフィールドの値は返送されません。設定値はコード説明部分参照のこと。

## 20.4.4 自己識別パケット送信

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
																											1110b				
SelfID Packet Data																															

bit 名	機能
SelfID Packet Data	SelfID Packet Data フィールド Bit31 が 1、Bit30 が 0 である。



トランザクションコード (tCode):

コード	説明	コード	説明
0h	Write request for data quadlet	8h	Cycle start
1h	Write request for data block	9h	Lock request
2h	Write response	Ah	Isochronous data block
3h	Reserved	Bh	Lock response
4h	Read request for data quadlet	Ch	Reserved
5h	Read request for data block	Dh	Reserved
6h	Read response for data quadlet	Eh	Reserved (internal use)
7h	Read response for data block	Fh	Reserved

Bus Number/Node Number 表:

Destination Bus_ID	Destination Node_ID	内容
0-3Feh	0-3Eh	Bus_ID 及び node_ID で定義されたノードに転送
3FFh	0-3Eh	ローカルバス内の node_ID で定義されたノードに転送
0-3Feh	3Fh	Bus_ID で定義されたバスにブロードキャスト転送
3FFh	3Fh	ローカルバス内にブロードキャスト転送

拡張トランザクションコード (Extended tCode) 表:

コード	説明
0000h	Reserved
0001h	Mask_swap
0002h	Compare_swap
0003h	Fetch_add
0004h	Little_add
0005h	Bounded_add
0006h	Wrap_add
0007h	Vender_dependant
0008h-FFFFh	Reserved

Speed コード (spd) 表:

コード	Speed
00b	100Mbps
01b	200Mbps
10b	400Mbps
Others	Reserved

アクリッジコード (Ack) 表:

コード	説明
0h	Reserved
1h	Ack_complete
2h	Ack_pending
3h	Reserved
4h	Ack_busy_X
5h	Ack_busy_A
6h	Ack_busy_B
7h	Reserved
8h	Reserved
9h	Reserved
Ah	Reserved
Bh	Reserved
Ch	Reserved
Dh	Ack_data_error
Eh	Ack_type_error
Fh	Reserved

アシンクロナスパケット内応答コード (rcode) 表:

コード	説明	内容
0h	Resp_complete	ノードは要求された動作を成功裡に完了した
1h	Reserved	未使用
2h	Reserved	未使用
3h	Reserved	未使用
4h	Resp_conflict_error	応答しているノードによって資源の衝突が検出された
5h	Resp_data_error	ハードウェアエラー、データは有効ではない
6h	Resp_type_error	パケットタイプエラー又は無効なエラーを格納している
7h	Resp_address_error	指定されたノード内のアドレス位置はアクセスできない
8h	Reserved	未使用
9h	Reserved	未使用
Ah	Reserved	未使用
Bh	Reserved	未使用
Ch	Reserved	未使用
Dh	Reserved	未使用
Eh	Reserved	未使用
Fh	Reserved	未使用

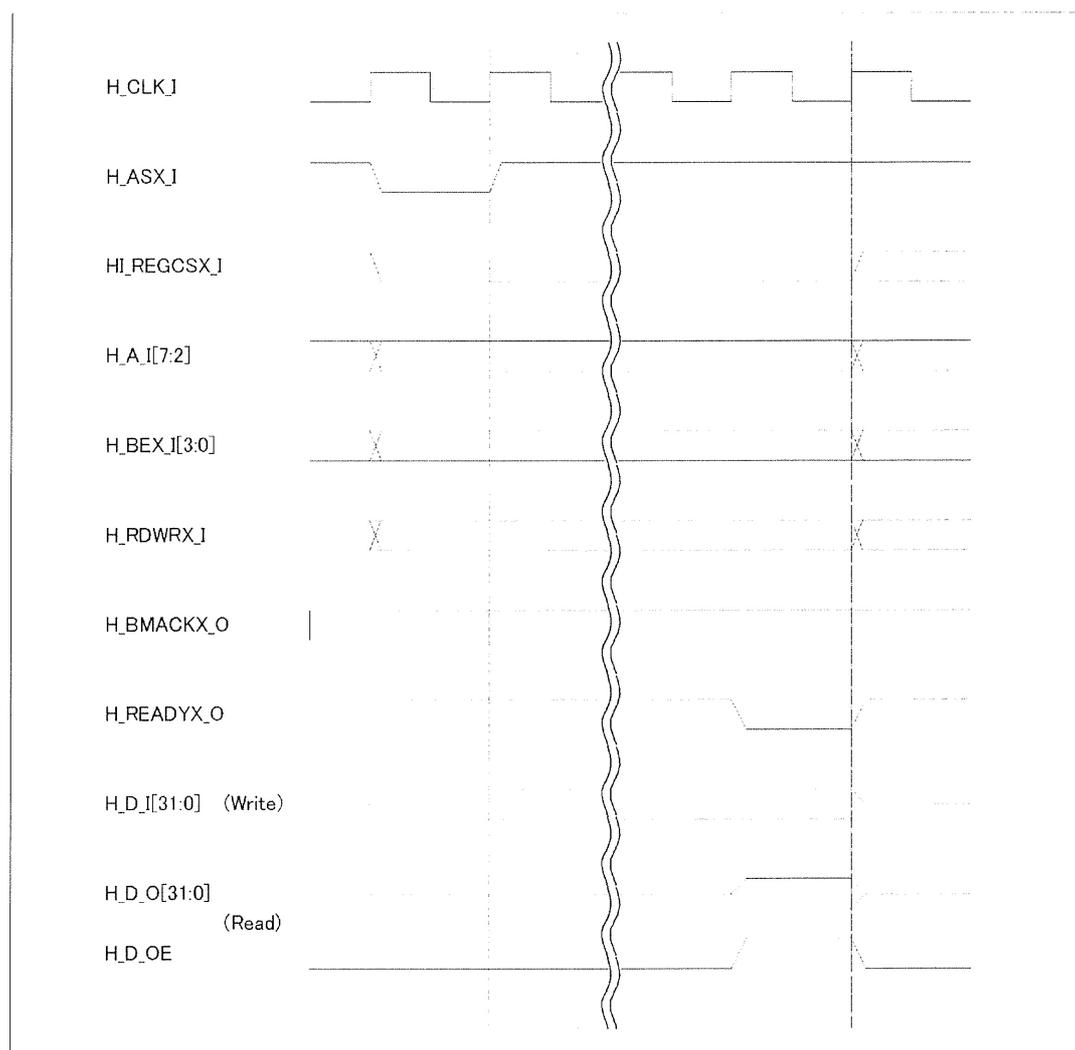
## 20.5 機能

### 20.5.1 ホストインターフェース

LLC の制御及び、送信/受信データの転送はすべてホストインターフェースから行われます。内部レジスタ及び、IEEE1394 のパケットデータは基本的に 32 ビット幅でアクセスします。

#### レジスタアクセス

以下は、レジスタへのアクセスタイミングを示します。

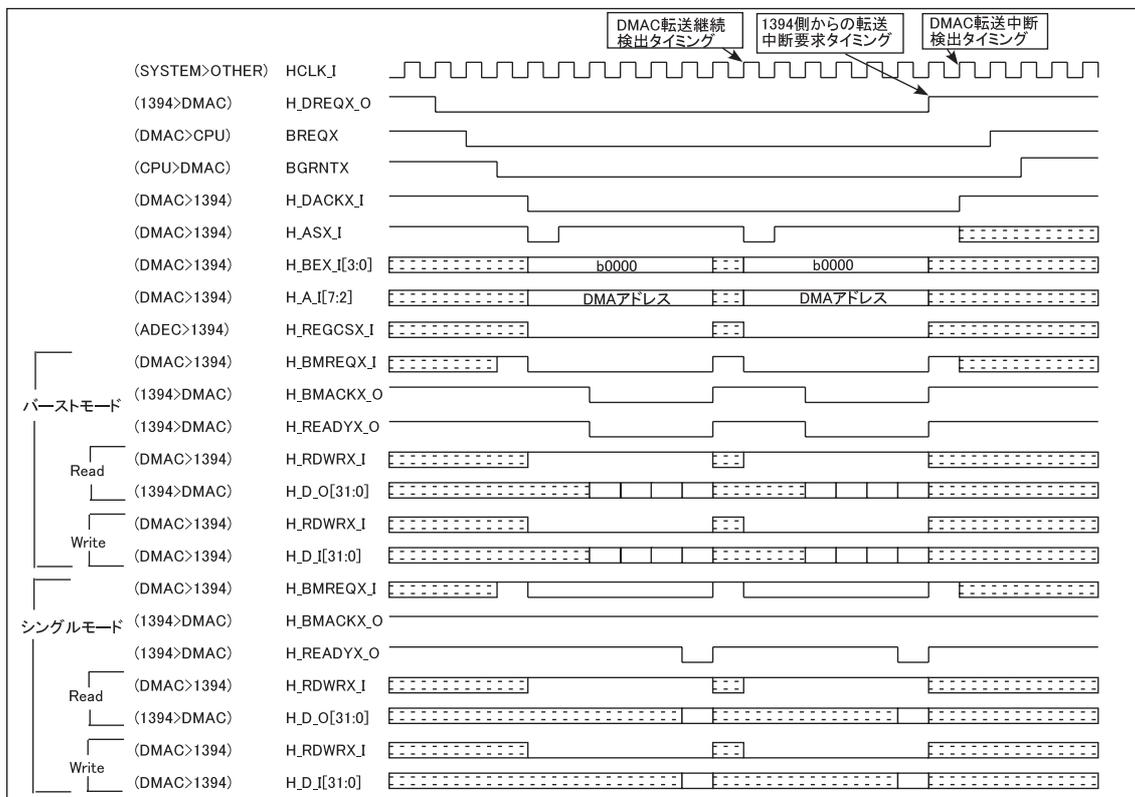


## DMA 転送

送信/受信バッファとのデータ転送方法に DMA 転送の機能をサポートしています。すべての送受信バッファにそれぞれ 1DMA 転送チャンネルを専用にするのに、「アシンクロナス送信バッファDMA チャンネル」と「アシンクロナス受信バッファDMA チャンネル」と「アイソクロナス送信バッファDMA チャンネル」と「アイソクロナス受信バッファDMA チャンネル」の四つの DMA 転送チャンネルを設けています。各 DMA チャンネルは、それぞれの制御レジスタを持っていて、独自の制御設定ができます。

送信データを送信バッファに DMA 転送で転送する時は、DMAC に転送したいデータの大きさを設定し、実行する。その後 DREQEN レジスタビットを '1' にすることで DREQ 要求が DMAC に対して発行されるため、DMA 転送が実行されます。

