アナログ・デバイセズの DSP、プロセッサ、開発ツール用テクニカル・ノート
http://www.analog.com/jp/ee-notes、http://www.analog.com/jp/processorsにはさまざまな情報を掲載しています。

Blackfin® プロセッサと SDRAM 技術

著者: Fabian Plepp

Rev 2 – 2008 年 12 月 11 日

はじめに

アナログ・デバイセズの Blackfin® ファミリー・プロセッサは、SDRAM へインターフェースするための EBIU (External Bus Interface Unit)を提供しています。

この EE ノートでは次の内容を説明します。

- レジスタの設定とその意味
- SDRAM の初期化
- SDRAM のハードウェア・デザイン
- 16MB 未満の SDRAM の使用
- 性能の最適化
- 消費電力の最適化

この EE ノートでは SDR-SDRAM (DDR-SDRAM ではなく) デバイスのみについて説明します。さらに、このドキュメントは ADSP-BF53x、ADSP-BF52x、ADSP-BF51x、ADSP-BF561 の各プロセッサのみに適用されます。ADSP-BF54x プロセッサには適用されません。

このドキュメントは SDRAM 機能の基本的な面をカバーしますが、詳細については「*The ABCs of SDRAM (EE-126)*[1]」をお読みください。

このドキュメントはさまざまなトピックスに分かれています。特定の情報を探す場合はドキュメント全体を読む必要はありません。

目次

1 SDRAMの概要

このセクションでは、 SDRAM (Synchronous Dynamic Random Access Memory) のパラメータと SDRAM 関連システムのセットアップについて説明します。この説明により、EBIU (External Bus Interface Unit) のレジスタ変数を理解する基礎を提供します。さらに、その他の判断を行う基礎も提供します。SDRAM の詳しい説明は行っていないため、詳細については「*The ABCs of SDRAM (EE-126)*[1]」を参照してください。

1.1 SDRAMの基礎

SRAM はトランジスタを使ってバイナリ・データを保持しますが、DRAM はコンデンサとトランジスタを使います。コンデンサは保持媒体そのものとして機能します。すなわち、充電されたときにロジック 1 になり、その他のときにロジック 0 になります。トランジスタは、セルに対するアクセスと電荷のトラップを制御するゲートとして機能します。この技術の欠点は、コンデンサは時間の経過により放電することです。

データの喪失を防止するため、DRAM を周期的にリフレッシュして、メモリ・セル上の電荷を回復させる必要があります。このため、読み出し動作と書き込み動作をメモリ・アレイ内の全メモリ・ロケーションに対してリフレッシュ・レート周期(一般に msec で規定)に少なくとも 1 回実行する必要があります。

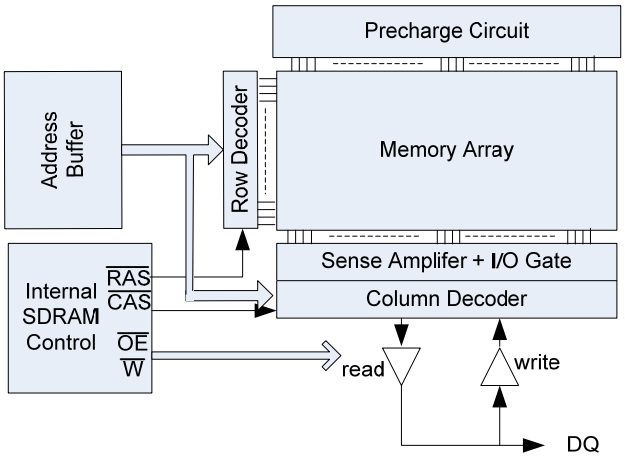


図1.SDRAM

DRAM セルは、行と列のアレイ内に構成されています。各セルは、行と列アドレスを使ってアクセスすることができます。行はページと呼ばれることもあり、列数はページ・サイズと呼ばれます。アドレスは時分割多重化され、行アドレスの後ろに列アドレスが続いて送信されます。

/RAS (行アクセス・ストローブ)信号と/CAS (列アクセス・ストローブ) 信号が、行アドレスと列アドレスの時分割多重化を制御しています。/RAS 信号は、行アドレスがアドレス・バッファに存在し、内部行アドレス・デコーダによりデコードされたことを表示します。短い遅延の後、/CAS 信号はアドレス・バッファ上にある列アドレスが列アドレス・デコーダに渡されたことを表示します。

/WE がハイ・レベルの場合、読み出しコマンドが処理されます。行によりアドレス指定されたセルが読み出しされ、増幅されて、セルへ再び書き込まれます。アドレス指定された列は、データ・バスを使って送信されます。

/WE がロー・レベルの場合、書き込みコマンドが処理されます。1 つのセルに対する単独の書き込みが不可能なため、行全体がバッファに読み出されます。バッファでは、変更対象のエレメントが書き込まれ、行全体が再びアレイに書き込まれます。

1.2 Blackfin レジスタ内のSDRAM パラメータ

1.2.1 EBCAW (SDRAM 外部バンク列アドレス幅)

外部バンク列アドレス幅は SDRAM のページ・サイズと呼ばれることもあります。Blackfin プロセッサは、512 バイト (8 ビット)、1 K バイト (9 ビット)、2 K バイト (10 ビット)、4 K バイト (11 ビット)のページ・サイズをサポートしています。

Bank	Row Address	Column Address	Byte
2 MSB	[(EBCAW+[10,16]):(EBCAW+1)]	[EBCAW:1]	0

下位ビット (LSB)であるバイト・アドレス・ビットは、同時に論理アドレスの下位ビットでもあります。列アドレスの幅に応じて、行アドレスとバンクは論理アドレス内で異なるビット位置を持ちます。例えば、列アドレシング幅 (CAW) が 11 ビットの場合、行アドレスの LSB は 32 ビット論理アドレス[31:0]のビット 12 になります。SDRAM にアクセスしようとする場合、これは重要パラメータになります。行アドレス数は、SDRAM のサイズに依存します。データ・シートで列アドレシング幅を調べるときは、列アドレシングに使用されるアドレス・ピン数を注目してください。詳細については、Blackfin プロセッサの「*ハードウェア・リファレンス*」の EBIU の章を参照してください。

1.2.2 EBSZ (SDRAM 外部バンク・サイズ)

SDRAM 外部バンク・サイズは次式から求めることができます。

$$\text{MemorySize} = \frac{\text{Bank size x Number of Data pins x Number of Banks}}{8}$$

例えば、MT48LC32M16A2 は 8 M バイト×16 ピン×4 バンクを持ち、これは 536,870,912 ビット = 64 M バイトになります。

16 MB未満を使う場合、16 MB未満のSDRAMでのBlackfin プロセッサの使用を参照してください。

1.2.3 SDRAM のタイミング

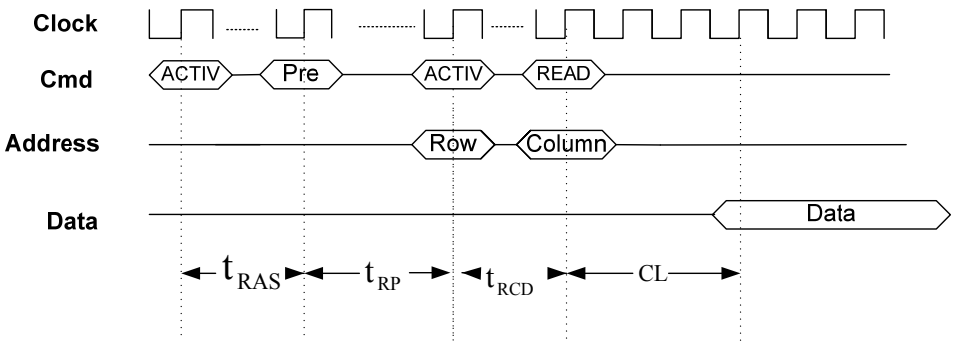


図2. SDRAM のタイミング

SDRAM CAS 遅延 (CL)、SDRAM バンク・アクティベート・コマンド遅延 (t_{RAS})、SDRAM バンク・プリチャージ遅延 (t_{RP})、RAS—CAS 間遅延 (t_{RCD})、書き込み—プリチャージ間遅延 (t_{WR})の SDRAM タイミングの詳細は、プロセッサのハードウェア・リファレンスの EBIU の章で説明してあります。

1.3 マルチプロセッサ環境オプション

Blackfin プロセッサは、マルチプロセッサ環境で使用することができます。複数のプロセッサを相互にインターフェースさせる 1 つの方法は、外部メモリを共用させて、この外部メモリを介して相手のプロセッサへメッセージを渡す方法です。調停機能を提供するため、Blackfin プロセッサにはこの用途の専用ピンがあります。

Blackfin プロセッサを共用メモリを使うマルチプロセッサ環境で使うときは、/BR ピン、/BG ピン、/BGH ピンを接続してアクセス制御を行う必要があります。外部デバイスがバスへのアクセスを要求するときは、バス要求信号(/BR ピン)をアサートします。他の待ち状態の要求がない場合は、外部バス要求が許可されます。プロセッサは、データおよびアドレス・バスをスリー・ステートにし、バス許可信号 (/BG ピン)がアサートされます。バスが別の外部デバイスに許可される場合は、外部メモリからのすべてのデータまたは命令のフェッチによりプロセッサが停止され、バスが解放されてアクセスが実行できるようになるまでこの停止が続きます。外部デバイスが /BR を解除すると、プロセッサは /BG のアサートを解除して、停止したポイントから実行を再開します。プロセッサは次の外部ポート・アクセスを開始する準備ができるとプロセッサは/BGH ピンをアサートしますが、バスが既に許可されているときは/BGH ピンのアサートが遅れます。

1.3.1 BGSTAT (バス許可ステータス)

バスが許可されると、SDSTAT レジスタの BGSTAT ビットがセットされます。プロセッサはこのビットを使ってバス・ステータスをチェックすることにより、外部バス許可による遅延を伴うトランザクションの開始を回避することができます。

1.3.2 PUPSD (パワーアップ・スタートアップ遅延)

このオプションでは、パワーアップ起動シーケンスに対してシステム・クロックで 15 サイクルの遅延を設定します。一方のプロセッサが SDRAM を他方のプロセッサへ渡す場合、そのプロセッサは SDRAM をセルフ・リフレッシュ・モードに設定する必要があります。セルフ・リフレッシュ・モードが開始されると、SDRAM は自分の内部クロックを使って、自分の自動リフレッシュ・サイクルを実行します。SDRAM は、t_{RAS} に等しい最小周期の間セルフ・リフレッシュ・モードに留まる必要があります。

セルフ・リフレッシュ・モードを終了する手順では、一連のコマンドが必要です。先ず、CKE がハイ・レベルに戻る前に CLK が安定する必要があります (安定なクロックとは、クロック・ピンに対して規定されたタイミング制約内で繰り返し変化する信号と定義されます)。 CKE がハイ・レベルになると、SDRAM は t_{XSR} 間 NOP コマンドを発行する必要があります。これは、進行中の内部リフレッシュを完了させるためにこの時間が必要なためです。この時間 t_{XSR} を考慮すると、Blackfin プロセッサがパワーアップ・シーケンスを開始させるまでに 15 サイクルの遅延が必要になります。

1.3.3 CDDBG (許可中のコントロール・ディスエーブル)

Blackfin プロセッサ以外の他のデバイスがメモリをアクセスする、共用メモリ使用のマルチプロセッサ環境では、この機能を使うと、プロセッサの外部メモリ・インターフェースが、さらにアドレス・ピン、データ・ピン、メモリ・コントロール・ピン (SRAS、SCAS、SWE、SA10、SCKE)、クロック・ピン (CLKOUT)をスリー・ステートにすることができるようになります。

1.4 モバイル/ ローパワー SDRAM オプション

Blackfin プロセッサは、モバイル SDRAM チップ (ローパワー SDRAM とも呼ばれます)をサポートしています。Blackfin プロセッサ・シリーズに応じて、3.3V、2.5V、1.8V の SDRAM を使用することができます。2.5V と 1.8V のチップを使うときは、プロセッサが SDRAM デバイスのデータ・シートで規定する低電圧 (最大)を出力できることをプロセッサのデータ・シート (電気的仕様)で確認してください。モバイル SDRAM が低消費電力を実現するのは、低電圧だけではなく低電流にもよります。モバイル SDRAM のもう 1 つの利点は、使用しない複数のバンクのセルフ・リフレッシュを停止する機能と、温度を指定してセルフ・リフレッシュ・レートを低下させる機能を持っていることです。これらの機能を使うと、アプリケーションの消費電力を最適化することができます。Blackfin のデータ・シートで別の注記がないかぎり、1.8V VDDEXT では、システム・クロックの最大周波数は 100 MHz に制限されています。

	ADSP-BF51x	ADSP-BF52x	ADSP-BF533/2/1	ADSP-BF537/6/4	ADSP-BF539/8	ADSP-BF561
1.8V	✓	✓	✓	×	×	×
2.5 V	✓	✓	✓	✓/×*	✓/×*	✓/×*
3.3 V	✓	✓	✓	✓	✓	✓