受信データを受信バッファからホスト側に DMA 転送で転送する場合は、FFCNT レジスタビットからデータ長を読み込んでその値を DMAC にセットし、DMAC を実行する。その後 DREQEN レジスタビットを '1' にすることで DREQ 要求が DMAC に対して発行されるため、DMA 転送が実行されます。



送信バッファ(ATF と ITF) への DMA 転送には、TGOLEN レジスタビットによる DMA 転送モードと TGOEOP レジスタビットによる DMA 転送モードがあります。

TGOLEN レジスタビットにより DMA 転送する場合には、PKTLEN レジスタに設定された値になった時、DREQ をディセーブルし、TGo(ATGo 又は ITGo) を発行して自動送信をさせ、次のパケットを送信出来るようになった時、DREQ をイネーブルします。

TGOEOP レジスタビットにより DMA 転送する場合には、DMAC からの EOPX を検出時、DREQ をディセーブルし、TGo(ATGo 又は ITGo) を発行して、自動送信させます。

#### 割り込み処理

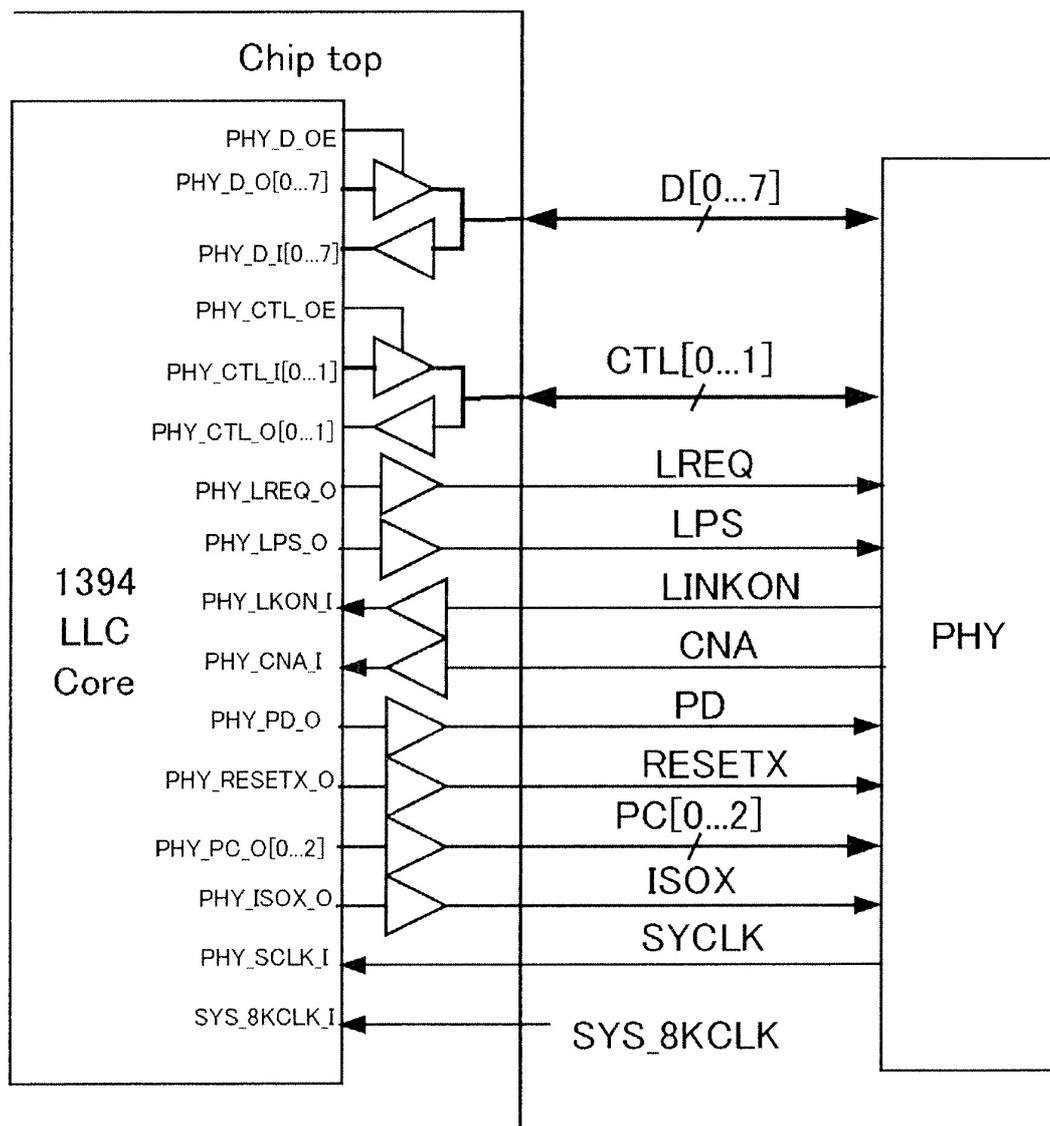
LLC の Interrupt レジスタ、Interrupt Mask レジスタで定義された割り込み要因についてホストに通知する手段として H<sub>INTX</sub>\_O 端子を持っています。この H<sub>INTX</sub>\_O 信号はアクティブローで Interrupt Mask レジスタにてマスクされていない割り込み要因の OR を取った形でアサートされます。

Interrupt レジスタ内のすべてのビットに'1' をライトすることで、各ビットがクリアされ、H<sub>INTX</sub>\_O 端子はネゲートされます。

## 20.5.2 PHY インターフェース

PHY とのインターフェースは、SYCLK と LREQ と CTL[0:1] と CTL[0:7] などによって構成されます。転送速度によって使われる D のビット数が違います。100Mbps 転送速度には D[0:1] を使って、200Mbps 転送速度には D[0:3] を使って、400Mbps 転送速度には D[0:7] を使います。

下記の図では、LLC と PHY の DC 接続について示します。PHY からの CAN 信号と接続する必要がない場合に、LLC の PHY\_CAN\_I 入力はローに固定してください。また、外部から 8KHz クロックを提供していないときに、LLC の SYS\_8KCLK\_I 入力もローに固定してください。



## PHY コントロール

IEEE1394 リンクレイヤーコントローラ (LLC) は、PHY をコントロールするために、4 種類の動作モードで定義される通信手段を使って行われます。それぞれの動作モードは CTL[0:1] 端子の条件によって定義されます。

PHY が LINK-PHY バスを駆動する場合:

PHY_CTLI[0:1]	動作	内容
00b	Idle	何も動作しておらずアイドル状態 (デフォルトモード)
01b	Status	PHY チップからステータス情報の転送が行われている
10b	Receive	PHY チップから受信パケット内容の転送が行われている
11b	Grant	リンクレイヤーコントローラは、送信パケットを PHY チップに転送するために、PHY-LINK バスの駆動を許されている

リンクレイヤーコントローラが LINK-PHY バスを駆動する場合:

PHY_CTLI[0:1]	動作	内容
00b	Idle	リンクレイヤーコントローラが転送完了したために、PHY-LINK のバスを開放する
01b	Hold	リンクレイヤーコントローラは送出するデータを準備する間バスをホールドしている。または、現在のデータ転送を完了して、次のパケットの送出前に調停せずに所有権を保有しようとしている
10b	Transmit	PHY チップに送信パケットのデータを転送している
11b	Reserved	未使用

## サービス要求

シリアルバスにアクセスまたは、PHY のレジスタにアクセスするための要求として、リンクレイヤーコントローラは PHY\_LREQ\_O 端子に短いシリアルストリームを送ります。ストリームには要求するタイプ、転送するパケットの速度、読み出しまたは書き込みコマンドの情報を含んでいます。要求の種類によって、ストリームのビット長とフォーマットが異なります。



## シリアルバス要求

シリアルバス要求は、PHY\_LREQ\_O に以下の 8 ビット長のフォーマットにて行われます (P1394a)。

## フォーマット:

Bit	タイプ	内容
0	Start Bit	転送開始を示す。いつも'1'を転送する
1-3	Request Type	要求のタイプを示す。要求タイプ表を参照すること
4-6	Request Speed	要求する PHY チップの転送速度を示す。転送速度表を参照すること。
7	Stop Bit	転送完了を示す。いつも'0'を転送する。

## 要求タイプ表:

PHY_LREQ_O[1:3]	タイプ	内容	対応転送
000b	ImmReq	即時バス要求	アシンクロナスのアクノリッジ パケット転送
001b	IsoReq	アイソクロナス要求	アイソクロナスパケット転送
010b	PriReq	優先バス要求	サイクルスタートパケット転送
011b	FairReq	平等バス要求	アシンクロナス転送
100b	RdReg	PHY レジスタ読み出しバス要求	PHY レジスタリード
101b	WrReg	PHY レジスタ書き込み要求	PHY レジスタライト
110b	AccCtrl	アシンクロナス調停加速制御要求	アシンクロナス転送
111b	Reserved	未使用	未使用

## 転送速度表:

PHY_LREQ_O[4:6]	Data Rate
000b	100Mbps
010b	200Mbps
100b	400Mbps

## PHY レジスタ読み出し要求

PHY レジスタ読み出し要求は、PHY\_LREQ\_O に以下の 9 ビット長のフォーマットにて行われます。

## フォーマット:

Bit(s)	タイプ	内容
0	Start Bit	転送開始を示す。いつも'1'を転送する
1-3	Request Type	要求のタイプを示す。要求タイプ表を参照すること
4-7	Address	読み出す PHY のレジスタのアドレスを示す。
8	Stop Bit	転送完了を示す。いつも'0'を転送する。

## PHY レジスタ書き込み要求

PHY レジスタ書き込み要求は、PHY\_LREQ\_O に以下の 17 ビット長のフォーマットにて行われます。

フォーマット:

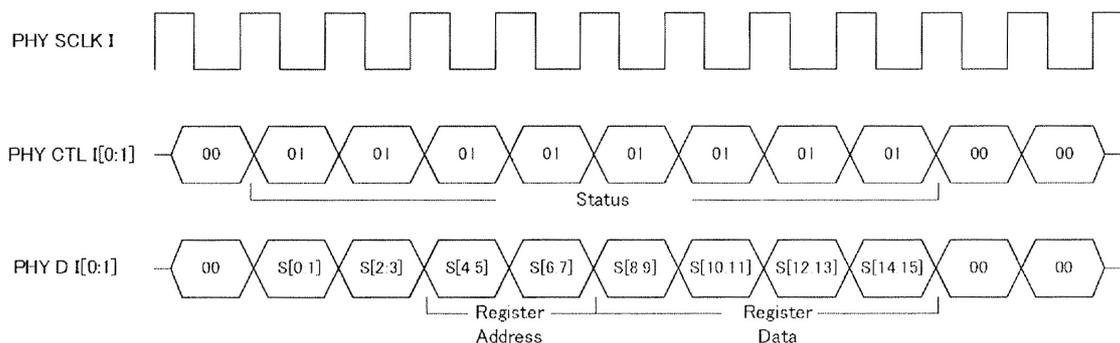
Bit(s)	タイプ	内容
0	Start Bit	転送開始を示す。いつも'1'を転送する
1-3	Request Type	要求のタイプを示す。要求タイプ表を参照すること
4-7	Address	書き込む PHY のレジスタのアドレスを示す。
8-15	Data	書き込む PHY のレジスタのデータを示す。
16	Stop Bit	転送完了を示す。いつも'0'を転送する。

## 転送

## ステータス転送

PHY は、ステータス情報を持つ又はバスアイドルを検出した時、ステータス転送を起動します。PHY は PHY\_CTLI[0:1] 線を 01b(ステータスを示す) に駆動し、PHY\_DI[0:1] でステータスを配信します。PHY は、通常に最初の 4 ステータスビット (S[0:1] と S[2:3]) だけを PHY/Link インターフェースに転送します。もし別のノードからパケットを受信したら、このステータス転送を中断します。

PHY レジスタ読み出し要求によって、PHY は 16 ビットの完整ステータスパケットを PHY/Link インターフェースに転送します。

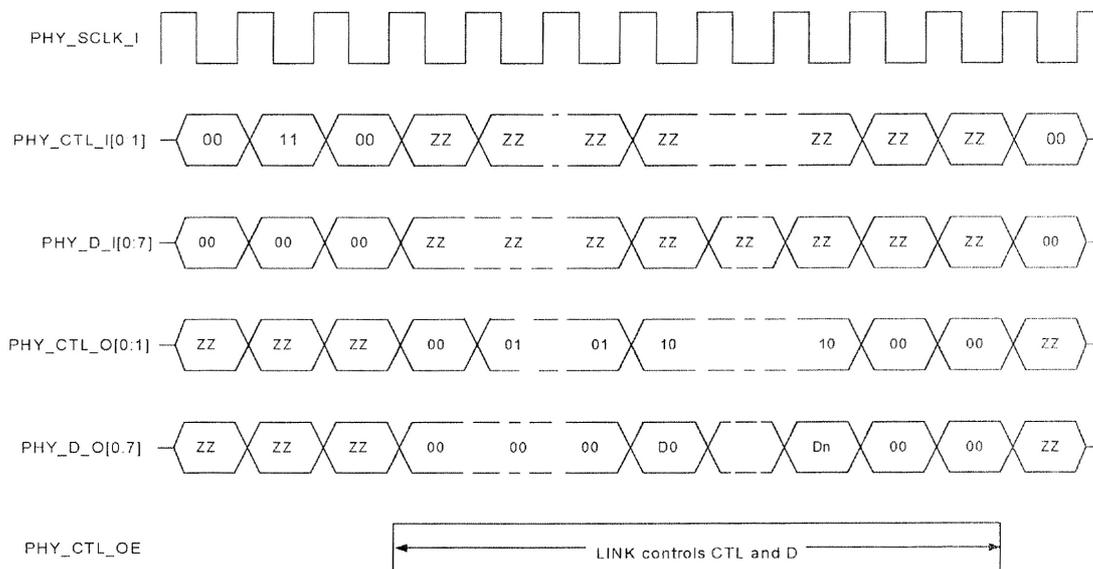


各ビットの意味:

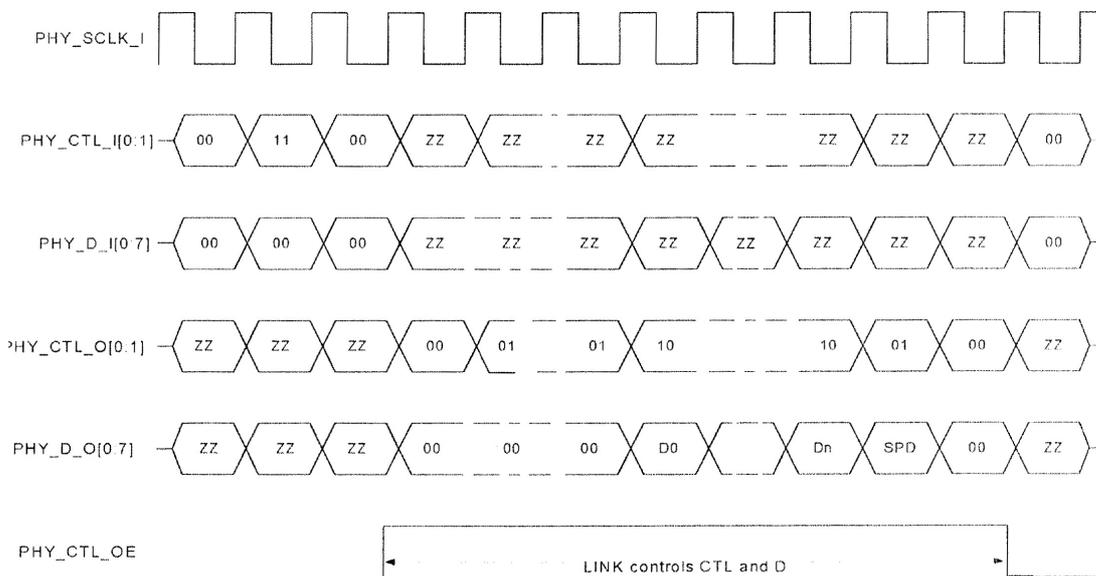
Bit(Sn)	タイプ	内容
0	Arbitration Reset Gap	Arbitration Reset Gap を検出
1	Subaction Gap	Subaction Gap を検出
2	Bus Reset	Bus Reset を検出
3	PHY Interrupt	PHY 割り込み発生
4-7	Address	PHY レジスタのアドレス
8-15	Data	レジスタデータ

パケット送信

Single Packet 送信:



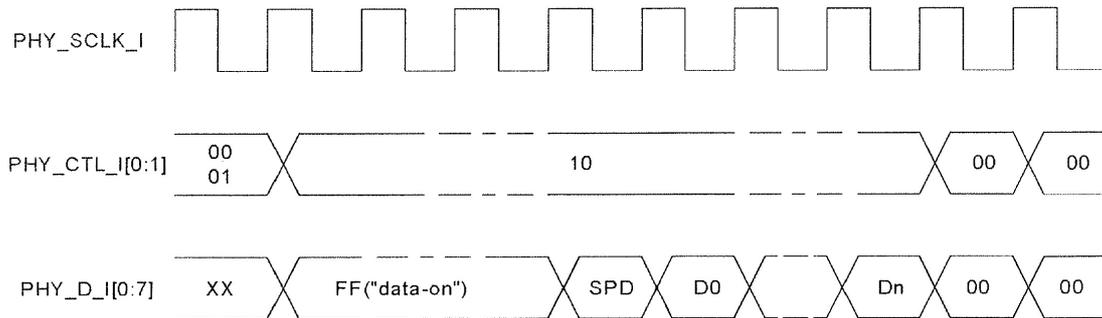
Concatenated Packet 送信:



SPD コード表:

SPD[0:7]	Data Rate
00XXXXXX	100Mbps
0100XXXX	200Mbps
01010000	400Mbps

## パケット受信



注: アイドル状態からパケット受信状態に入れば、PHY\_CTL\_I[0:1] は、00b から 10b に変更します。ただし、ステータス転送状態が中断されて受信状態に入る場合、PHY\_CTL\_I[0:1] は 01b から 10b に変更します。

### 20.5.3 バッファのコントロール

IEEE1394 の送信、受信ともホスト側はパケット単位でバッファにアクセスします。

Buffer\_Status\_and\_Control\_Register によって、バッファの Empty/Full ステータスの検視ができます。Reset レジスタ内の各バッファのリセットビットを行った場合は、リセットされたバッファは初期状態に戻り、その内のデータは全て失われ、Empty フラグがアクティブになります。

#### 送信用バッファ

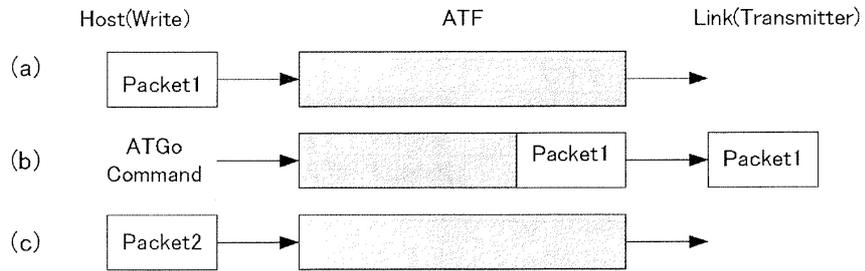
Isochronous パケットを送信するとき、送信したい Isochronous パケットを Isochronous 送信バッファ (ITF) に書き込んで、ITGo で送信コマンドを発行します。Isochronous パケット送信終了後、Isochronous 送信バッファ内に、送信した Isochronous パケットの格納された領域のみが空けられます。ホストは ITXEND 割り込みにて、その Isochronous パケット送信の終了がわかります。

Asynchronous パケットを送信するとき、送信したい Asynchronous パケットを Asynchronous 送信バッファ (ATF) に書き込んで、ATGo で送信コマンドを発行します。現在送信中のパケットデータは、そのデスティネーションコードから Complete、Pending の Ack コードの返答があるまで ATF 内にバッファリングされたままです。よって Ack コードがビジーの時は、そのリトライフェーズ中そのバッファリングしているデータを、リトライ回数内繰り返し送信します。その間に Complete、Pending Ack コードの返答があった及びリトライカウント値が設定した最大リトライ回数を超えた場合にそのパケットが入っている ATF 内の領域のみを自動的にフラッシュします。リトライフェーズ中にエラーの Ack コードを受けた場合、設定した最大リトライ回数に満たなくても、その時点でフェーズを中止し、ATF をフラッシュします。ホストは、ATXEND 割り込みにて、そのパケット送信の終了がわかります。