*選択した SDRAM デバイスに依存します。

図3.プロセッサ・シリーズがサポートしている SDRAM 電圧

モバイル SDRAM の拡張レジスタを使用するときは、EMREN ビットをセットする必要があります。

1.4.1 PASR (Partial Array Self Refresh)

各リフレッシュでは電力を消費します。アプリケーションの特別なモードでメモリ全体を使わない場合、この機能を使うと、SDRAM の幾つかのバンクのリフレッシュをディスエーブルすることができます。この機能の利点は、消費電力を小さくすることです。

例えば、データ(例えば画像)を SDRAM に保存し、何らかの信号処理を行い、そのデータを送信するか不揮発性メモリ（フラッシュなど）へ保存し、スリープ・モードに戻るアプリケーションを考えましょう。大部分の SDRAM は、信号処理にのみ必要です。処理の後データは不要となるので、データが置かれているバンクのリフレッシュは不要になります。

1.4.2 TCSR (Temperature-Compensated Self-Refresh)

標準 SDRAM では、セルフ・リフレッシュ・レートはワーストケースに設定されています。すなわち、高い温度ではコンデンサの放電が多くなり、高いリフレッシュ・レートでデータを保持する必要があります。リフレッシュ・レートを上げると消費電力が大きくなります。TCSR ビットを使うと、測定している温度に応じてユーザ・アプリケーションからセルフ・リフレッシュ・レートを設定することができます。

1.5 SDRAM タイミングを調整するオプション

SDRAM のタイミング・パラメータ仕様が EBIU の仕様を満たさないことがあります。このため、PLL コントロール・レジスタでは、出力信号を遅延させ、入力ラッチ・メカニズムを 200 ps シフトさせるオプションを提供しています。

PLL Control Register (PLL_CTL)

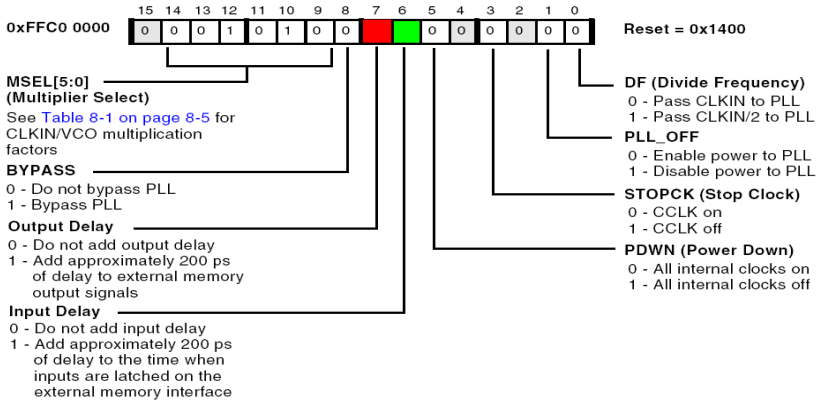


図4.PLL_CTL レジスタ (「ハードウェア・リファレンス」からコピー)

ビット 6 とビット 7 (図 4)がSDRAM 信号への遅延を提供します。書き込みアクセスが仕様を満たさないときに出力遅延を使用してください。データ・ホールドを 200 ps遅延させます。読み出しアクセスの仕様を満たさないときは、入力遅延を使用します。入力データ信号のラッチを 200 ps遅延させます。

t_{sclk} CLKOUT Period

t_{SSDAT} Data Setup Before CLKOUT (BF data sheet)

t_{AC} Access time from CLK (SDRAM data sheet)

t_{DH} Input Hold time (SDRAM data sheet)

t_{HSDAT} Data Hold after CLKOUT (BF data sheet)

t_{OH} Output Hold time (SDRAM data sheet)

t_{OLZ} Delay to the output high impedance from CLK (SDRAM data sheet)

t_{OHZ} Delay from high impedance to signal from CLK (SDRAM data sheet)

1.5.1 Blackfin 出力 / SDRAM 入力の式 (書き込み)

SDRAM 入力の場合、次式が有効です:

$$t_{DH} < t_{HSDAT}$$

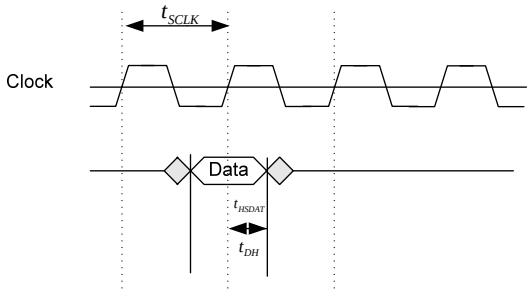


図5.タイミング

1.5.2 Blackfin 入力 / SDRAM出力の式 (読み出し)

これは、SDRAM クロックを 133 MHz に設定して、タイミング・パラメータを仕様限界にする特別なケースに適用されます。

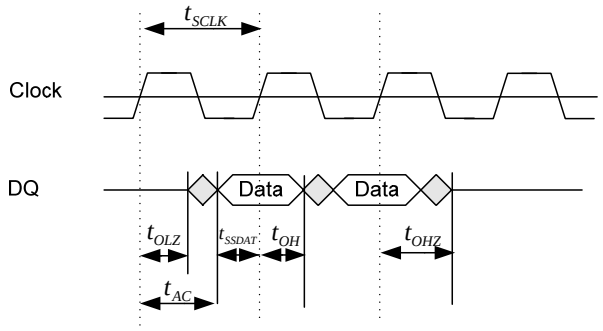


図6.タイミング

読み出し動作が正しく行われたことを確認するときは、 $t_{sclk} > t_{SSDAT} + t_{AC}$ を保証してください。

2 SDRAMの初期化

SDRAM の初期化は、アプリケーションの性能と SDRAM の消費電力に影響を与えます。このため、設定を良く理解することが望まれます。このセクションでは、アプリケーションを動作させるための EBIU (External Bus Interface Unit) レジスタの設定方法について説明します。SDRAM を初期化するときは、次のいずれかの方法を使います。

- エミュレータと VisualDSP++® `.XML` ファイルを使う SDRAM の初期化
- アプリケーション内でのレジスタ設定による SDRAM の初期化
- VisualDSP++ システム・サービス・モデルの使用による SDRAM の初期化
- 実際のアプリケーションをロードする前に初期化ファイルによる SDRAM の初期化
- OTP (One Time Programmable) メモリ内の値による SDRAM の初期化

2.1 エミュレータとVisualDSP++ .XML ファイルを使うSDRAM の初期化

ソフトウェア開発の早期ステージでは、インサーキット・エミュレータ (ICE)を使ってソフトウェアをアップロードします。エミュレータは、プログラムをプロセッサへアップロードする際に EBIU レジスタを自動的に設定することができます。

この機能をイネーブルするときは、VisualDSP++ の`Settings`メニューから、`Target Options`を閉じます。表示される`Target` オプション・ダイアログ・ボックス (図 7)で、`Use XML reset values`オプションを選択します。

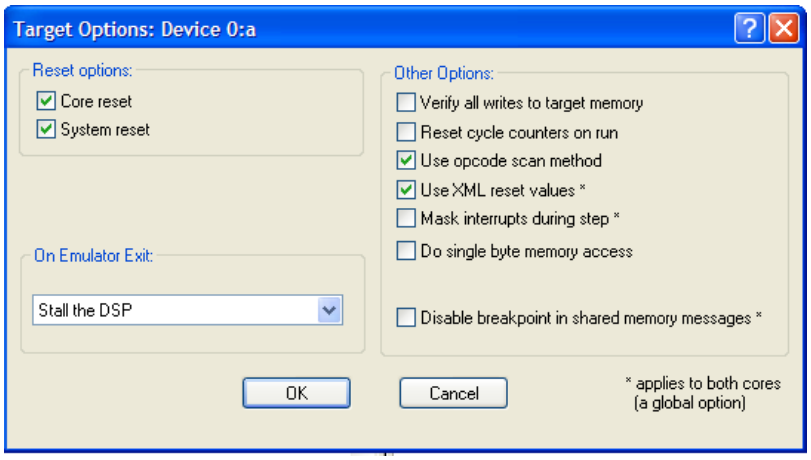


図7.Target Options ダイアログ・ボックス

VisualDSP++ リリース 4.5 以前の場合、値はADSP-BF5XX-proc.xml(XXは、例えばADSP-BF537-proc.xmlやADSP-BF561-proc.xmlなどのアーキテクチャを表します)という名前の.xml (extensible markup language) ファイルから取得されていました。これらのファイルは<*install_path*>\Analog Devices\VisualDSP X.X\System\ArchDefディレクトリにロードされます。これらの値を変更することにより、EBIUを設定することができます。 図 8 に、プロセッサの.XML ファイルの一部を示します。詳細については、VisualDSP++ Helpの“カスタム・ボード・サポート” を参照してください。

```
<!--  ********************************************  -->
<!--  Register resets used by emulator              -->
<!--  ********************************************  -->
- <register-reset-definitions>
    <register name="EBIU_SDRRC" reset-value="0x03A0" core="Common" />
    <register name="EBIU_SDBCTL" reset-value="0x25" core="Common" />
    <register name="EBIU_SDGCTL" reset-value="0x0091998d" core="Common" />
    <register name="EBIU_AMGCTL" reset-value="0xff" core="Common" />
  </register-reset-definitions>
  <!--  ************************************************************  -->
  <!--  ******* Customizations: If you make changes to this file, make a  -->
  <!--  ******* copy of your customized file to facilitate future upgrades.  -->
  <!--  ************************************************************  -->
</visualdsp-proc-xml>
```

図 8: ADSP-BF5xx-proc.xml ファイル (VisualDSP++ 4.5)の一部

プロセッサ .xml ファイルはスタートアップ時にのみ読み出されます。このファイルに対する VisualDSP++ IDDE の動作中の変更は無視されます。VisualDSP++ 開発ツールを起動する前に値を編集しておくように注意してください。VisualDSP++ ツールの動作中は、.XML ファイルの値を使ってアナログ・デバイセズ EZ-KIT Lite® 開発ボードの対応する SDRAM 値が設定されます。

VisualDSP++ リリース 5.0 以降の場合は、ArchDef フォルダ内の.XML ファイルを直接変更することは推奨できません。次の節で説明するカスタム・ボード・サポートを使用してください。

VisualDSP++ リリース 5.0 以降のカスタム・ボード・サポートでは、アプリケーション内でのリセット値の設定は容易になりました。この機能を使うと、*複数セットのリセット設定値*を持つことができ、VisualDSP++ ツールを再起動しないでこれらの間で切り替えることができます。テキスト・エディタを開いて、次のようにコードをファイル内に配置して保存してください。この例では(リスト 1)、ファイル名は`My_custom_board_reset_settings.xml`です。

```
<?xml version="1.0" standalone="yes"?>

<custom-visualdsp-proc-xml
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="\Program Files\Analog Devices\VisualDSP
5.0\System\ArchDef\ADSP-custom-board.xsd"
     processor-family="BLACKFIN"
     file="My custom board reset settings.xml">

<!-- ************************************************************************ -->
<!-- ******* My custom board reset settings.xml                               -->
<!-- ************************************************************************ -->

<custom-register-reset-definitions>

     <register name="EBIU_SDRRC" reset-value="0x03A0" core="Common" />
     <register name="EBIU_SDBCTL" reset-value="0x25" core="Common" />
     <register name="EBIU_SDGCTL" reset-value="0x0091998D" core="Common" />

</custom-register-reset-definitions>
</custom-visualdsp-proc-xml>
```

リスト 1.カスタム・ボード・リセット設定の例 (VisualDSP++ 5.0)

VisualDSP++ 開発ツール内でこの機能をイネーブルするときは、次のステップを実行します。

1. `Settings` メニューから、`Session` を選択します。
2. `Session Settings` ダイアログ・ボックスで、`Enable customizations` を選択します。
3. `Custom board support file name` で、SDRAM リセット値を含む“`My_custom_board_reset_settings.xml`”という名前のファイルを選択します。

プロセッサが VisualDSP++ ツールによりリセットされるごとに、これらのリセット値がレジスタに設定されます。

2.2 メモリ・マップド・レジスタを使う初期化

SDRAM コントローラを初期化する旧来の方法は、メモリ・マップド・レジスタ (MMR)に直接値を割り当てることです。

一例として、モバイル SDRAM のデータ・シート[1]を使います。

[1] Samsung 社の K4M56163 R(B)N/G/L/F モバイル SDRAM

• 64ms refresh period (8K cycle).