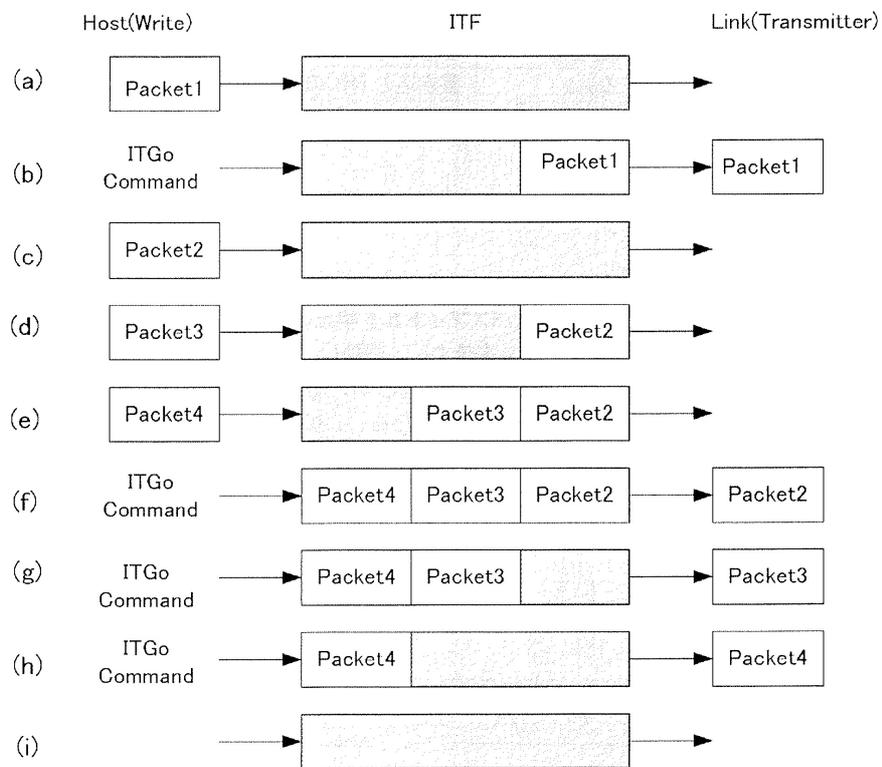
基本的に、Asynchronous 送信時、ATGo を発行した時点で ATF に入っているパケットは 1 パケットのみで、その送信パケットの転送が終了し、ATXEND 割り込みを検出した後、次の転送したいパケットが ATF に入り始めることができます。

Isochronous 送信の場合に、複数の転送したい Isochronous パケットが ITF に入ることができます。ITGo コマンドの発行で、一番前の Isochronous パケットを転出して、ITXEND 割り込みが発生します。次の Isochronous サイクルで、ITGo コマンドの発行により、次の Isochronous パケットを転送できます。

Asynchronous Stream パケットを送信したい時、送信データフォーマットで Asynchronous Stream パケットを Asynchronous 送信バッファ(ATF) に格納して、ATGo で転送コマンドを発行して、Asynchronous Stream パケットの送信ができます。



ATF動作概念図



ITF動作概念図

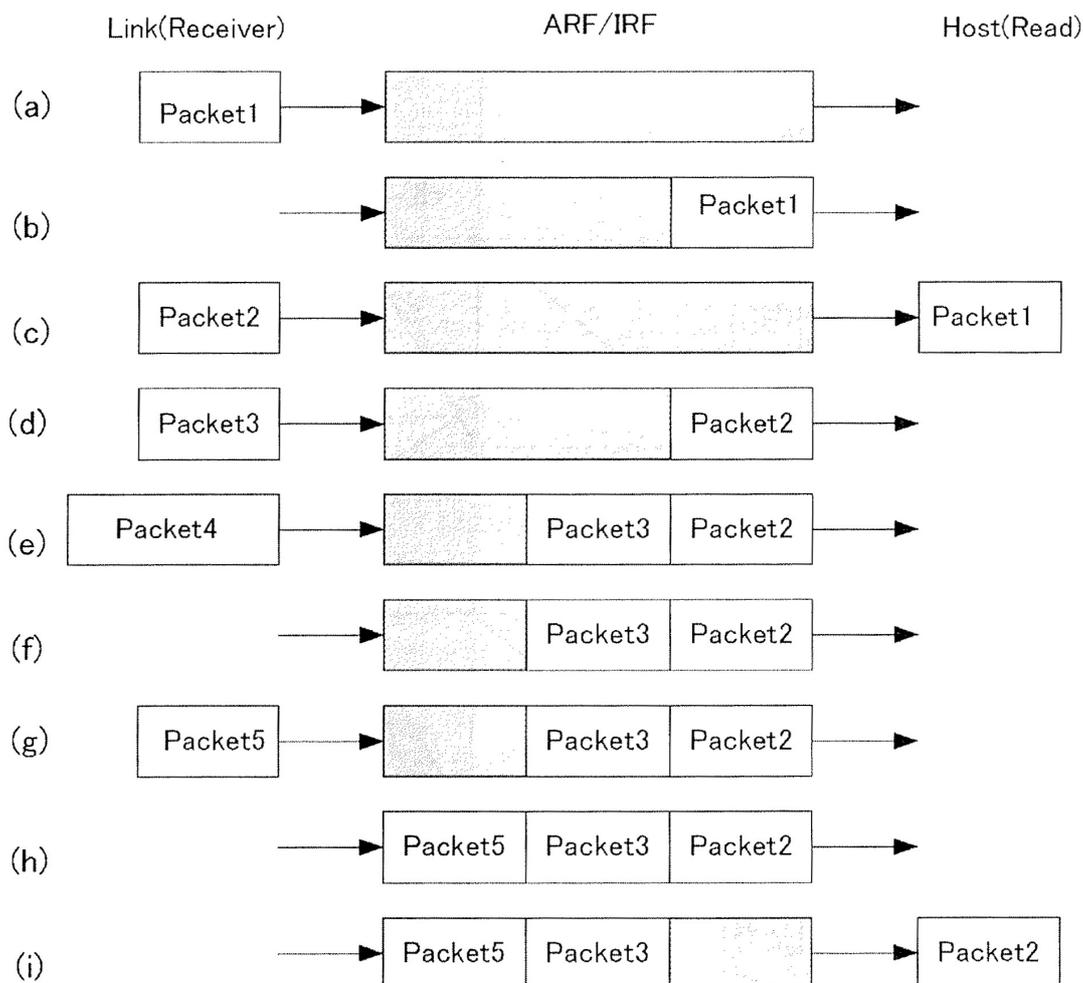
## 受信用バッファ

受信した Isochronous パケットが Isochronous 受信バッファ(IRF) に全て格納されて、IRXEND 割り込みが発生します。この割り込みの検出によって、ホスト側は Isochronous パケットを受信したのがわかって、Isochronous 受信バッファ(IRF) から受信した Isochronous パケットを読みだせます。

受信した Asynchronous パケットが Asynchronous 受信バッファ(ARF) に全て格納されて、ARXEND 割り込みが発生します。この割り込みの検出によって、ホスト側は Asynchronous 受信バッファにパケットを受信したのがわかって、Asynchronous 受信バッファ(ARF) から受信した Asynchronous パケットを読み出せます。ARF バッファ内に受信したパケット分の空き領域がない場合に、Ack\_Busy\_X コードを送信元ノードに返送し、そのパケットの再送要求 (Retry) を行うことになります。

受信したパケットのデータ数分バッファ空き領域がない、そのパケットのヘッダー及びペイロード領域がCRC エラー、ヘッダー内のデータレンクス領域値と実際のペイロード領域内のデータ数が一致しない、の条件で、そのパケットを自動的にフラッシュします。

Asynchronous Stream パケット (Isochronous サイクル外の Isochronous パケット) を受信した時、受信データフォーマットで受信した Asynchronous Stream パケットを Asynchronous 受信バッファ(ARF) に格納します。



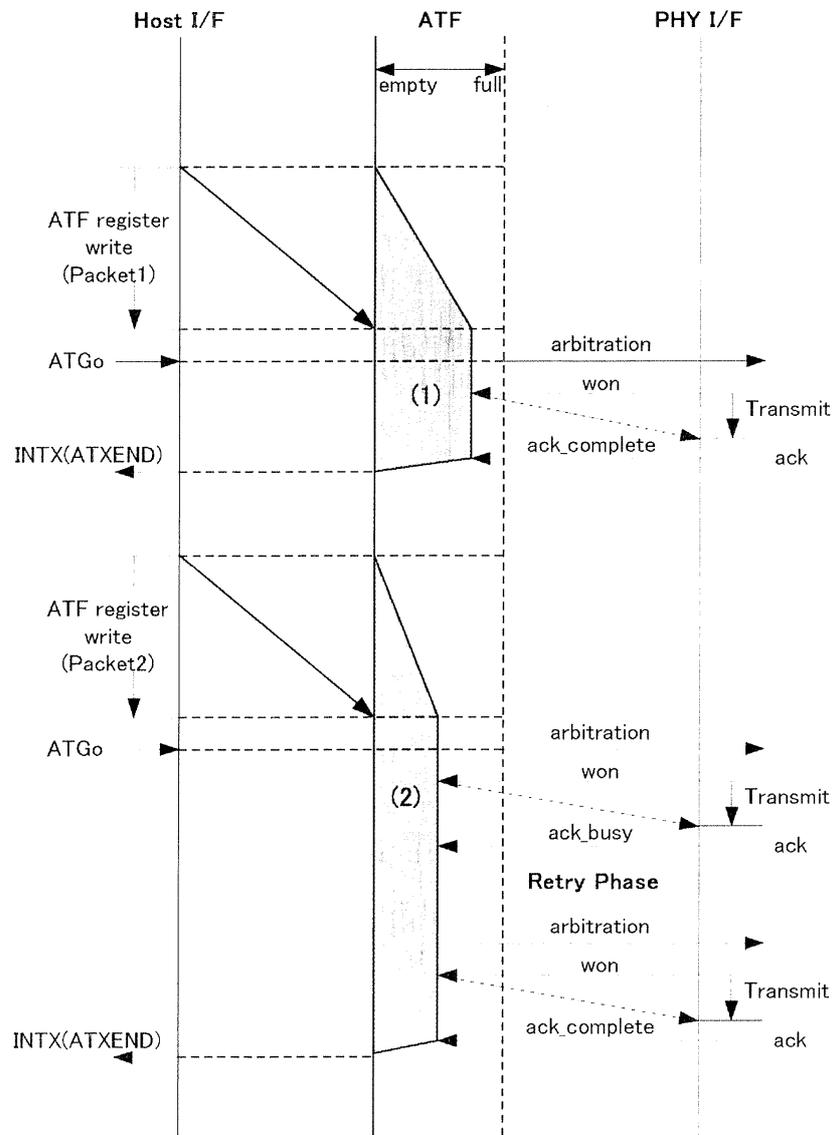
ARF/IRF動作概念図

#### 20.5.4 制御フロー

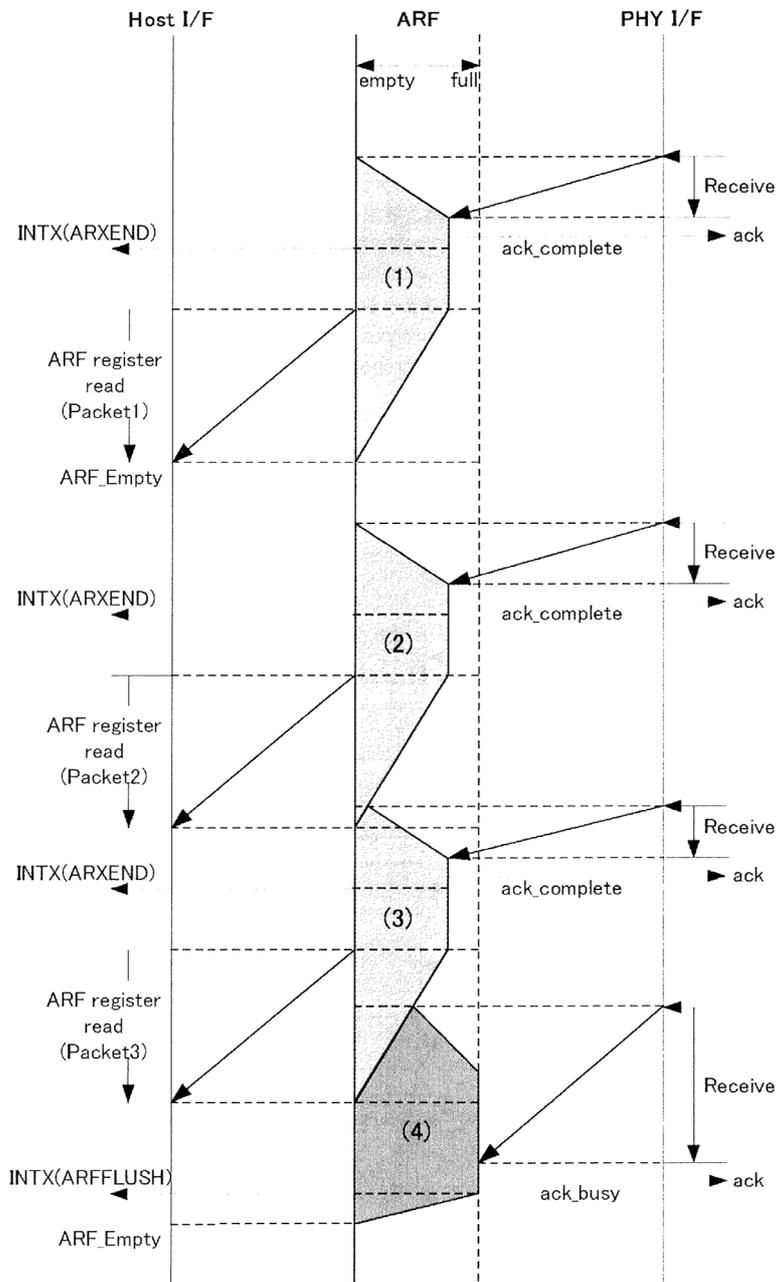
電源投入時の LLC の設定は以下のようになります。

- (a) H\_RSTX\_I をアサートし、リセットを行います。
- (b) バス初期化時に必要な初期設定を行います。例えば、SelfID パケットを受信するために Packet\_Control\_Register の PKTCTL\_SID レジスタビットの設定と、Control\_Register の CTL\_REN レジスタビットの設定など。
- (c) PHY\_Control\_Register の PHY\_PC ビットや PHY\_ISOX ビットを適当な値に設定し、PHY\_PD ビットを '0' に設定し、PHY\_RSTX ビットと PHY\_LPS ビットを '1' に設定して、IEEE1394 バスの初期化を行います。
- (d) バス初期化終了後、PHY との通信によって、ルートノードになったかどうかの情報及び NodeID を取得して、LLC の対応レジスタを適当な値に設定します。
- (e) Control\_Register の CTL\_REN レジスタビットと CTL\_TEN レジスタビットをイネーブルして、下記のフローでパケットの送信と受信を行うことができます。

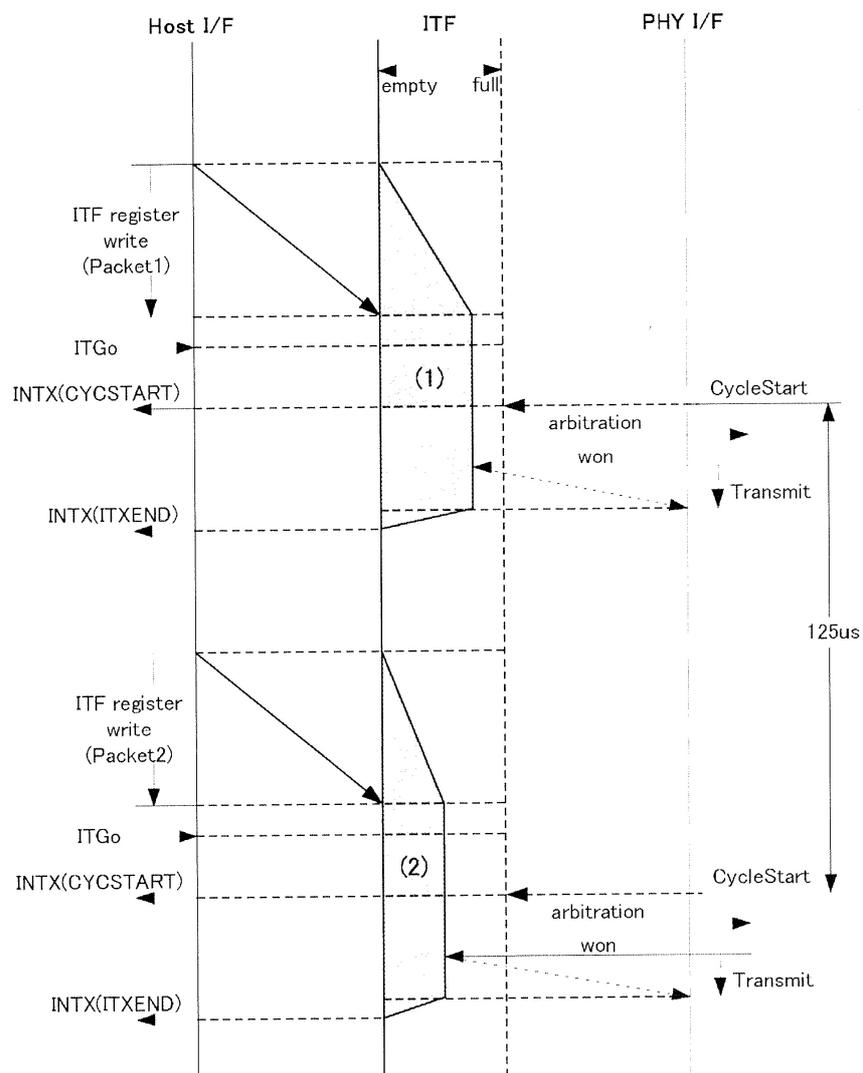
Asynchronous 送信



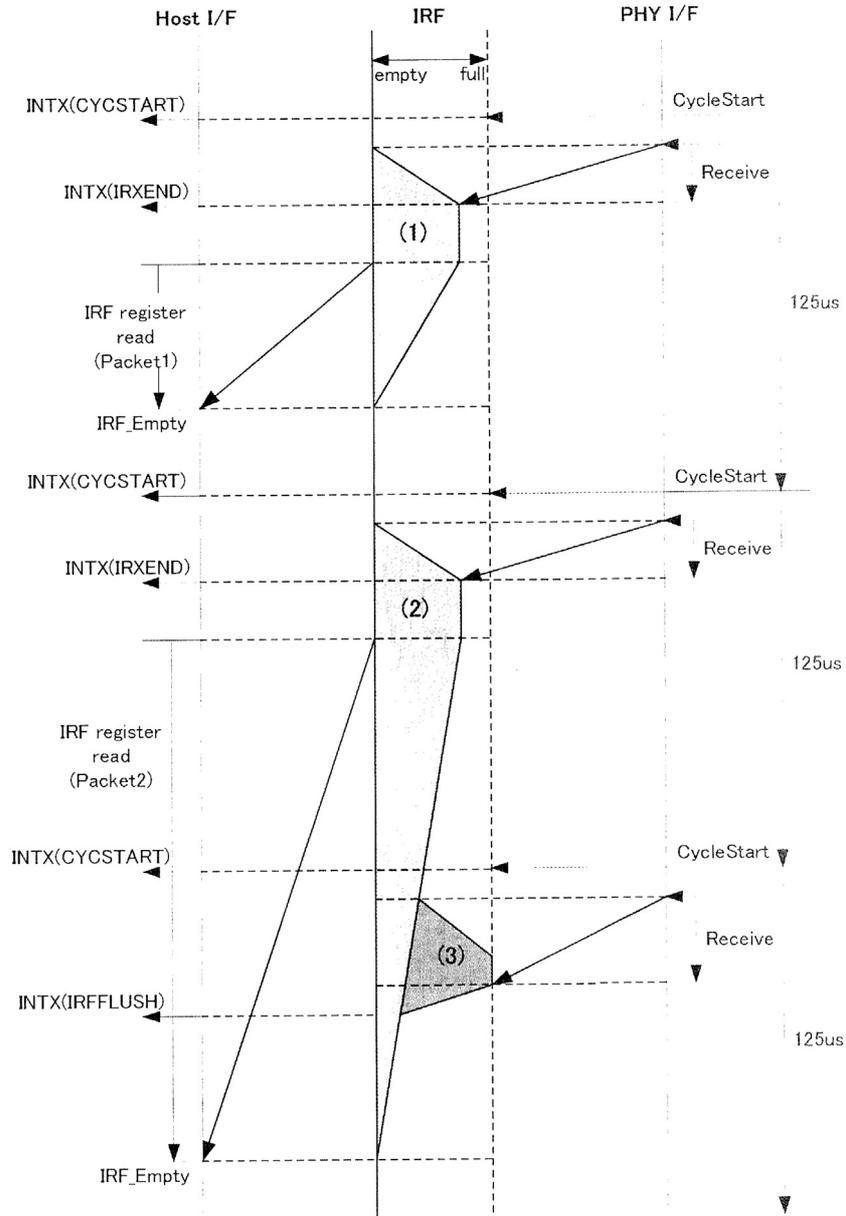
Asynchronous 受信



## Isochronous 送信



Isynchronous 受信



### 20.5.5 Asynchronous パケット送信時の BusNumber

LLC では Asynchronous パケット送信時、IEEE1394 パケットフォーマットのヘッダー内、ソース領域に自ノードのバスナンバーを組み込む機能を持っています。組み込まれる値は、LLC の Node\_Identification\_Register の NID\_BUSID ビットで設定された値、もしくは”3FFh” のどちらか選択された値となります。