Part No.	Max Freq.	Interface	Package
K4M56163LG-R(B)N/G/L/F75	133MHz(CL3), 111MHz(CL2)	LVCMOS	54 FBGA Pb (Pb Free)
K4M56163LG-R(B)N/G/L//F1H	111MHz(CL2)		
K4M56163LG-R(B)N/G/L/F1L	111MHz(CL=3)*1, 83MHz(CL2)		

Parameter	Symbol	Version			Unit	Note
		-75	-1H	-1L		
Row active to row active delay	tRRD(min)	15	18	18	ns	1
$\overline{RAS}$ to $\overline{CAS}$ delay	tRCD(min)	18	18	24	ns	1
Row precharge time	tRP(min)	18	18	24	ns	1
Row active time	tRAS(min)	45	50	60	ns	1
	tRAS(max)	100			us	
Row cycle time	tRC(min)	63	68	84	ns	1,2
Last data in to row precharge	tRDL(min)	2			CLK	3
Last data in to Active delay	tDAL(min)	tRDL + tRP			-	4

図9.例: モバイル SDRAM のデータ・シートの一部

Address configuration

Organization	Bank	Row	Column Address
16Mx16	BA0,BA1	A0 - A12	A0 - A8

図 10: データ・シートのアドレス設定仕様

SDRAM レジスタを設定する前に、PLL を設定する必要があります。この例では、システム・クロック周波数 133 MHz を使用します。先ず、リフレッシュ・レートを次のように計算します。

$$\mathrm{RDIV} = \frac{f_{\mathrm{SCLK}} \cdot t_{\mathrm{REF}}}{\mathrm{NRA}} - (t_{\mathrm{RAS}} + t_{RP})$$

データ・シート (-75)から、次の情報を得ます。

$f_{\mathrm{SCLK}} = 133\ \mathrm{MHz}$

$t_{\mathrm{REF}} = 64\ \mathrm{ms}$

NRA = 8192

$t_{\mathrm{RAS}} = 45\ \mathrm{ns}$ (133 MHz): $t_{\mathrm{RAS}} = 6\ \mathrm{cycles}$

$t_{\mathrm{RP}} = 18\ \mathrm{ns}$ (133 MHz): $t_{\mathrm{RP}} = 3\ \mathrm{cycles}$

$$\mathrm{RDIV} = \frac{133 \cdot 10^6 \cdot 64 \cdot 10^{-3}}{8192} - (6+3) = 1030.0625 \approx 1030 \ \text{ in Hex} : 0\text{x}406$$

さらに、次の値を計算する必要があります。

$t_{\mathrm{RCD}} = 18\ \mathrm{ns}$ (133 MHz): $t_{\mathrm{RCD}} = 3\ \mathrm{cycles}$

Samsung 社のデバイスでは、t_{WR} は t_{RDL} と呼ばれます。

$t_{\mathrm{WR}} = t_{\mathrm{RDL}} = 2\ \mathrm{cycles}$

次に、指定したタイミングから導出したすべてのパラメータが SDRAM 仕様も満たすことを保証する必要があります。したがって、次を制御する必要があります。

$t_{XSR} = t_{RAS} + t_{PR}$ -> 6 サイクル + 3 サイクル = 9 サイクル (133 MHz) : 67.5 ns

$t_{RRD} = t_{RCD} + 1$ -> 3 サイクル + 1 サイクル = 4 サイクル (133 MHz) : 30 ns

$t_{RC} = t_{RAS} + t_{PR}$ -> 6 サイクル + 3 サイクル = 9 サイクル (133 MHz) : 67.5 ns

$t_{RFC} = t_{RAS} + t_{PR}$ -> 6 サイクル + 3 サイクル = 9 サイクル (133 MHz) : 67.5 ns

これらの値と SDRAM のデータ・シートを比較すると、仕様を満たしていることが分かります。

さらに、拡張レジスタをイネーブルして温度とパーシャル・セルフ・リフレッシュを設定する必要があります。PASR は次のように設定することができます。

- **PASR_ALL** – SDRAM の 4 バンクをすべてセルフ・リフレッシュでリフレッシュ
- **PASR_B0_B1** – SDRAM のバンク 0 とバンク 1 をセルフ・リフレッシュでリフレッシュ
- **PASR_B0** – SDRAM のバンク 0 のみをセルフ・リフレッシュでリフレッシュ

リスト 2 に、C言語による初期化を示します。

```
//SDRAM Refresh Rate Setting
    *pEBIU_SDRRC = 0x406;
//SDRAM Memory Bank Control Register
    *pEBIU_SDBCTL =          EBCAW_9    | //Page size 512
                             EBSZ_64    | //64 MB of SDRAM
                             EBE;         //SDRAM enable
//SDRAM Memory Global Control Register
      *pEBIU_SDGCTL    = ~CDDBG      & // Control disable during bus grant off
                         ~FBBRW      & // Fast back to back read to write off
                         ~EBUFE      & // External buffering enabled off
                         ~SRFS       & // Self-refresh setting off
                         ~PSM        & // Powerup sequence mode (PSM) first
                         ~PUPSD      & // Powerup start delay (PUPSD) off
                         TCSR        | // Temperature compensated self-refresh at 85
                         EMREN       | // Extended mode register enabled on
                         PSS         | // Powerup sequence start enable (PSSE) on
                         TWR_2       | // Write to precharge delay TWR = 2 (14-15 ns)
                         TRCD_3      | // RAS to CAS delay TRCD =3 (15-20ns)
                         TRP_3       | // Bank precharge delay TRP = 2 (15-20ns)
                         TRAS_6      | // Bank activate command delay TRAS = 4
                         PASR_B0     | // Partial array self refresh Only SDRAM Bank0
                         CL_3        | // CAS latency
                         SCTLE       ; // SDRAM clock enable
```

リスト 2.レジスタによる初期化

アセンブリ言語のコードは、初期化コード (2 章)に記載してあります。

アプリケーション内の PLL 設定を後で変更するときはどうなるのでしょうか?最適メモリ性能を得るためには、設定を変更する必要があります。メモリ性能に問題ない場合は、ワーストケースの値を計算する必要があります。

ADSP-BF537 EZ-KIT Lite (アセンブラ言語と C 言語) と ADSP-BF561 EZ-KIT Lite (C 言語)での SDRAM 初期化の例は、この EE ノートに記載してあります。

2.3 システム・サービスを使用する初期化

SDRAM を初期化するもう 1 つの方法は、アプリケーション自体の中でセットアップを処理する方法です。EBIU 設定を変更する最も楽な方法は、システム・サービスを使う方法です。EBIU システム・サービスを使う主な利点は、計算なしで EBIU を設定できることです。2 つ目の利点は、EBIU の変更が PLL の変更に従うため、実際のシステム・クロック周波数について心配する必要はありません。

初期化は `adi_ebiu_init` 関数により行われます。ここで、前のセクションと同じ SDRAM を使います。

システム・サービスを使って初期化を行うときは、SDRAM パラメータをシステムへ渡す必要があります。このため、`ADI_EBIU_COMMAND_PAIR`と呼ばれるコマンド構造体を使います(表 1)。

この例の構造体を初期化するため、変数 (-75 用) を リスト 3に示すように設定します。

```
// set the sdram to 64 MB
ADI_EBIU_SDRAM_BANK_VALUE bank_size = {0, ADI_EBIU_SDRAM_BANK_64MB};

//set the sdram to 9 bit Column Address Width (like we see in the Address
//Configuration)
// A0-A8 -> 9bits Column Address
ADI_EBIU_SDRAM_BANK_VALUE bank_caw = {0,
(ADI EBIU SDRAM BANK SIZE)ADI EBIU SDRAM BANK COL 9BIT}; // 9bit CAW

//set the twr to 2 sclk +  no time
ADI EBIU TIMING VALUE MyTWR = {          2,          // 2 cycle
{0, ADI_EBIU_TIMING_UNIT_PICOSEC}};  // 0 ns

//set refresh rate to 64ms at 8192 rows as seen in the data sheet
ADI EBIU TIMING VALUE Refresh = {                    // SDRAM Refresh rate:
                                 8192,               // 8192 cycles
                                {64, ADI_EBIU_TIMING_UNIT_MILLISEC     }}; // 64ms

//set TRAS to 45ns
ADI_EBIU_TIME MyTRAS = {45, ADI_EBIU_TIMING_UNIT_NANOSEC};
//set TRP to 18ns
ADI_EBIU_TIME MyTRP  = {18, ADI_EBIU_TIMING_UNIT_NANOSEC};
//set TRCD to 18ns
ADI_EBIU_TIME MyTRCD = {18, ADI_EBIU_TIMING_UNIT_NANOSEC};
//set CAS threshold frequency
u32 MyCAS = 133;
//Enable Extended Mode Register because we are using Mobile SDRAM
ADI_EBIU_SDRAM_EMREN MyEMREN = ADI_EBIU_SDRAM_EMREN_ENABLE;
//Refresh only the first bank
ADI_EBIU_PASR  MyPASR = ADI_EBIU_PASR_INT0_ONLY;

//Temperature Compensation at 85°C
ADI_EBIU_SDRAM_TCSR  MyTCSR = ADI_EBIU_SDRAM_TCSR_85DEG;
//we don't have any registered buffer
ADI_EBIU_SDRAM_EBUFE MyEBUFE = ADI_EBIU_SDRAM_EBUFE_DISABLE;
//no fast-back-to-back read/write
ADI_EBIU_SDRAM_FBBRW MyFBBRW = ADI_EBIU_SDRAM_FBBRW_DISABLE;
//Do not disable the control during bus grant
ADI_EBIU_SDRAM_CDDBG MyCDDBG = ADI_EBIU_SDRAM_CDDBG_DISABLE;
//We don't need any delay at Power Up
ADI_EBIU_SDRAM_PUPSD MyPUPSD = ADI_EBIU_SDRAM_PUPSD_NODELAY;
//Do first the refresh
ADI_EBIU_SDRAM_PSM MyPSM = ADI_EBIU_SDRAM_PSM_REFRESH_FIRST;
```

リスト 3. EBIU 構造体の初期化

Description	Command	Value Type
Bank Size	ADI_EBIU_CMD_SET_SDRAM_SIZE	ADI_EBIU_SDRAM_BANK_VALUE
Bank col. address width	ADI_EBIU_CMD_SET_SDRAM_BANK_COLUMN_WIDTH	ADI_EBIU_SDRAM_BANK_VALUE
CAS latency	ADI_EBIU_CMD_SET_SDRAM_CL_THRESHOLD	u32
TRAS	ADI_EBIU_CMD_SET_SDRAM_TRASMIN	ADI_EBIU_TIME
TRP	ADI_EBIU_CMD_SET_SDRAM_TRPMIN	ADI_EBIU_TIME
TRCP	ADI_EBIU_CMD_SET_SDRAM_TRCDMIN	ADI_EBIU_TIME
TWR	ADI_EBIU_CMD_SET_SDRAM_TWRMIN	ADI_EBIU_TIMING_VALUE
Refresh period	ADI_EBIU_CMD_SET_SDRAM_REFRESH	ADI_EBIU_TIMING_VALUE
Extended Mode Enable	ADI_EBIU_CMD_SET_SDRAM_EMREN	ADI_EBIU_SDRAM_EMREN
PASR*	ADI_EBIU_CMD_SET_SDRAM_PASR	ADI_EBIU_SDRAM_PASR
TCSR**	ADI_EBIU_CMD_SET_SDRAM_TCSR	ADI_EBIU_SDRAM_TCSR
External Buffer are used	ADI_EBIU_CMD_SET_SDRAM_EBUFE	ADI_EBIU_SDRAM_EBUFE
Fast Back-to-Back-Read/Write	ADI_EBIU_CMD_SET_SDRAM_FBBRW	ADI_EBIU_SDRAM_FBBRW
Control disable during bus grant	ADI_EBIU_CMD_SET_SDRAM_CDDBG	ADI_EBIU_SDRAM_CDDBG
Power Up Start Delay	ADI_EBIU_CMD_SET_SDRAM_PUPSD	ADI_EBIU_SDRAM_PUPSD
SDRAM powerup sequence	ADI_EBIU_CMD_SET_SDRAM_PSM	ADI_EBIU_SDRAM_PSM

表1.EBIU コマンドの概要

パラメータを設定した後、サービスを初期化する必要があります。このため、コマンド・テーブルにパラメータをバンドルした後に `adi_EBIU_init` 関数を呼び出します。

```
ADI EBIU COMMAND PAIR Sdram Values[] = {

        { ADI_EBIU_CMD_SET_SDRAM_BANK_SIZE,          (void*)&bank_size  },
        { ADI EBIU CMD SET SDRAM BANK COL WIDTH,     (void*)&bank caw   },
        { ADI EBIU CMD SET SDRAM CL THRESHOLD,       (void*)&MyCAS      },
        { ADI EBIU CMD SET SDRAM TRASMIN,            (void*)&MyTRAS     },
        { ADI EBIU CMD SET SDRAM TRPMIN,             (void*)&MyTRP      },
        { ADI EBIU CMD SET SDRAM TRCDMIN,            (void*)&MyTRCD     },
        { ADI_EBIU_CMD_SET_SDRAM_TWRMIN,             (void*)&MyTWR      },
        { ADI EBIU CMD SET SDRAM REFRESH,            (void*)&Refresh    },
        { ADI EBIU CMD SET SDRAM FBBRW,              (void*)&MyFBBRW    },
        { ADI_EBIU_CMD_SET_SDRAM_EMREN,              (void*)&MyEMREN    },
        { ADI_EBIU_CMD_SET_SDRAM_PASR,               (void*)&MyPASR     },
        { ADI_EBIU_CMD_SET_SDRAM_TCSR,               (void*)&MyTCSR     },
        { ADI_EBIU_CMD_SET_SDRAM_EBUFE,              (void*)&MyEBUFE    },
        { ADI_EBIU_CMD_SET_SDRAM_CDDBG,              (void*)&MyCDDBG    },
         { ADI_EBIU_CMD_SET_SDRAM_PUPSD,             (void*)&MyPUPSD    },
         { ADI_EBIU_CMD_SET_SDRAM_PSM,               (void*)&MyPSM      },
        { ADI EBIU CMD END, 0}
    };
//Init the service and ensure that the Refresh rate is reset if fsclk is changing
Result = adi_ebiu_Init( Sdram_Values, true);   // true enables automatic adjustment
```

リスト 4.EBIU の初期化

ADSP-BF537 プロセッサと ADSP-BF561 プロセッサのシステム・サービスの使用をデモストレーションするコード例は、この EE ノートに添付されている .ZIP ファイルに含まれています。

2.4 OTP メモリ内の値による SDRAM の初期化

OTP メモリを内蔵するデバイス (ADSP-BF51x、ADSP-BF52x、ADSP-BF54x の各プロセッサ)は、EBIU レジスタ設定値を初期化するもう 1 つの方法を提供します。ブート時に、ブート ROM プログラムが OTP に設定された値を使って EBIU 設定値を初期化することができます。OTP は PBS (Preboot Settings) ブロックと呼ばれる領域から構成され、この領域は OTP_write 関数を使って書き込む必要があります。

PBS02L ページに EBIU 設定値が含まれています。概要については、プロセッサのハードウェア・リファレンスを参照してください。

EBIU コントロール・レジスタを設定した後、SDRAMを初期化するため、ブート・プロセスはアドレス 0x00000000 のSDRAM をアクセスします。デフォルトでは、読み出しアクセスの設定が実行されます。読み出しアクセスは書き込みアクセスより時間を要するため、OTP_EBIU_POWERON_DUMMY_WRITE ビット (図 11)を使って、このアクセスをカスタマイズすることができます。このカスタマイズにより読み出しアクセスが書き込みアクセスで置き換えられて時間が節約されます。このオプションを使用することにより、リセットまたはハイバネート状態になる前に、このアドレスには重要なデータを保存しないことを保証する必要があります。

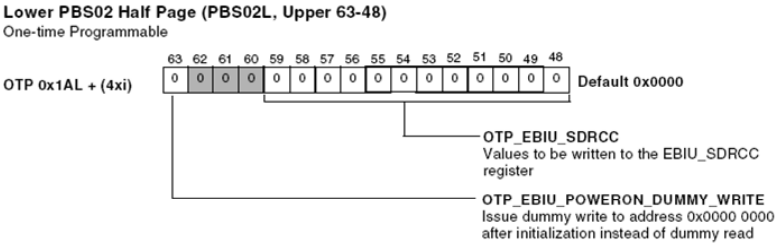


図11.ダミー書き込みオプション

```
/* Declare local variable */
DU64 Data;
.
.
/* Enable the EBIU Init settings */
Data.h = 0x04000000    | //use the content of PBS02L for the EBIU
         ...           | //other settings
Data.l = ...

otp_write(0x18, OTP_LOWER_HALF, &Data);

/* Assign SDRAM values */

/* The low data byte is the EBIU_SDGCTL register */
Data.l =     ~CDDBG    & // Control disable during bus grant off
             ~FBBRW    & // Fast back to back read to write off
             ~EBUFE    & // External buffering enabled off
             ~SRFS     & // Self-refresh setting off
             ~PSM      & // Powerup sequence mode (PSM) first
             ~PUPSD    & // Powerup start delay (PUPSD) off
              TCSR     | // Temperature compensated self-refresh at 85
              EMREN    | // Extended mode register enabled on
              PSS      | // Powerup sequence start enable (PSSE) on
              TWR_2    | // Write to precharge delay TWR = 2 (14-15 ns)
              TRCD_3   | // RAS to CAS delay TRCD =3 (15-20ns)
              TRP_3    | // Bank precharge delay TRP = 2 (15-20ns)
              TRAS_6   | // Bank activate command delay TRAS = 4
              PASR_B0  | // Partial array self refresh Only SDRAM Bank0
```

```
                CL_3      | // CAS latency
                SCTLE;

/* The upper 32 bit are EBIU_SDBCTL(0-15) and EBIU_SDRRC (16-27). Further we will
   Set the dummy write bit (32), which will speed up the initialization          */

Data.h =      0x80000000      | // dummy write bit
              (0x406)<<16     | // Refresh rate
              EBCAW_9         | //Page size 512
              EBSZ_64         | //64 MB of SDRAM
              EBE;              //SDRAM enable

;
otp_write(0x1A, OTP_LOWER_HALF, &Data);
```

リスト 5. PBS による初期化

2.5 アプリケーションをロードする前の、初期化コードによるメモリの初期化

初期化時に命令セクションとデータ・セクションを SDRAM に配置する場合は、アプリケーションをロードする前に SDRAM を初期化する初期化ファイルを使う必要があります。このため、プロジェクトを開始して初期化ファイルのコードを書きます。このプロジェクトを.DXE ファイルにビルドします。この初期化 .DXE ファイルは、Project Options ダイアログ・ボックス (Project:Load:Options ページ)を使って実際のアプリケーションのローダ・ファイルへインクルードすることができます。

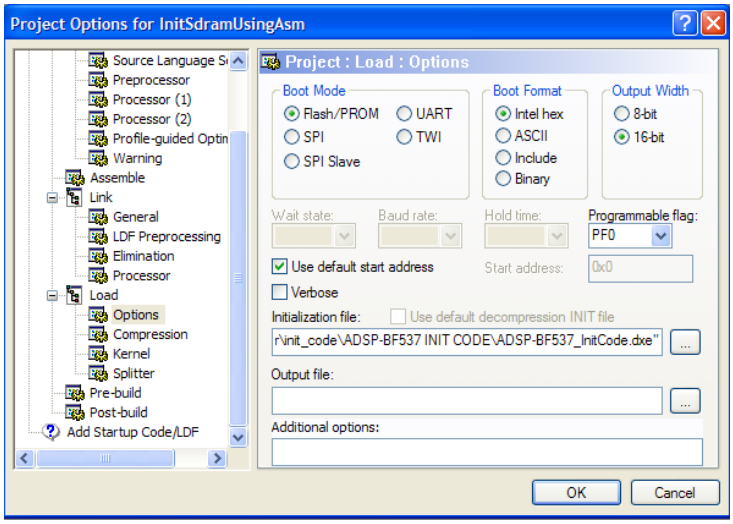


図12.プロジェクト・ロード・オプションの指定

リスト 6 に、初期化コードの例を示します。

```
#include <defBF537.h>
.section program;
    //save all registers on the stack
    [--SP] = ASTAT;
    ...
    [--SP] = L3;

//Setup the PLL Settings
    //Ensure no other interrupt will disturb us
    CLI R1;
    //Setup the voltage regulator
    P0.L = lo(VR_CTL);
    P0.H = hi(VR_CTL);
    R0 = 0x40DB (z);
    w[P0] = R0;

    //Wait the VR settings are done
    idle;
    //Setup the DIV of ssclk and cclk
    P0.L = lo(PLL_DIV);
    R0 = 0x0003 (z);
    w[P0] = R0;

    //Setup wait cycles until PLL is set
    P0.L = lo(PLL_LOCKCNT);
    R0 = 0x200 (z);
    w[P0] = R0;

    //Setup the PLL
    P0.L = lo(PLL_CTL);
    R0 = 0x2000 (z);
    w[P0] = R0;

    //Wait the PLL settings are done
    idle;
    //restore interrupts
    STI R1;

//Our program needs 4 MBs of RAM
    P0.L = lo(EBIU_AMGCTL);
    P0.H = hi(EBIU_AMGCTL); //Asynchronous Memory Global Control Register
//Uncomment your setting
    R0 = 0x00F0
//    |AMBEN_NONE(Z);          //No Asynchronous Memory
//    |AMBEN_B0(Z);            //1MB Asynchronous Memory
//    |AMBEN_B0_B1(Z);         //2MB Asynchronous Memory
//    |AMBEN_B0_B1_B2(Z);      //3MB Asynchronous Memory
      |AMBEN_ALL(Z);           //4MB Asynchronous Memory
W[P0] = R0;

//Setup the SDRAM
//SDRAM Refresh Rate Setting
    P0.H = hi(EBIU_SDRRC);
    P0.L = lo(EBIU_SDRRC);
    R0 = 0x406 (z);
    w[P0] = R0;

//SDRAM Memory Bank Control Register
    P0.H = hi(EBIU_SDBCTL);
    P0.L = lo(EBIU_SDBCTL);
    R0 =         EBCAW_9   | //Page size 512
```

```
                EBSZ_64    | //64 MB of SDRAM
                EBE;         //SDRAM enable
    w[P0] = R0;

//SDRAM Memory Global Control Register
    P0.H = hi(EBIU_SDGCTL);
    P0.L = lo(EBIU_SDGCTL);
    R0.H=        hi(    ~CDDBG     & // Control disable during bus grant off
                        ~FBBRW     & // Fast back to back read to write off
                        ~EBUFE     & // External buffering enabled off
                        ~SRFS      & // Self-refresh setting off
                        ~PSM       & // Powerup sequence mode (PSM) first
                        ~PUPSD     & // Powerup start delay (PUPSD) off
                        TCSR       | // Temperature compensated self-refresh at 85
                        EMREN      | // Extended mode register enabled on
                        PSS        | // Powerup sequence start enable (PSSE) on
                        TWR_2      | // Write to precharge delay TWR = 2 (14-15 ns)
                        TRCD_3     | // RAS to CAS delay TRCD =3 (15-20ns)
                        TRP_3      | // Bank precharge delay TRP = 2 (15-20ns)
                        TRAS_6     | // Bank activate command delay TRAS = 4
                        PASR_B0    | // Partial array self refresh Only SDRAM Bank0
                        CL_3       | // CAS latency
                        SCTLE      ); // SDRAM clock enable

    R0.L=     lo(       ~CDDBG     & // Control disable during bus grant off
                        ~FBBRW     & // Fast back to back read to write off
                        ~EBUFE     & // External buffering enabled off
                        ~SRFS      & // Self-refresh setting off
                        ~PSM       & // Powerup sequence mode (PSM) first
                        ~PUPSD     & // Powerup start delay (PUPSD) off
                        TCSR       | // Temperature compensated self-refresh at 85
                        EMREN      | // Extended mode register enabled on
                        PSS        | // Powerup sequence start enable (PSSE) on
                        TWR_2      | // Write to precharge delay TWR = 2 (14-15 ns)
                        TRCD_3     | // RAS to CAS delay TRCD =3 (15-20ns)
                        TRP_3      | // Bank precharge delay TRP = 2 (15-20ns)
                        TRAS_6     | // Bank activate command delay TRAS = 4
                        PASR_B0    | // Partial array self refresh Only SDRAM Bank0
                        CL_3       | // CAS latency
                        SCTLE    ) ; // SDRAM clock enable
    [P0] = R0;
    ssync;
    //Restore registers from the stack
    L3 = [SP++];
    ...
    ASTAT = [SP++];
    RTS;
```

リスト 6.初期化ファイルによる初期化

3 .LDF ファイルを使用した、データとプログラム・コードのメモリへの配置

アプリケーションをロードする前にSDRAMを初期化する場合 (OTP メモリ内の値によるSDRAM の初期化を参照)、データとコードをメモリ・セクションに配置することができます。このため、リンカー・ディスクリプション・ファイル (.LDF)内でメモリ領域を定義して、メモリへ配置されるデータまたはコードについてリンカーに知らせる必要があります。

ここで、データを SDRAM バンク 1 に配置するものとします。VisualDSP++ Expert リンカー・ウィザードで生成された標準リンカー・ディスクリプション・ファイル (.LDF)を詳しく調べると(この例では 512 MB の SDRAM を使用)、内部 SDRAM バンクは別々の名前になることが分かります。

```
MEMORY
{   MEM_SYS_MMRS         { TYPE(RAM) START(0xFFC00000) END(0xFFDFFFFF) WIDTH(8) }
    […]
    MEM_SDRAM0_BANK0          { TYPE(RAM) START(0x00000004) END(0x07ffffff) WIDTH(8) }
    MEM_SDRAM0_BANK1          { TYPE(RAM) START(0x08000000) END(0x0fffffff) WIDTH(8) }
    MEM_SDRAM0_BANK2          { TYPE(RAM) START(0x10000000) END(0x17ffffff) WIDTH(8) }
    MEM_SDRAM0_BANK3          { TYPE(RAM) START(0x18000000) END(0x1fffffff) WIDTH(8) }
} /* MEMORY */
```

リスト 7.メモリ割り当て

次に示すように、“SECTIONS” 領域でもメモリ・ラベル “MEM_SDRAM0_BANK1”が見つかります。

```
PROCESSOR p0
{
   OUTPUT($COMMAND_LINE_OUTPUT_FILE)
   RESOLVE(start, 0xFFA00000)
   KEEP(start, _main)

   SECTIONS
   {     [...]

      sdram0_bank1
      {
         INPUT_SECTION_ALIGN(4)
         INPUT_SECTIONS($OBJECTS(sdram0) $LIBRARIES(sdram0))
         INPUT_SECTIONS($OBJECTS(sdram0_bank1) $LIBRARIES(sdram0_bank1))
         INPUT_SECTIONS($OBJECTS(sdram0_data) $LIBRARIES(sdram0_data))
         INPUT_SECTIONS($OBJECTS(cplb) $LIBRARIES(cplb))
         INPUT_SECTIONS($OBJECTS(data1) $LIBRARIES(data1))
         INPUT_SECTIONS($OBJECTS(voldata) $LIBRARIES(voldata))
         INPUT_SECTIONS($OBJECTS(constdata) $LIBRARIES(constdata))
         INPUT_SECTIONS($OBJECTS(cplb_data) $LIBRARIES(cplb_data))
         INPUT_SECTIONS($OBJECTS(.edt) $LIBRARIES(.edt))
         INPUT_SECTIONS($OBJECTS(.cht) $LIBRARIES(.cht))
     } > MEM_SDRAM0_BANK1

         [...]

   } /* SECTIONS */
} /* p0 */
```