Node\_Identification\_Register の NID\_BUSID ビット設定値を有効にするには、送信バッファ内パケットフォーマットの 1Quadlet 目の BusID ビットを’0’ に設定する必要があります。バスナンバーを”3FFh” の固定値とするには、BusID ビットを’1’ に設定する必要があります。

BusID ビットを’1’ に設定した場合、Node\_Identification\_Register の NID\_BUSID ビット設定値は無効となります。



# 21

## Universal Asynchronous Receiver/Transmitter

初期アドレス: Channel0:0xffff6000 Channel1:0xffff6080

### 21.1 アドレスマップ

offset	31	24 23	16 15	8 7	0
0x0000					RB
0x0000					THR
0x0000					DL1
0x0004					IER
0x0004					DL2
0x0008					IIR
0x0008					FCR
0x000c					LCR
0x00010					MCR
0x0014					LSR
0x0018					MSR

#### 21.1.1 Receiver Buffer (RB) / Transmitter Holding Register (THR)

オフセット: 0x0000

7	0

bit 名	機能
7-0	送信 FIFO の入力および受信 FIFO の出力 .



bit 名	機能																														
0	When this is ' 0 ', an interrupt is pending. When this is ' 1 ', no interrupt is pending.																														
3-1	<p>The following table displays the list of possible interrupts along with the bits they enable, priority, and their source and reset control.</p> <table border="1"> <thead> <tr> <th></th> <th>Prio- rity</th> <th>Interrupt Type</th> <th>Interrupt Source</th> <th>Interrupt Reset Control</th> </tr> </thead> <tbody> <tr> <td>011</td> <td>1th</td> <td>Receiver Line Status</td> <td>Parity, Overrun or Framing errors or Break Interrupt</td> <td>Reading the Line Status Register</td> </tr> <tr> <td>010</td> <td>2nd</td> <td>Receiver Data available</td> <td>FIFO trigger level reached</td> <td>FIFO drops below trigger level</td> </tr> <tr> <td>110</td> <td>2nd</td> <td>Timeout Indication</td> <td>There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 char times.</td> <td>Reading from the FIFO (Receiver Buffer Register)</td> </tr> <tr> <td>001</td> <td>3rd</td> <td>Transmitter Holding Register empty</td> <td>Transmitter Holding Register Empty</td> <td>Writing to the Transmitter Holding Register or reading the IIR</td> </tr> <tr> <td>000</td> <td>4th</td> <td>Modem Status</td> <td>CTS, DSR, RI or DCD</td> <td>Reading the Modem Status Register</td> </tr> </tbody> </table>		Prio- rity	Interrupt Type	Interrupt Source	Interrupt Reset Control	011	1th	Receiver Line Status	Parity, Overrun or Framing errors or Break Interrupt	Reading the Line Status Register	010	2nd	Receiver Data available	FIFO trigger level reached	FIFO drops below trigger level	110	2nd	Timeout Indication	There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 char times.	Reading from the FIFO (Receiver Buffer Register)	001	3rd	Transmitter Holding Register empty	Transmitter Holding Register Empty	Writing to the Transmitter Holding Register or reading the IIR	000	4th	Modem Status	CTS, DSR, RI or DCD	Reading the Modem Status Register
	Prio- rity	Interrupt Type	Interrupt Source	Interrupt Reset Control																											
011	1th	Receiver Line Status	Parity, Overrun or Framing errors or Break Interrupt	Reading the Line Status Register																											
010	2nd	Receiver Data available	FIFO trigger level reached	FIFO drops below trigger level																											
110	2nd	Timeout Indication	There's at least 1 character in the FIFO but no character has been input to the FIFO or read from it for the last 4 char times.	Reading from the FIFO (Receiver Buffer Register)																											
001	3rd	Transmitter Holding Register empty	Transmitter Holding Register Empty	Writing to the Transmitter Holding Register or reading the IIR																											
000	4th	Modem Status	CTS, DSR, RI or DCD	Reading the Modem Status Register																											
5-4	Reserved. Should be logic ' 0 '.																														
7-6	Reserved. Should be logic ' 1 ' for compatibility reason.																														

#### 21.1.4 FIFO Control Register (FCR)

オフセット: 0x0008

7	6	5	3	2	1	0





bit 名	機能
0	Data Terminal Ready (DTR) signal control. ' 0 ' - DTR is ' 1 ' ' 1 ' - DTR is ' 0 '
1	Request To Send (RTS) signal control ' 0 ' - RTS is ' 1 ' ' 1 ' - RTS is ' 0 '
2	Out1. In loopback mode, connected Ring Indicator (RI) signal input.
3	Out2. In loopback mode, connected to Data Carrier Detect (DCD) input.
4	Loopback mode. ' 0 ' - normal operation. ' 1 ' - loopback mode. When in loopback mode, the Serial Output Signal (STX_PAD_O) is set to logic ' 1 ' . The signal of the transmitter shift register is internally connected to the input of the receiver shift register. The following connections are made: DTR DSR RTS CTS Out1 RI Out2 DCD
7-5	Ignored.

### 21.1.7 Line Status Register (LSR)

オフセット: 0x0014

7	6	5	4	3	2	1	0

bit 名	機能
0	Data Ready (DR) indicator. ' 0 ' - No characters in the FIFO. ' 1 ' - At least one character has been received and is in the FIFO.
1	Overrun Error (OE) INDICATOR. ' 1 ' - If the FIFO is full and another character has been received in the receiver shift register. If another character is starting to arrive, it will overwrite the data in the shift register but the FIFO will remain intact. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ' 0 ' - No overrun state.
2	Parity Error (PE) indicator. ' 1 ' - The character that is currently at the top of the FIFO has been received with parity error. The bit is cleared upon reading from the register. Generate Receiver Line Status interrupt. ' 0 ' - No parity error in the current character.
3	Framing Error (FE) indicator. ' 1 ' - The received character at the top of the FIFO did not have a valid stop bit. The UART core tries re-synchronizing by assuming that the bit received was a start bit. Of course, generally, it might be that all the following data is corrupt. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ' 0 ' - No framing error in the current character.
4	Break Interrupt (BI) indicator. ' 1 ' - A break condition has been reached in the current character. The break occurs when the line is held in logic 0 for a time of one character (start bit + data + parity + stop bit). In that case, one zero character enters the FIFO and the UART waits for a valid start bit to receive next character. The bit is cleared upon reading from the register. Generates Receiver Line Status interrupt. ' 0 ' - No break condition in the current character.
5	Transmit FIFO is empty. ' 1 ' - The transmitter FIFO is empty. Generates Transmitter Holding Register Empty interrupt. The bit is cleared in the following cases: The LSR has been read, the IIR has been read or data has been written to the transmitter FIFO. ' 0 ' - Otherwise.
6	Transmitter Empty indicator. ' 1 ' - Both the transmitter FIFO and transmitter shift register are empty. The bit is cleared upon reading from the register or upon writing data to the transmit FIFO. ' 0 ' - Otherwise.
7	' 1 ' - At least one parity error, framing error or break indications have been received and are inside the FIFO. The bit is cleared upon reading from the register. ' 0 ' - Otherwise.

### 21.1.8 Modem Status Register (MSR)

オフセット: 0x0018



bit 名	機能
0	Delta Clear To Send (DCTS) indicator. ' 1 ' - The CTS line has changed its state.
1	Delta Data Set Ready (DDSR) indicator. ' 1 ' - The DSR line has changed its state.
2	Trailing Edge of Ring Indicator (TERI) detector. The RI line has changed its state from low to high state.
3	Delta Data Carrier Detect (DDCD) indicator. ' 1 ' - The DCD line has changed its state.
4	Complement of the CTS input or equals to RTS in loopback mode.
5	Complement of the DSR input or equals to DTR in loopback mode.
6	Complement of the RI input or equals to Out1 in loopback mode.
7	Complement of the DCD input or equals to Out2 in loopback mode.

### 21.1.9 Divisor Latches (DL)

オフセット: 0x0000(DL1), 0x0004(DL2)

The divisor latches can be accessed by setting the 7th bit of LCR to ' 1 '. You should restore this bit to ' 0 ' after setting the divisor latches in order to restore access to the other registers that occupy the same addresses.



bit 名	機能
DL1, DL2	The 2 bytes form one 16-bit register, which is internally accessed as a single number. You should therefore set all 2 bytes of the register to ensure normal operation. The register is set to the default value of 0 on reset, which disables all serial I/O operations in order to ensure explicit setup of the register in the software. The value set should be equal to (system clock speed) / (16 times desired baud rate). The internal counter starts to work when the LSB of DL is written, so when setting the divisor, write the MSB first and the LSB last.

## 21.2 動作/使用方法

This UART core is very similar in operation to the standard 16550 UART chip with the main exception being that only the FIFO mode is supported. The scratch register is removed, as it serves no purpose.

### 21.2.1 Initialization

Upon reset the core performs the following tasks:

- The receiver and transmitter FIFOs are cleared.
- The receiver and transmitter shift registers are cleared.
- The Divisor Latch register is set to 0.
- The Line Control Register is set to communication of 8 bits of data, no parity, 1 stop bit.
- All interrupts are disabled in the Interrupt Enable Register.

For proper operation, perform the following:

- Set the Line Control Register to the desired line control parameters. Set bit 7 to ' 1 ' to allow access to the Divisor Latches.
- Set the Divisor Latches, MSB first, LSB next.
- Set bit 7 of LCR to 0 to disable access to Divisor Latches. At this time the transmission engine starts working and data can be sent and received.
- Set the FIFO trigger level. Generally, higher trigger level values produce less interrupt to the system, so setting it to 14 bytes is recommended if the system responds fast enough.
- Enable desired interrupts by setting appropriate bits in the Interrupt Enable register.

Remember that  $(\text{Input Clock Speed})/(\text{Divisor Latch value}) = 16 \times \text{the communication baud rate}$ . Since the protocol is asynchronous and the sampling of the bits is performed in the perceived middle of the bit time, it is highly immune to small differences in the clocks of the sending and receiving sides, yet no such assumption should be made calculating the Divisor Latch values.



# 22

## DDR SDRAM I/F

- 主記憶
  - 32/128 bit I/F のいずれかを選択可能
- Link SDRAM
  - 32 bit I/F
- 2/2.5/3 の CAS Latency に対応
- tWTR(Internal Write to Read Command Delay) が 1 の DDR チップにのみ対応
- 設定レジスタはワードアクセスのみ有効

### 22.1 レジスタマップ

DDR SDRAM I/F	初期アドレス
主記憶 I/F	FFFFFF00
Link SDRAM	FFFFE000

offset	31	24	23	16	15	8	7	0	
0x0	-		-		-		State	S	
0x4	-		CS	-	RAS	-	CAS		
0x8	-		-		-		EMRS		
0xC	-	MRS2		-	-	MRS1			
0x10	-								
0x14	-	RFC	-	RP	-	RCD	-	MRD	
0x18	-	RASmax		-	RASmin				
0x1C	-		REFRESH						
0x20	-							W	

### 22.1.1 主記憶 I/F 幅設定レジスタ

オフセット: 0x20

31	-	1 0	W
----	---	-----	---

bit 名	機能
W	Width :Default 0 本ビットで主記憶 I/F のビット幅を設定する。本レジスタは主記憶 I/F でのみ有効である。0: 32 bit 1: 128 bit

### 22.1.2 I/F 起動レジスタ

オフセット: 0x0

31	-	8 7	State	1 0	S
----	---	-----	-------	-----	---

bit 名	機能
State	State (Read Only) 本ビットは I/F の内部状態を示す。
S	Start :Default 1 本ビットで I/F の起動/停止を設定する。0: I/F 起動 1: I/F 停止

### 22.1.3 メモリモジュール設定レジスタ

オフセット: 0x4

31	-	18 17 16 15	CS	-	12 11	RAS	8 7	-	4 3	CAS	0
----	---	-------------	----	---	-------	-----	-----	---	-----	-----	---

bit 名	機能
State	CS: Default 2(主記憶 128 bit I/F) 2(主記憶 32 bit I/F) 1(Link SDRAM I/F) 本ビットは各 I/F の CS 出力信号の接続本数を設定する。
RAS	Row Address Width: Default 12(主記憶 128 bit I/F) 13(主記憶 32 bit I/F) 13(Link SDRAM I/F) 本ビットは各 I/F に接続されている DDR チップの Row Address 幅を設定する。
CAS	Column Address Width: Default 10(主記憶 128 bit I/F) 9(主記憶 32 bit I/F) 9(Link SDRAM I/F) 本ビットは各 I/F に接続されている DDR チップの Column Address 幅を設定する。

#### 22.1.4 EMRS 設定レジスタ

オフセット: 0x8

31	-	12 11	0
-		EMRS	

bit 名	機能
EMRS	Extended Mode Register Set: Default 0 本レジスタは I/F の起動時に、DDR チップの Extended Mode Register Set に書き込む値を設定する。

#### 22.1.5 MRS 設定レジスタ

オフセット: 0xC

31	28 27	16 15	12 11	0
-	MRS2	-	MRS1	

bit 名	機能
MRS2	Mode Register Set 2: Default 0x21 本レジスタは I/F の起動時に、DDR チップの Mode Register Set に二度目に書き込む値を設定する。
MRS1	Mode Register Set 1: Default 0x121 本レジスタは I/F の起動時に、DDR チップの Mode Register Set に最初に書き込む値を設定する。

### 22.1.6 DDR 設定レジスタ 1

オフセット: 0x14

31	28 27	24 23	20 19	16 15	12 11	8 7	4 3	0
-	RFC	-	RP	-	RCD	-	MRD	

bit 名	機能
RFC	tRFC: Default 9 本レジスタは、I/F に接続されている DDR チップの tRFC 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
RP	tRP: Default 2 本レジスタは、I/F に接続されている DDR チップの tRP 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
RCD	tRCD: Default 1 本レジスタは、I/F に接続されている DDR チップの tRCD 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
MRD	tMRD: Default 1 本レジスタは、I/F に接続されている DDR チップの tMRD 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。

### 22.1.7 DDR 設定レジスタ 2

オフセット: 0x18

31 30 29	16 15 14 13	0
-	RASmax	RASmin

bit 名	機能
RASmax	tRAS max: Default 0x2328 本レジスタは、I/F に接続されている DDR チップの tRASmax 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。
RASmin	tRAS min: Default 6 本レジスタは、I/F に接続されている DDR チップの tRASmin 値をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。

### 22.1.8 リフレッシュインターバル設定レジスタ

オフセット: 0x18

31	16 15	0
-	REFRESH	

bit 名	機能
REFRESH	REFRESH: Default 0x48a8 本レジスタは、I/F に接続されている DDR チップのリフレッシュ周期をサイクル単位で指定する。サイクルの周期は DDR チップに与えているクロックと同じである。



# 23

## Serial Peripheral Interface Unit

---

### 23.1 Outline

Serial Peripheral Interface Unit はクロック同期式のシリアルインターフェースであり、SPIの規格に準拠した周辺機器を制御する。本ユニットは4つのスレーブに対応するが、M-RMTPではチップセレクト (cs\_) が3本しか外に出ていないため、スレーブ3の設定は意味を持たない。

### 23.2 Interface

#### 23.2.1 Address Format

Serial Peripheral Interface Unit の初期ベースアドレスは 0xffffb000 である。Serial Peripheral Interface Unit の制御レジスタのアドレスは次のようになる。



Field Name	Range	Description
Offset	5:0	設定する項目を指定する。

#### 23.2.2 Control Register

Serial Peripheral Interface Unit の制御を行う場合、以下に示す Offset をアドレスの Offset に指定することにより、該当設定レジスタにアクセスする。

**Slave Control**

offset: 0x00

Slave の設定を行う。

31	Reserved	5 4	SS_	1 0	A
----	----------	-----	-----	-----	---

Field Name	Range	Description
Auto (A)	0	1 に設定すると自動的にデータ転送を繰り返す。0 を設定すると、マニュアルで SPI の転送を行う。
Slave Select_ (SS_)	4:1	Auto bit が 0 の場合は、アクセスする Slave を指定する。複数の bit が 0 の場合、下位の bit が優先される。Auto bit が 1 の場合は、自動でアクセス Slave を指定する。複数の bit が 0 の場合、下位の bit から順番にアクセスを行う。

**FIFO Control**

offset: 0x04

FIFO の設定を行う。

31	Reserved	10 9 8 7	CLR	4 3	DREQ	0	INTR
----	----------	----------	-----	-----	------	---	------

Field Name	Range	Description
Interrupt (INTR)	3:0	1 を設定した場合、要因により割り込みを発生させる。 0: 受信 FIFO にデータが半分たまと割り込みを発生させる。 1: 受信 FIFO がいっぱいになると割り込みを発生させる 2: 送信 FIFO のデータが半分未満になると割り込みを発生させる。 3: 送信 FIFO が空になると割り込みを発生させる。
DMA Request (DREQ)	7:4	1 を設定した場合、要因により DMA Request を発生させる。 4: 受信 FIFO にデータが半分たまと DMA Request を発生させる。 5: 受信 FIFO がいっぱいになると DMA Request を発生させる。 6: 送信 FIFO のデータが半分より少くなると DMA Request を発生させる。 7: 送信 FIFO がからになると DMA Request を発生させる。
Clear (CLR)	9:8	1 にすると FIFO をクリアする。この bit は自動的に 0 になる。 8: 受信 FIFO をクリアする。 9: 送信 FIFO をクリアする。

**FIFO Status**

offset: 0x08 (Read Only)

31	Reserved	6	5	4	3	2	1	0
		TF	TH	TE	RF	RH	RE	

Field Name	Range	Description
Rx Empty (RE)	0	受信 FIFO が空の場合 1 になる。
Rx Half (RH)	1	受信 FIFO に半分以上データがたまっている場合 1 になる。
Rx Full (RF)	2	受信 FIFO がいっぱいの場合 1 になる。
Tx Empty (RE)	3	送信 FIFO が空の場合 1 になる。
Tx Half (RH)	4	送信 FIFO に半分以上データがたまっている場合 1 になる。
Tx Full (RF)	5	送信 FIFO がいっぱいの場合 1 になる。

**FIFO**

offset: 0x0c

31	FIFO	0
----	------	---

Field Name	Range	Description
FIFO	31:0	書き込みの場合は送信 FIFO に値が書き込まれる。Slave Control レジスタの Auto bit が 0 の場合、値を書き込むことにより、データの転送を開始する。読み込みの場合は受信 FIFO から値が読み出される。

**Interrupt**

offset: 0x10

割り込みの要因を示す。1 を書き込むことにより、その bit をクリアする。

31	Reserved	4 3 2 1 0
		TE TH RF RH

Field Name	Range	Description
Rx Half (RH)	0	受信 FIFO に半分以上データがたまっている。
Rx Full (RF)	1	受信 FIFO がいっぱいである。
Tx Half (RH)	2	送信 FIFO のデータが半分未満になった。
Tx Empty (RE)	3	送信 FIFO が空である。

**Interval**

offset: 0x14

31	Interval	0
----	----------	---

Field Name	Range	Description
Interval	31:0	Slave Control レジスタの Auto bit が 1 の場合、Slave に対する一連のアクセスが終わった後の待ち時間を指定する。

**Mode0**

offset: 0x20

Slave Select0 用の設定を行なう。

31	30	29	28	24	23	22	21	0
W	R	L	Size	OL	HA	Clock Ratio		

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する。実際には指定した数値 × 2 で SPI の内部クロックを分周し、出力する。0 を指定した場合は $2^{22} \times 2$ 分周される。デフォルトは 0。
HA, OL	23:22	SPI の動作モードを指定する。 0x0 同期クロックは正極性。立ち上りでデータを受け取る。 0x1 同期クロックは正極性。立ち下りでデータを受け取る。 0x2 同期クロックは負極性。立ち上りでデータを受け取る。 0x3 同期クロックは負極性。立ち下りでデータを受け取る。
Size	28:24	データの転送サイズ。指定した値 + 1 bit を転送する。
LSB (L)	29	1 を指定すると LSB から転送を開始する。0 の場合は MSB から転送する。
Read Enable (R <sub>-</sub> )	30	0 を指定すると外部からの入力を読み込み、受信 FIFO に値を格納する。1 の場合は外部からのデータを読み込まない。
Write Enable (W <sub>-</sub> )	31	0 を指定すると送信 FIFO のデータを外部に転送する。1 の場合はデータを送信しない。