リスト 8.メモリ・セクションの指定

幾つかのライブラリとオブジェクトがこのメモリ・空間に配置されていることが分かります。オブジェクトまたはライブラリを複数のメモリ・セクションに配置することもできます。このケースでは、メモリの各セクションに配置するデータまたはコードの部分をリンカーが決めています。

C/C++を使用して、データを明示的にメモリ・セクションに配置する場合は、pragma section ディレクティブを使ってください。このケースでは、グローバル変数 x を内部 SDRAM バンク 1 に配置します。ラベル sdram0_bank1 は内部 SDRAM バンク 1 にのみ割り当てられ、他のセクションには割り当てられません。したがって、このラベルを使って変数を次のように配置します。

```
//define a variable which lies in SDRAM Bank 1
#pragma section ("sdram0_bank1")
long int x = 0;
```

関数のプロトタイプをセクションに割り当てることにより、次のように命令コードで同じことを行うことができます。

```
//define a function prototype of a function which lies in SDRAM Bank 1
#pragma section ("sdram0_bank1")
void foo();
```

システムのSDRAM 性能の強化で説明したように、データ・セクションを手動で配置して、ページのオープンとクローズにより発生する遅延を回避することが役立つ場合もあります。このため、.LDF ファイル内に次のようにメモリ・セクションを定義します。

```
   MEM_SDRAM0_BANK2         { TYPE(RAM) START(0x02000000) END(0x02ffffff) WIDTH(8) }
   MEM_SDRAM0_BANK3         { TYPE(RAM) START(0x03000800) END(0x03ffffff) WIDTH(8) }
   //define my own page
   MEM_SDRAM0_BANK3_PAGE0   { TYPE(RAM) START(0x03000000) END(0x030007FF) WIDTH(8) }
```

リンカー・ディスクリプション・ファイルの SECTIONS 部分で、MyDefinedMemory とラベルされたすべてのオブジェクトをリンクしてこのメモリ空間に入れます。

```
sdram0_bank3_page_0
 {
   INPUT_SECTION_ALIGN(4)
   INPUT_SECTIONS($OBJECTS(MyDefinedMemory) )
 } > MEM_SDRAM0_BANK3_PAGE0
```

その後、C/C++ ファイル内で、配列をこのメモリ・セクションに配置します。

```
//define an array which lies in my own defined memory space (Bank 3 Page 0)
#pragma section ("MyDefinedMemory")
long int MyArray[100];
```

4 SDRAM のハードウェア・デザイン

SDRAM は 50 MHz より高い周波数で駆動されるため、ハードウェア・レイアウトでは高速デザインの条件を満たす必要があります。今日では、ハードウェア・デザインは EMI と EMC に関する多くの規格を満たしています。高い周波数で信号インテグリティを保証し、その多くは低消費電力環境に置かれます。このため、正しいプリント回路ボード (PCB) デザインが重要になります。このセクションでは、PCB 上の Blackfin プロセッサと SDRAM との間の接続のデザイン方法について説明します。

4.1 SDRAMとBlackfinプロセッサとの接続 (回路図)

Blackfin プロセッサは SDRAM に対する外付け部品不要のインターフェースを提供しています。Blackfin プロセッサに応じて、1.8V～3.3V の電源条件を持つ SDRAM がサポートされています。

多くの設計ミスは、Blackfin プロセッサのアドレス・ピンが正しく SDRAM に接続されていないときに発生します。

アドレス・ラインは次のように接続する必要があります。

4.1.1 ADSP-BF53x シリーズ・プロセッサ

- Blackfin プロセッサの ADDR1 を SDRAM の A0 へ、ADDR2 を A1 へ、以下同様に接続します。
- Blackfin の ADDR11 は使用しないでください。SA10 を A10 へ接続します。
- ADDR18 を SDRAM の BA0 へ接続します。
- ADDR19 を SDRAM の BA1 へ接続します。
- /ABE0 を DQML ピンへ接続するか (16 ビット SDRAM の場合)、あるいは D0～D7 に接続されたチップの DQM へ接続します。
- /ABE1 を DQMH ピン (16 ビット SDRAM の場合)に接続するか、あるいは D8～D15 に接続されたチップの DQM へ接続します。

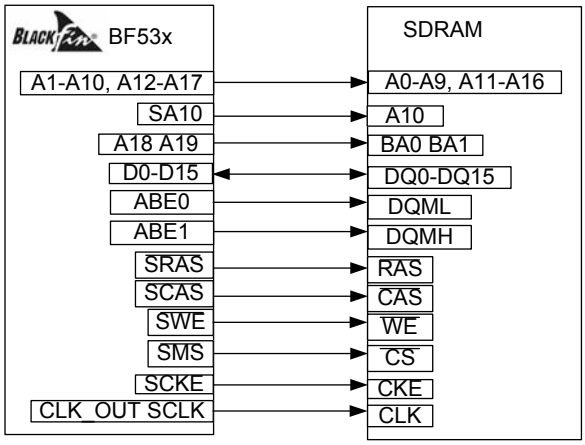


図13.ADSP-BF53x プロセッサ: Blackfin プロセッサと SDRAM との間の接続

4.1.2 ADSP-BF561 プロセッサ (16 ビット SDRAM)

- Blackfin プロセッサの ADDR2 を SDRAM の A1 へ、ADDR3 を A2 へ、以下同様に接続します。
- Blackfin プロセッサの SDQM3 を SDRAM の A0 へ接続します。
- Blackfin プロセッサの ADDR11 は使用しないでください。SA10 を A10 へ接続します。
- ADDR18 を SDRAM の BA0 へ接続します。
- ADDR19 を SDRAM の BA1 へ接続します。
- SDQM0 を DQML または DQM へ接続します (上記参照)

- SDQM1 を DQMH または DQM へ接続します (上記参照)
- /SMSx を /CS ラインへ接続します。
  (x16 を使い複数の SDRAM を使う場合: 各チップに/SMS ラインを 1 本接続)
  (x8 を使い 3 個以上の SDRAM を使う場合: 各 2 個対に /SMS ラインを 1 本接続)

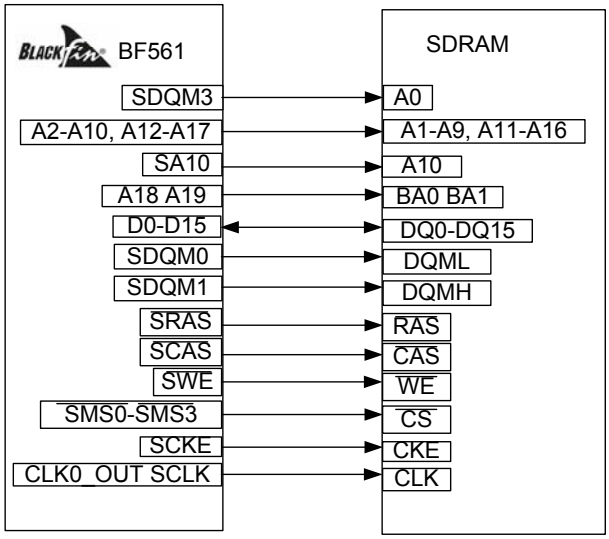


図14.ADSP-BF561 プロセッサ: Blackfin プロセッサと 16 ビット SDRAM との間の接続

4.1.3 ADSP-BF561 プロセッサ (32 ビット SDRAM)

- Blackfin プロセッサの ADDR2 を SDRAM の A0 へ、ADDR3 を A1 へ、以下同様に接続します。
- Blackfin の ADDR12 は使用しないでください。SA10 を A10 へ接続します。
- ADDR18 を SDRAM の BA0 へ接続します。
- ADDR19 を SDRAM の BA1 へ接続します。
- SDQM0 を SDRAM1(D0～D15)の DQML へ接続します。
- SDQM1 を SDRAM1 の DQMH へ接続します。
- SDQM3 を SDRAM2(D16～D31)の DQML へ接続します。
- SDQM4 を SDRAM2 の DQMH へ接続します。

8 ビット SDRAM を使用する場合

- SDQM0 を SDRAM1(D0～D7)の DQM へ接続します。
- SDQM0 を SDRAM2(D8～D15)の DQM へ接続します。
- SDQM0 を SDRAM3(D16～D23)の DQM へ接続します。
- SDQM0 を SDRAM4(D24～D31)の DQM へ接続します。

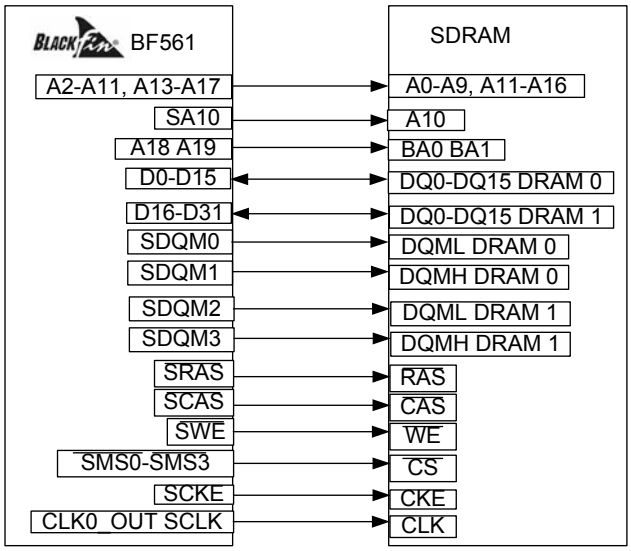


図15.ADSP-BF561 プロセッサ: Blackfin プロセッサと 32 ビット SDRAM との間の接続

/BR ピンがフローティングのままの場合にも問題が発生します。Blackfin プロセッサはこの信号をバス要求と解釈して/BG 信号で応答します。このためにパラレル・バスが永久にブロックされてしまいます。したがって、/BR ピンには必ずプルアップ抵抗が必要です。

コンデンサを使用して SDRAM 電源をデカップリングしてください。

SDRAM の各データ・ピンの近くに直列抵抗を接続してください。次のセクションでは、抵抗値の決定方法について説明します。

4.2 高速デザイン

SDRAM 接続のレイアウトは、特に低消費電力デザインでは重要です。このセクションでは、アプリケーションの要求に合うように SDRAM レイアウトのデザインを最適化する方法を説明します。最も重要な接続は、クロック、下位アドレス・ライン、DQM、データ・ラインです。

4.2.1 信号品質に与える影響

このセクションでは、ハードウェア・デザインから受ける影響について説明します。

反射

接続ラインのインピーダンスがレシーバの入力抵抗に等しい場合、エネルギーは抵抗にフルに吸収されます。そうでない場合は、送信エネルギーが反射されます。この反射が重畳されて、必要な信号と干渉します。

カップリング

さまざまなパターンを流れる電流が相互に影響を与えます。変化する電流がパターン A を流れると、変化する磁界が発生してパターン B へカップリング(結合) します。この結合によりパターン B に結合係数に依存する電流が発生します。誘導される電流の向きはパターン A の電流と逆です。 2 本の信号パターンが相互に干渉する場合 (クロストーク)、悪影響を与えます。その干渉が信号ラインとそのリターン・ライン (GND)との間である場合は、良い影響を与えることができます。結合した信号は、リターン信号の増幅に役立ち、リターンする信号は元の信号を増幅します。

4.2.2 反射の防止

特性インピーダンスを計算して正しい終端抵抗を接続してください。

伝送ラインの終端には次の方法があります。

- 直列終端
- 並列終端
- テブナン終端
- ダイオード終端
- AC 終端

SDRAM デバイスに対する最も重要技術を次に説明します。

直列終端

SDRAMには直列終端 (図 16)を使用してください。トランスミッタの出力ピンの近くに直列抵抗を接続します。利点はDC 電流が流れないことです (並列終端の場合は流れます)。これは低消費電力デザインには不可欠です。欠点は、レシーバ端でほぼ 100%の反射が発生することです。この反射がトランスミッタまで戻ります。SDRAM のパターンは短く、さらにBlackfin プロセッサから各SDRAM チップまでの信号パターンはほぼ同じ長さであるため、これによるSDRAM機能への影響はほとんどありません。

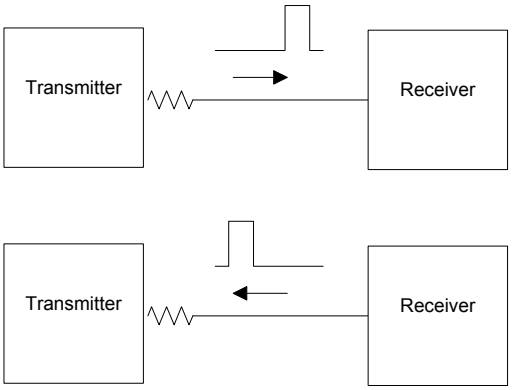


図16.直列終端の使用

ドライバ (トランスミッタ)からの 2 回目の反射を防止するため、抵抗は正しい値である必要があります。直列終端抵抗の値は、その総和およびドライバの出力インピーダンスがパターンのインピーダンスに一致するように設定する必要があります。次の式が得られます。