**Mode1**

offset: 0x24

Slave Select1 用の設定を行なう。

31	30	29	28	24	23	22	21	0
W	R	L	Size	OL	HA	Clock Ratio		

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する。実際には指定した数値 × 2 で SPI の内部クロックを分周し、出力する。0 を指定した場合は $2^{22} \times 2$ 分周される。デフォルトは 0。
HA, OL	23:22	SPI の動作モードを指定する。 0x0 同期クロックは正極性。立ち上りでデータを受け取る。 0x1 同期クロックは正極性。立ち下りでデータを受け取る。 0x2 同期クロックは負極性。立ち上りでデータを受け取る。 0x3 同期クロックは負極性。立ち下りでデータを受け取る。
Size	28:24	データの転送サイズ。指定した値 + 1 bit を転送する。
LSB (L)	29	1 を指定すると LSB から転送を開始する。0 の場合は MSB から転送する。
Read Enable (R <sub>-</sub> )	30	0 を指定すると外部からの入力を読み込み、受信 FIFO に値を格納する。1 の場合は外部からのデータを読み込まない。
Write Enable (W <sub>-</sub> )	31	0 を指定すると送信 FIFO のデータを外部に転送する。1 の場合はデータを送信しない。

**Mode2**

offset: 0x28

Slave Select2 用の設定を行なう。

31	30	29	28	24	23	22	21	0
W	R	L	Size	OL	HA	Clock Ratio		

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する。実際には指定した数値 × 2 で SPI の内部クロックを分周し、出力する。0 を指定した場合は $2^{22} \times 2$ 分周される。デフォルトは 0。
HA, OL	23:22	SPI の動作モードを指定する。 0x0 同期クロックは正極性。立ち上りでデータを受け取る。 0x1 同期クロックは正極性。立ち下りでデータを受け取る。 0x2 同期クロックは負極性。立ち上りでデータを受け取る。 0x3 同期クロックは負極性。立ち下りでデータを受け取る。
Size	28:24	データの転送サイズ。指定した値 + 1 bit を転送する。
LSB (L)	29	1 を指定すると LSB から転送を開始する。0 の場合は MSB から転送する。
Read Enable (R <sub>-</sub> )	30	0 を指定すると外部からの入力を読み込み、受信 FIFO に値を格納する。1 の場合は外部からのデータを読み込まない。
Write Enable (W <sub>-</sub> )	31	0 を指定すると送信 FIFO のデータを外部に転送する。1 の場合はデータを送信しない。

**Mode3**

offset: 0x2c

Slave Select3 用の設定を行なう。

31	30	29	28	24	23	22	21	0
W	R	L	Size	OL	HA	Clock Ratio		

Field Name	Range	Description
Clock Ratio	21:0	同期クロックで出力するクロックの分周率を指定する。実際には指定した数値 × 2 で SPI の内部クロックを分周し、出力する。0 を指定した場合は $2^{22} \times 2$ 分周される。デフォルトは 0。
HA, OL	23:22	SPI の動作モードを指定する。 0x0 同期クロックは正極性。立ち上りでデータを受け取る。 0x1 同期クロックは正極性。立ち下りでデータを受け取る。 0x2 同期クロックは負極性。立ち上りでデータを受け取る。 0x3 同期クロックは負極性。立ち下りでデータを受け取る。
Size	28:24	データの転送サイズ。指定した値 + 1 bit を転送する。
LSB (L)	29	1 を指定すると LSB から転送を開始する。0 の場合は MSB から転送する。
Read Enable (R <sub>-</sub> )	30	0 を指定すると外部からの入力を読み込み、受信 FIFO に値を格納する。1 の場合は外部からのデータを読み込まない。
Write Enable (W <sub>-</sub> )	31	0 を指定すると送信 FIFO のデータを外部に転送する。1 の場合はデータを送信しない。

**Configuration**

offset: 0x30 (Read Only)

31	Reserved		2	1	0
			FS		

Field Name	Range	Description
FIFO Size (FS)	1:0	FIFO のサイズを示す。 0x1 : 8 Entry 0x2 : 16 Entry 0x3 : 32 Entry

## 23.3 Operation

本 SPI Unit は 4 本の Slave Select を持ち、各 Slave に対して個別の設定を行うことができる。設定は各 Slave の Mode レジスタで行う。

本 SPI Unit は Slave へのアクセスの方法として、自動で継続的に Slave から値を読み込むモードと、1 つ 1 つ送受信を行うモードがある。

### 23.3.1 Manual Mode

1 つ 1 つ送受信を行う場合、Slave Control レジスタの Auto bit を 0 にし、アクセスする Slave を Slave Control レジスタの Slave Select<sub>n</sub> bit で指定する。

値を送信したい場合は、各 Slave の Mode レジスタの W<sub>n</sub> bit を 0 にする。送信したい値を FIFO に書き込むことにより、インターフェースから値が送信される。

値を受信したい場合は、各 Slave の Mode レジスタの R<sub>n</sub> bit を 0 にする。FIFO に値を書き込むことによりインターフェースが動作し、Slave から値を受信する。受信した値は受信 FIFO に書き込まれる。

SPI は送受信を同時に行うことができる。各 Slave の Mode レジスタの W<sub>n</sub> bit と R<sub>n</sub> bit を両方 0 にすることにより、送受信を同時に行う。

### 23.3.2 Auto Mode

Auto Mode は Slave から自動で継続的に値を読み出す場合に使用する。Slave Control レジスタの Auto bit を 1 にすることにより Auto Mode として動作する。値を読み出す Slave は Slave Control レジスタの Slave Select<sub>n</sub> bit で指定する。

各 Slave に対するアクセスは各 Slave の Mode レジスタの設定による。そのため、Mode レジスタの R<sub>n</sub> bit を 0 にし、W<sub>n</sub> bit を 1 にする必要がある。

Auto Mode では、SPI Unit は Slave Control レジスタの Slave Select<sub>n</sub> bit が 0 になっている Slave に対して下位側 (0 番) から順番にアクセスを行う。指定された全ての Slave に対してアクセスを行った後、Interval レジスタで指定されているサイクル数だけ待った後、指定された Slave に対してアクセスを開始する。



# 24

## Parallel I/O Unit

---

### 24.1 Outline

Parallel I/O Unit は 16 bit の入出力を提供する。ただし、M-RMTP では 16 bit 中 15bit が入出力ピンとして外部に接続されているため、bit 15 は意味をもたない。

### 24.2 Interface

#### 24.2.1 Address Format

Parallel I/O Unit の初期ベースアドレスは 0xffffc000 である。Parallel I/O Unit の制御レジスタのアドレスは次のようになる。



Field Name	Range	Description
Offset	4:0	設定する項目を指定する。

#### 24.2.2 Control Register

Parallel I/O Unit の制御を行う場合、以下に示す Offset をアドレスの Offset に指定することにより、該当設定レジスタにアクセスする。

**Data**

offset: 0x00

31	16 15	0
Reserved	Data	

Field Name	Range	Description
Data	15:0	bit が入力の場合は、読むことにより外部からの入力を得る。bit が出力の場合は、書き込むことにより外部に出力を与える。

**Direction**

offset: 0x04

31	16 15	0
Reserved	Direction	

Field Name	Range	Description
Direction	15:0	0 の場合は入力。1 の場合は出力となる。

**Interrupt Enable**

offset: 0x08

31	16 15	0
Reserved	Interrupt Enable	

Field Name	Range	Description
Interrupt Enable	15:0	1 の場合、Data レジスタの対応する値が変化した時に割り込みを発生させる。割り込み発生条件は Interrupt Upedge レジスタ、Interrupt Downedge レジスタで設定する。

**Interrupt Sense**

offset: 0x0c

31	16 15	0
Reserved	Interrupt Sense	

Field Name	Range	Description
Interrupt Sense	15:0	割り込みの発生要因になった bit に 1 がセットされる。このレジスタに 1 を書き込むと、対応する bit がクリアされる。

**Interrupt Upedge**

offset: 0x10

31	16 15	0
Reserved	Interrupt Upedge	

Field Name	Range	Description
Interrupt Upedge	15:0	1 の場合、データが 0 から 1 に変化した時に割り込みを発生させる。

**Interrupt Downedge**

offset: 0x14

31	16 15	0
Reserved	Interrupt Downedge	

Field Name	Range	Description
Interrupt Downedge	15:0	1 の場合、データが 1 から 0 に変化した時に割り込みを発生させる。

**Configuration**

offset: 0x18 (Read Only)

31	2 1 0
Reserved	BW

Field Name	Range	Description
Bit Width (BW)	1:0	Parallel I/O の Bit Width を示す。 0x1 : 8 Bit   0x2 : 16 Bit   0x3 : 32 Bit

**24.3 Operation**

Parallel I/O は 16 bit の幅を持ち、bit ごとに入出力の方向を設定することができる。設定は Direction レジスタで行う。

入力の場合、Parallel I/O に入力されるクロックにより、I/O ピンのデータが Data レジスタにラッチされる。出力の場合、Data レジスタの値が I/O ピンに出力される。

各 bit は指定した条件により割り込みを発生させることができる。割り込みを発生させるためには、Interrupt Enable レジスタの対応する bit を 1 にセットする。また、割り込み発生条件により、Interrupt Upedge レジスタ、Interrupt Downedge レジスタの対応する bit を 1 にセットする。両方 1 にセットした場合、値が変化するたびに割り込みが発生する。

# 25

## 更新履歴

---

2006年3月30日

PCI インターフェースの Mailbox のアドレスマップ (ローカル側オフセット) を修正。

2006年3月30日

更新履歴の追加。

2006年4月18日

MDMA の仕様を追加。

2006年6月13日

ベクトル演算器の仕様を追加。

2006年7月11日

ベクトル命令のニーモニックを修正。

2006年7月13日

ベクトルシフト命令のオペランドの位置を修正。

2006年7月19日

ベクトル制御レジスタの説明を追加。

2006年8月25日

概要の I/O 部分を修正。全体のブロック図を修正。

2006年9月11日

例外処理の割り込み信号を修正。

2006 年 12 月 15 日

命令の説明文を加筆・修正。アドレスマップを修正。MMU の制御レジスタに TLB リードレジスタを追加。MMU で発生しない例外を削除。

2007 年 4 月 11 日

IEEE1394 のマニュアルを修正。