$R_S = Z_0 - Z_{OUT}$ 、ここで Z_{OUT} はトランスミッタの出力インピーダンスです。

読み出しコマンドの方が書き込みコマンドよりクリティカルです。このため、データ・ラインの抵抗をできるだけ SDRAM データ・ピンの近くに接続してください。

並列終端

これに代わるのが並列終端 (図 17)です。前述のように、このセクションの終わりに示すデザイン・ガイドラインに従う場合は、標準 SDRAMに対してこれは不要です。

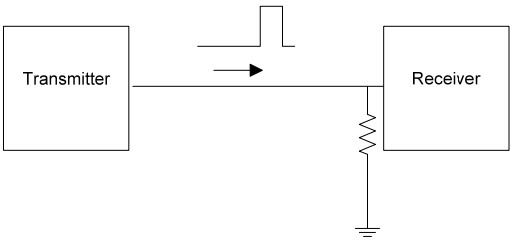


図17.並列終端の使用

$$\text{Reflection coefficent } \rho = \frac{Z_L - Z_0}{Z_L + Z_0}$$

反射をできるだけ小さくするためには、並列終端抵抗 (Z_L) と Z_0 が一致する必要があります。

- 終端がそれでも必要か否か検討してください。抵抗を追加すると EMI が増えます。

ADSP-BF51x プロセッサと ADSP-BF52x プロセッサの終端は必ず必要です。詳細については、データシートを参照してください。

4.3 SDRAM 接続のデザイン・ガイドライン

EMC と信号インテグリティに関して、次のデザイン・ガイドラインが推奨されます。SDRAM PCB のレイアウトを開始するときは、すべての信号を同じに扱わずに、各信号の重要度を考慮し、最もクリティカルな信号のパターンを先に配置してください。次の順序が推奨されます。

1. クロック分配
2. データ・ラインと DQM ライン、SA10 などのコマンド・ライン
3. アドレス・ライン
4. その他の信号 (例えば CKE)

4.3.1 部品配置での考慮事項

PCB のレイアウトでは次のポイントを考慮してください。

- SDRAM チップは Blackfin プロセッサの近くに配置します。
- パターンをできるだけ短くします。
- 信号を分配するときは、可能な場合、デバイスまでのすべてのパターンを同じパス長にします (図 18)。図 19に示すようなループは回避してください。

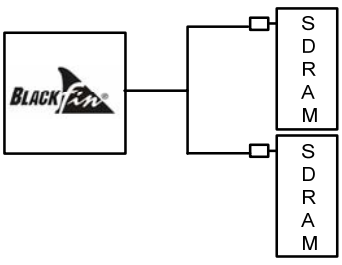


図18.適切: パターンの分岐は同じ長さで

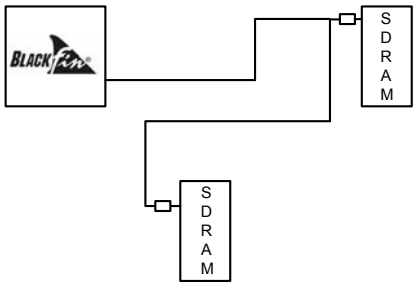


図19.不適切: ループを構成するパターン

4.3.2 パターン・エッジでレイアウト・ツールのラウンディング機能を使用

図 20 に、ラウンディング機能を使用しないPCB パターン・エッジを示します。 図 21に、ラウンディング機能を使用した同じパターン・エッジを示します。

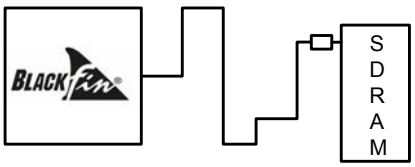


図20.不適切: ラウンディング機能を使用しないパターン・エッジ

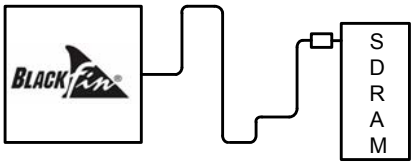


図21.適切: ラウンディング機能を使用したパターン・エッジ

4.3.3 VCC プレーンと GND プレーンをできるだけ近づけて配置

図 22 に、クリティカルな信号を分離していない 4 層 PCBを示します。 図 23 に、適切に分離した 4 層 PCBを示します。

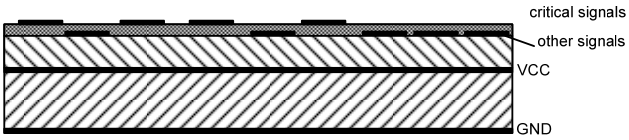


図22.不適切: VCC プレーンと GND プレーンとの間隔が大きすぎる例

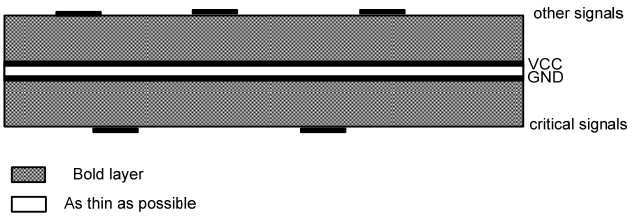


図23.適切: VCC プレーンと GND プレーンとの間隔が小さい例

4.3.4 クリティカルな信号を中間層に配置する分離

図 24 に、クリティカルな信号を分離していない 6 層 PCBを示します。 図 25 に適切な分離を示します。

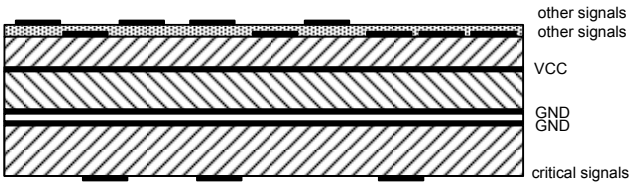


図24.不適切: クリティカルな信号が適切に分離されていない例

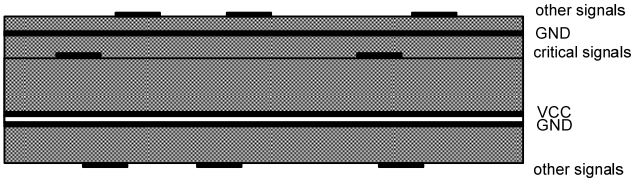


図25.適切: クリティカルな信号とグラウンド・プレーンとの距離を最小にする

4.3.5 直列抵抗を SDRAM の近くに配置

図 26 に、ビア数が多すぎる信号パスを示します。直列抵抗がSDRAMから離れすぎています。 図 27 に、ビアのない短い信号パスとSDRAMの近くに配置した直列抵抗を示します。

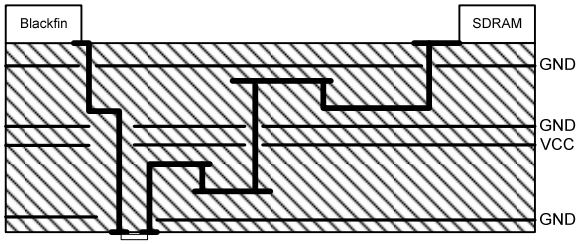


図26.不適切: ビアが多すぎるクリティカルな信号パスと離れすぎている直列抵抗

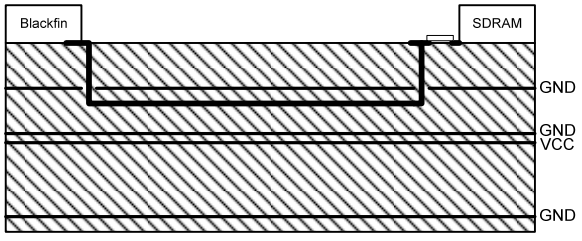


図27.適切: SDRAM の近くに配置した直列抵抗

4.3.6 GND プレーンでのトレンチ(溝)の回避

図 28 に、トレンチがあるGND プレーンを示します。図 29 のGND プレーンではトレンチの発生を回避しています。

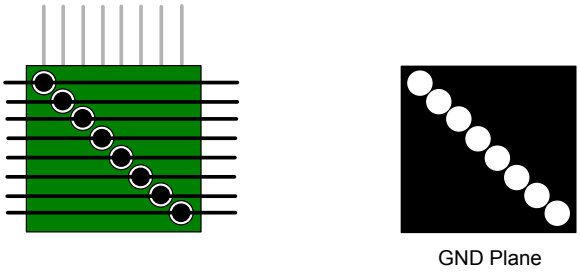


図28.不適切: トレンチのあるグラウンド・プレーン

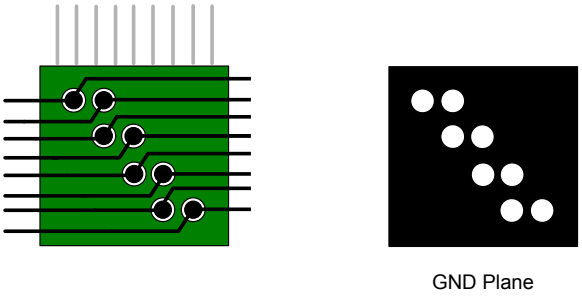


図29.適切: トレンチのないグラウンド・プレーン

4.3.7 ビアからの逆流パスの最小化

ビア内で 1 つの層から他の層への方向変化を回避できない場合には、逆流パスを最小にしてみてください。 図 30に、ビアから 2 方向へ走る信号パスを示します。図 31の方向変化は良くなっています。

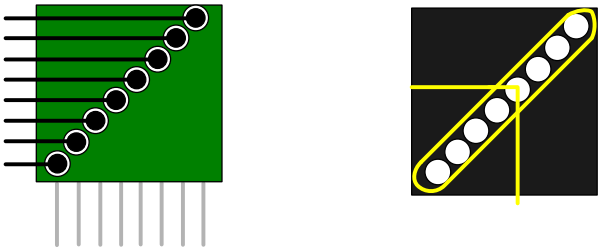


図30.不適切: 層—層間の方向変化

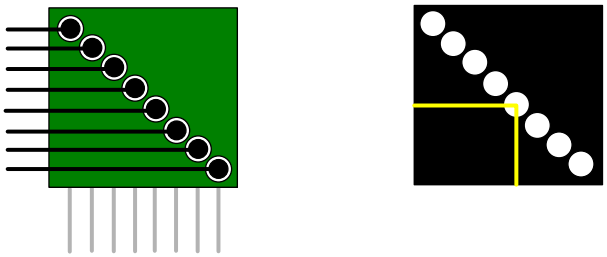


図31.適切: 方向の変化を回避

4.3.8 駆動強度制御機能の利用

電磁放射を減らすためには、EBIU ピンの駆動強度を下げてください。ただし信号には注意してください。タイミング仕様内でラインとピンの容量を充電する十分な電流を供給する必要があります。ビット・フィールドが 00b の場合は低駆動強度 (データ・シート参照)を、01b 場合は高駆動強度を、それぞれ指定します。

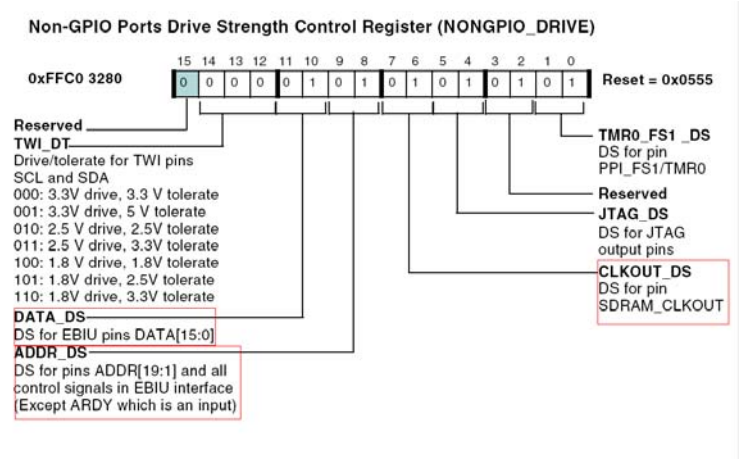


図32. BF52x の駆動強度コントロール・レジスタ

4.3.9 スルー制御機能の利用

変化を滑らかにすることも、電磁放射の軽減に役立ちます。スルーレートを低くした後にも、タイミング仕様を満たしていることを再度確認してください。00b は高速なスルーレートを、01b は低速なスルーレートを、それぞれ指定します。

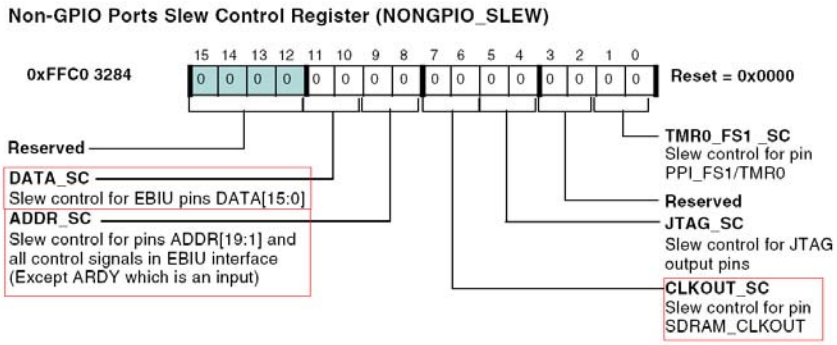
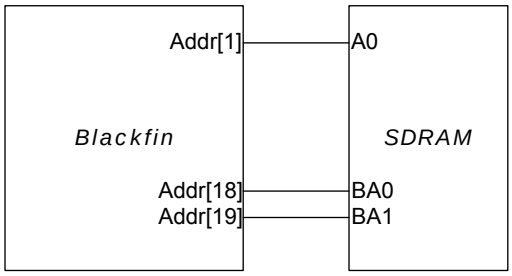


図33.BF52x のスルーレート・レジスタ

5 16 MB未満のSDRAMでのBlackfin プロセッサの使用

16 MB (128 M ビット)未満の SDRAM の使用は、低消費電力アプリケーションで特に重要です。このセクションでは、16 MB 未満を使用するアプリケーションに対するガイダンスを提供します。

5.1 システムの設定



ハードウェア側には、特別な設定はありません。SDRAM ハードウェア Designに説明するようにアドレス・ラインを接続するだけで済みます。

16 MB未満を使用する最初のステップは、EBIU_SDBCTL (SDRAM メモリ・バンク・コントロール) レジスタの SDRAM 外部バンク・サイズ・ビットに 16 Mバイトを設定することです。この設定により、Blackfin プロセッサの内部アドレスは 16 Mバイト用になり、アドレス空間は 図 34のように分割されます。

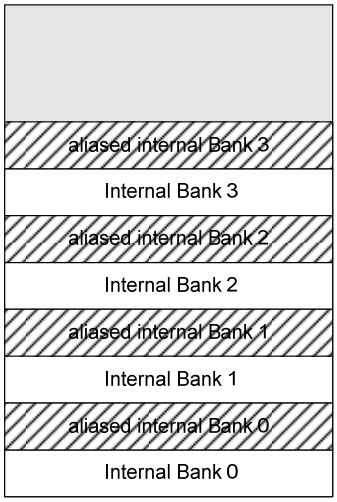


図34.アドレス空間

“エイリアス” アドレス空間の値は、アコーディング・バンクのコピーであり、この空間に対する書き込みアクセスごとに、“リアル” バンクへの書き込みアクセスが行われます。このことは、リンカー・ディスクリプション・ファイル (.LDF)で考慮する必要があります。そうしないと、Blackfin プロセッサは命令とデータを存在しないアドレスに配置してしまいます。

このタイプのアドレス・エラーは Blackfin プロセッサから検出できません。アドレスが有効か否かをテストする機能はありません。このために、後で、存在しないアドレス空間からコアが読み出しを行おうとすると、アプリケーションでエラーが発生し、ダミー値を取得するのでそれを有効な命令コードまたはデータとして解釈してしまいます。

5.2 .LDF ファイルの変更

まず、.LDF ファイルが Expert Linkerウィザードによって変更されないようにします。そのため、Project Optionsダイアログ・ボックス (Project -> Project Options)を開き、Remove Startup Code/LDFページに行き、Leave the files in the project, but stop regenerating them オプションを選択します (図 35)。

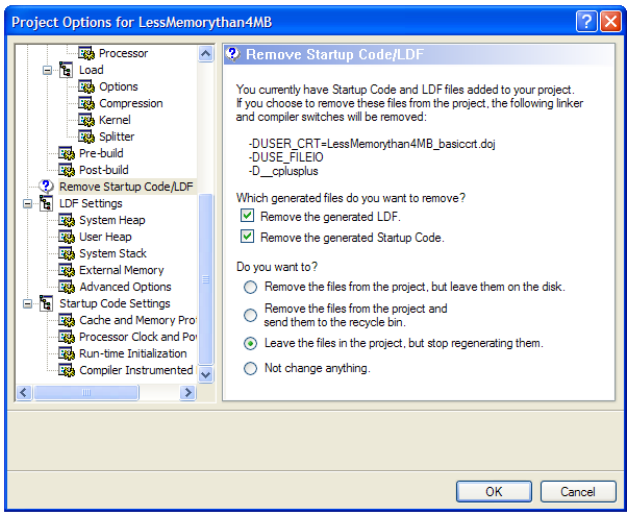


図35. .LDF ファイルを変更されないようにします

そうすると、.LDF ファイルを手動で変更できるようになります。

例えば、64M ビット SDRAM (8 MB)を使用する場合、リンカー・ディスクリプション・ファイル内のアドレス空間は次のようになります。

```
MEM_SDRAM0_BANK0 { TYPE(RAM) START(0x00000000) END(0x001FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK1 { TYPE(RAM) START(0x00400000) END(0x005FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK2 { TYPE(RAM) START(0x00800000) END(0x009FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK3 { TYPE(RAM) START(0x00C00000) END(0x00DFFFFF) WIDTH(8) }
```

図36. .LDF ファイルを変更されないようにします

図 36に示すように、アドレス空間には 2 MBの隙間があります。

5.2.1 解説: バックグラウンド

.LDF ファイルは、メモリ空間内に配置されるコードとデータの位置を指定します。EBIU 設定値は、 Blackfin プロセッサの SDRAM コントローラを設定し、SDRAM のサイズ、タイミング、機能を指定します。EBIU 設定では 8 MB SDRAM の設定ができないため、16 MB の SDRAM サイズを設定する必要があります。これはアドレシング機能にとってどんな意味があるのでしょうか?列アドレス幅 (CAW)が 10 ビットの SDRAM を使う場合、これは 2^10 = 1024 列のアドレス指定が可能なことを意味します。列アドレスと行アドレスにより、2 バイトのアドレシングになります (x16 SDRAM の場合)。ここで、RAM を 16 MB (=16777216 バイト)に設定します。これは行アドレス幅が 13 (メモリ・サイズ / (データ幅 * 2^アドレス) = 16777216/(2 バイト * 2^10)= 8192= 2^13)を意味しますが、8 MB の RAM しか使用しません。これは行アドレス幅 12 を意味しますが、コントローラは行アドレス幅を 13 と計算しています。行アドレスと列アドレスは SDRAM 時分割多重化へ送信されます。SDRAM を調べると、12 本のアドレス・ラインと 2 本のバンク・アドレス・ラインがありますが、コントローラは 13 と計算して 13 を使います。13 番目のアドレス・ラインは接続されていないので、行アドレスのビット 13 の状態に無関係に同じ物理アドレスをアドレス指定することになります。これが、メモリ空間がミラーされる理由です。

例えば、行アドレス 0x1000 は、行アドレス 0x0000 と同じデータをアクセスします。エイリアス・メモリ空間に何かを配置することは危険です。これは、他のアドレス空間内の何かを上書きすることになるためです。

したがって、次のように、ミラーされた RAM 内に何も配置しないように.LDF ファイルを定義します。

```
MEM_SDRAM0_BANK0 { TYPE(RAM) START(0x00000000) END(0x001FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK1 { TYPE(RAM) START(0x00400000) END(0x005FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK2 { TYPE(RAM) START(0x00800000) END(0x009FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK3 { TYPE(RAM) START(0x00C00000) END(0x00DFFFFF) WIDTH(8) }
```

5.3 2 バンク構成の SDRAM

SDRAMによってはバンクが 2 個しかないものがあります。ハードウェア接続は次のようになります (図 37)。

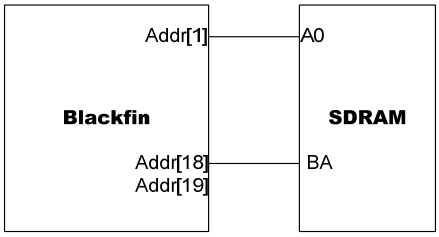


図37.2 バンク構成 SDRAM のハードウェア接続

BAは、SDRAMのバンク・セレクト・ピンです。このピンはBlackfin プロセッサのAddr[18]に接続する必要があります。Addr[19] はフローティングのままにしてください。他のアドレスは、SDRAM ハードウェア Designの説明のように接続します。EBIU_SDBCTL (SDRAM メモリ・バンク制御) レジスタに 16 Mバイトを設定します。論理アドレス空間は 図 38のように断片化します。

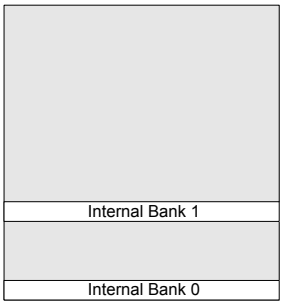


図38.断片化した論理アドレス空間

このため.LDF ファイルの調整を再度使って、メモリ空間をセットアップする必要があります。前述のように作業を進めてください。

図 39 に、16Mビット (2 MB) SDRAMのメモリ空間例を示します。

```
MEM_SDRAM0_BANK0 { TYPE(RAM) START(0x00000000) END(0x000FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK1 { TYPE(RAM) START(0x00400000) END(0x004FFFFF) WIDTH(8) }
```

図39.例: 16M ビット (2-MB) SDRAM のメモリ空間

図のように、アドレス空間には 3MB の隙間があります。

6 システムのSDRAM 性能の強化

多くのアプリケーションで、実行時間がキー・ファクタの 1 つになります。データと命令の配置は、アプリケーションの処理速度に大きな影響を与えることがあります。このセクションでは、SDRAM 性能を強化する方法の概要を紹介します。システム的な手法については、「*System Optimization Techniques for Blackfin Processors (EE-324)*[2]」を参照してください。

6.1 最適なマルチバンク・アクセス

アクティベート・コマンドによるページのオープンまたはプリチャージ・コマンドによるページのクローズには時間がかかります。したがって、ページの切り替えを減らすと、SDRAM 性能が向上します。

多くのアプリケーションに当てはまるケースは、メモリ DMA を使ってデータを 1 つの配列から別の配列へコピーすることです。この問題は、両配列が同じ内部バンク内にあるときに発生します。ページ内でこの問題が発生する場合は問題になりませんが、2 つの配列のサイズがページ・サイズを超える場合には、DMA は少なくとも 2 ページをアクセスすることになります。

図 40 に、DMAがバンク内で動作する方法を示します (シングル・バンク・アクセス)。

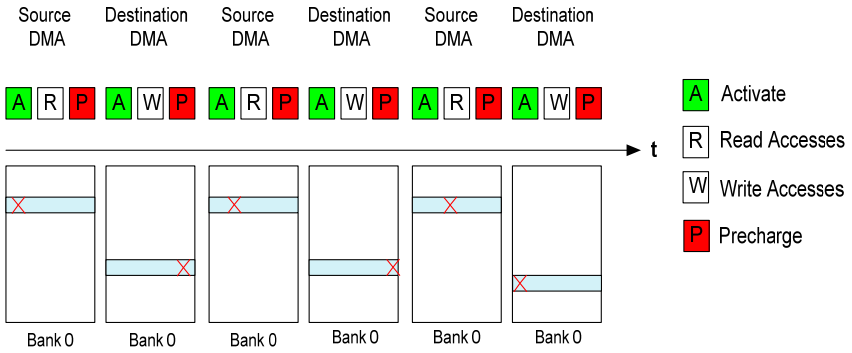


図40.1 メモリ・バンクに対するアクセス

データは、同じバンク内の異なるページに配置されています。ソース DMA とディステネーション DMA との間の各切り替え後に、アクティベートとプリチャージを実行する必要があります。さらに、コア (またはキャッシュ)もバンクをアクセスすることを考慮する必要があります。読み出しと書き込みとの間には毎回遅延が発生します。この遅延は内部 DMA のアーキテクチャにより大きくなることがあります。すなわち、DMA は特別な状況で待ち状態が追加される帰還制御ステート・マシンとしてデザインされているためです。

このような時間を要するケースを回避するためには、バンク間DMAコピーを可能にする方法でメモリを構成してください。 図 41 に、このような方法を示します。

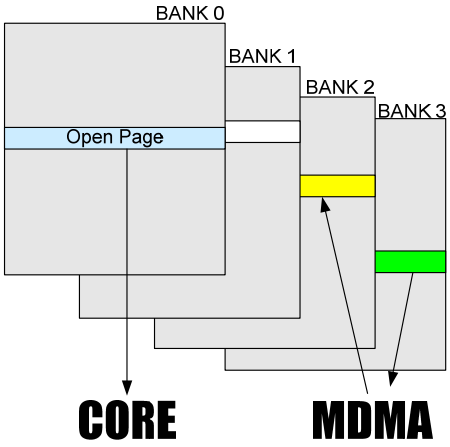


図41. DMA とコアによるマルチバンキング手法

コアはバンク 0 からコードを取得し、MDMA 転送はバンク 3 からバンク 2 へ転送します。 図 42 に、2 つのバンク間でのデータ転送のシーケンスを示します。図のように、プリチャージ・コマンドとアクティベート・コマンドの数は大幅に削減されます。前述のように、プリチャージ・コマンドとアクティベート・コマンドは時間を要する手順であるため、この技術は多くの時間を節約します。

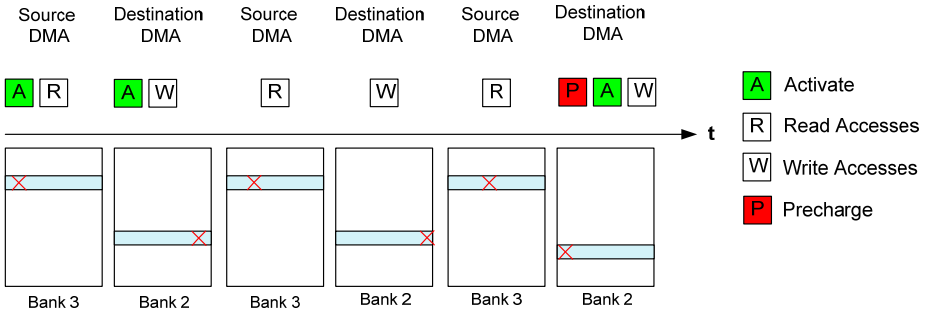


図42.インテリジェント・バンク・アクセス

6.2 最適なページ・アクセス

このセクションでは、ページ・アクセスの改善方法について説明します。ページ内にアクセスを維持できる場合でも、さらにアクティベート・コマンドとプリチャージ・コマンドが必要です (リフレッシュのために発行される他にも)。次に、ページを別々にアクセスする方法を示します。

`Project Options` ダイアログ・ボックスの `Remove Startup Code/LDF` ページを開き (`Project -> Project Options`)、`Leave the files in the project, but stop regenerating them` オプションを選択します。また、`Remove the generated LDF` チェック・ボックスも選択します。`OK` ボタンをクリックして選択を確認します。

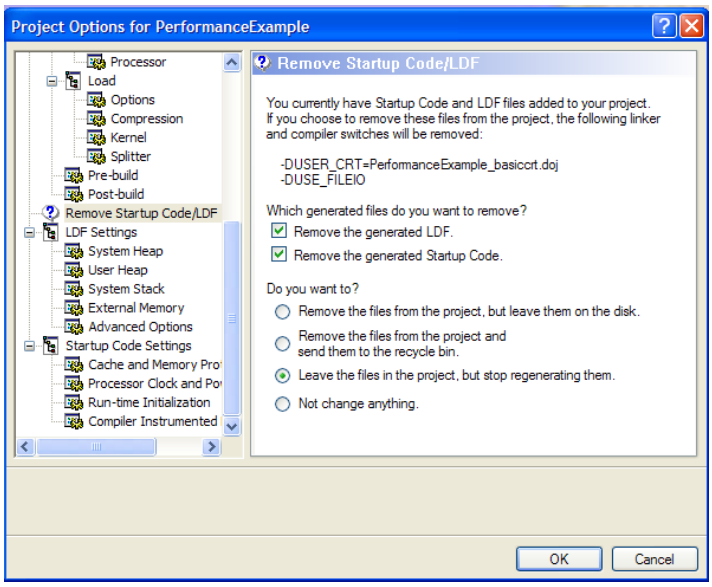


図43.Project Options ダイアログ・ボックスの設定

ここで、リンカー・ディスクリプション・ファイル(.LDF)を変更できるようになります。

ADSP-BF53x プロセッサのページ

ADSP-BF53x プロセッサは、16 ビット・メモリ・インターフェースを持っています。アドレス・マッピング方式を 図 44に定義します。

Bank	Row Address	Column Address	Byte
2 MSB	[(EBCAW+[10,16]):(EBCAW+1)]	[EBCAW:1]	0

図44.ADSP-BF53x のメモリ・マッピング方式

使用するSDRAMに応じてさまざまなページ・サイズがあります。表 2に、EBCAW ビットに対する 16 ビット EBIU のページ・サイズを示します。

EBCAW	Page bytes in Hex
8 bits	0x200
9 bits	0x400
10 bits	0x800
11 bits	0x1000

表2.16 ビット EBIU のページ・サイズ

列アドレス幅が 10 ビットの場合について考えます。各ページは 0x800 バイトで、メモリ・マップを .LDF ファイルで定義することができます(リスト 9)。

```
MEMORY
{...
/* We define 10 pages in bank 0 of a 10-bit memory of a BF53x
SDRAM_BANK_0_PAGE_0       { TYPE(RAM) START(0x00000000) END(0x000007FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_1       { TYPE(RAM) START(0x00000800) END(0x00000FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_2       { TYPE(RAM) START(0x00001000) END(0x000017FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_3       { TYPE(RAM) START(0x00001800) END(0x00001FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_4       { TYPE(RAM) START(0x00002000) END(0x000027FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_5       { TYPE(RAM) START(0x00002800) END(0x00002FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_6       { TYPE(RAM) START(0x00003000) END(0x000037FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_7       { TYPE(RAM) START(0x00003800) END(0x00003FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_8       { TYPE(RAM) START(0x00004000) END(0x000047FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_9       { TYPE(RAM) START(0x00004800) END(0x00004FFF) WIDTH(8) }

//if we would define a section for each page we have to define 8192...
//so we define sections only for the amount of pages which are performance
//relevant
SDRAM_BANK_0_OTHER{ TYPE(RAM) START(0x00005000) END(0x00FFFFFF) WIDTH(8) }
/* the pages on the second bank… */
SDRAM_BANK_1_PAGE_0       { TYPE(RAM) START(0x01000000) END(0x010007FF) WIDTH(8) }
SDRAM_BANK_1_PAGE_1       { TYPE(RAM) START(0x01000800) END(0x01000FFF) WIDTH(8) }
SDRAM_BANK_1_PAGE_2       { TYPE(RAM) START(0x01001000) END(0x010017FF) WIDTH(8) }
SDRAM_BANK_1_PAGE_3       { TYPE(RAM) START(0x01001800) END(0x01001FFF) WIDTH(8) }
SDRAM_BANK_1_PAGE_4       { TYPE(RAM) START(0x01002000) END(0x010027FF) WIDTH(8) }
SDRAM_BANK_1_PAGE_5       { TYPE(RAM) START(0x01002800) END(0x01002FFF) WIDTH(8) }
SDRAM_BANK_1_PAGE_6       { TYPE(RAM) START(0x01003000) END(0x010037FF) WIDTH(8) }
...
}
PROCESSOR p0
{
SECTIONS
    {    ...
         sdram0_page_0
        {
            INPUT_SECTION_ALIGN(4)
             INPUT_SECTIONS($OBJECTS(sdram0page0) )
        } > SDRAM_BANK_0_PAGE_0
    ...
}
}
```

リスト 9.メモリのページへの分割

ADSP-BF561 プロセッサのページ

ADSP-BF561 プロセッサは、32 ビット・メモリ・インターフェースを持っています。16 ビット・インターフェースを使用する場合は、アドレシングはADSP-BF53x プロセッサと同じです。ADSP-BF561 プロセッサの 32 ビット・アドレス・マッピング方式を 図 45に定義します。

Bank	Row Address	Column Address	Byte
2 MSB	[(EBCAW+[11,17]):(EBCAW+2)]	[(EBCAW+1):2]	1-0

図45.ADSP-BF561 のメモリ・マッピング方式

表 3 に、EBCAW ビットに対する 32 ビット EBIUのページ・サイズを示します。

EBCAW	Page bytes in Hex
8 bits	0x400
9 bits	0x800
10 bits	0x1000
11 bits	0x2000

表3.32 ビット EBIU のページ・サイズ

ページを別々にアクセスする利点は何でしょうか?ページ内に留まることを保証できる場合は、さらにプリチャージ・コマンドとアクティベート・コマンドを必要としないので、時間を節約できます。

図 46 に、ページ切り替えを回避するペリフェラル DMA 手法を示します。DMAは着信データを異なるバンクのページへ書き込みます。

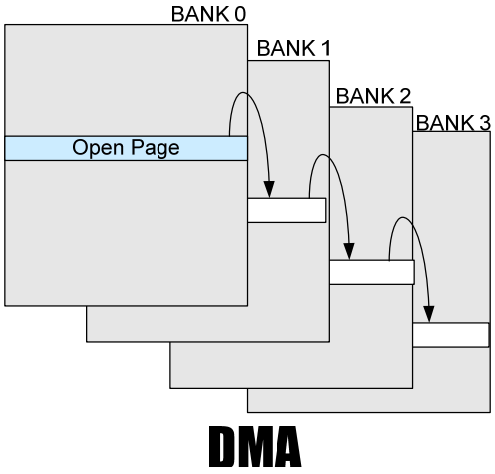


図46.オープン・ページ DMA 手法

6.3 コア・アクセスの SDRAM 性能項目

コア・アクセスは、最も性能にクリティカルな SDRAM アクセスです。したがって、何かの理由でデータ・キャッシュを使用しない場合には、SDRAM へのアクセスを構成する方針を決める必要があります。このセクションでは、このボトルネックの理由を説明しこれに対する対策を説明します。

バッファは、システム・クロック・ドメインとコア・クロック・ドメインとの間のデータ転送を処理するために使われます。これらのバッファはステート・マシンとして構成され、これを使うと、内部処理を気にしないでコア・クロックとシステム・クロックを変更できるようになります。コア/システム・クロック比に応じて、コアとシステム・ドメインとの間の読み出しと書き込みを構成するために待ち状態が導入されます。

この問題を解決する 1 つの手法は、DMA 転送を使用することです。このため、処理対象のデータを内部メモリへ転送するために DMA を使います。DMA には内部 FIFO バッファ構造体があるため、追加時間なしに SDRAM から読み出すことができます。ソフトウェアのプランニングとデザインは、非常に大きな多次元配列(配列サイズ >> 内部データ・メモリ)を扱うときには特に不可欠です。 この場合、使用するアルゴリズムでは、さまざまな方法でこの配列をアクセスする必要があります。DMA を使ってアルゴリズムに最適となるように配列を指図することができます。

6.3.1 コード・オーバーレイ

キャッシュ使用の欠点は、ユーザのプログラムに合わないプログラム・フローをキャッシュから要求されることがあることです。このためにキャッシュ・ミスが頻繁に発生してしまうようになります。もう 1 つの手法は、必要なコードを DMA 転送を使って内部メモリへロードする方法です。コードは、次に必要とされるコードを予測できるインテリジェントなオーバーレイ方式により構成されます。オーバーレイ方式による性能改善の有無は、ユーザのシステム・デザインとプログラム・フローに大きく依存します。大部分のアプリケーションでは、キャッシュの最適化の方がシステム性能を改善する優れた容易な手法です。最初の問題は、比較的独立していて、相互に直接呼び出しを必要としないプログラムのモジュールを特定することです。プログラムからオーバーレイを分離して、そのマシン・コードを大きなメモリへ配置します。SDRAM から内部メモリへオーバーレイのインテリジェントな DMA 転送を構成するオーバーレイ・マネージャを制定します。

6.4 キャッシュを使用するときの SDRAM 性能項目

コード

キャッシュ・アクセスに対してメモリを最適化することは、キャッシュ・ミスを減らすことを意味します。キャッシュ・ミスを最小にする方法でコードとデータを構成する必要があります。キャッシュ・アクセスに対してコードを最適化するときは、コードをできるだけ単調にして、インラインとしてプログラム・コード内で頻繁に使用されない関数を宣言します。これにより、コードが小さく維持されてキャッシュ・ミス数が少なくなります。頻繁に呼び出される関数は、可能な場合、内部メモリに配置する必要があります。

データ

アルゴリズムを調べて、シーケンシャルなデータ・アクセスを行う方法を探します。データをシーケンシャルにアクセスする高速フーリェ変換 (FFT) の例が、「*Writing Efficient Floating-Point FFTs for ADSP-TS201 TigerSHARC® Processors (EE-218)*[8]」に示してあります。C 言語では、多次元配列はメモリ内で次の順序で並びます (ここでは 3 次元配列)。

$A_{0,0,0}$、$A_{0,0,1}$、$A_{0,0,2}$、$A_{0,1,0}$、$A_{0,1,1}$、$A_{0,1,2}$、$A_{0,2,0}$、$A_{0,2,1}$、$A_{0,2,'2}$、$A_{1,0,0}$、$A_{1,0,1}$、$A_{1,0,2}$、$A_{1,1,0}$、$A_{1,1,1}$、$A_{1,1,2}$、$A_{1,2,0}$、$A_{1,2,1}$、$A_{1,2,'2}$、$A_{2,0,0}$、$A_{2,0,1}$、$A_{2,0,2}$、$A_{2,1,0}$、$A_{2,1,1}$、$A_{2,1,2}$、$A_{2,2,0}$、$A_{2,2,1}$、$A_{2,2,'2}$

リスト 10 に、シーケンシャルなアクセスで配列に書き込みを行うコードを示します。

```
    for (i=0;i<3; i++)
        {
         for (j=0;j<3; j++)
         {
              for (k=0;k<3; k++)
                 { MyArray[i][j][k] = getValue();
                 }
         }
        };
```

リスト 10.配列の最適なアクセス

7 消費電力の最適化

Blackfin プロセッサは、携帯型アプリケーションまたは低消費電力アプリケーションで良く使用されます。このようなアプリケーションでは、消費電力をできるだけ小さくすることが不可欠です。SDRAM は高いパーセント値で消費電力を増やします。このため、SDRAM の選択、使い方、設定は低消費電力デザインの基礎になります。このセクションでは、消費電力を削減する方法の概要を説明します。

7.1 消費電力の数値

デバイス・メーカは種々の標準シンボル (表 4) を使用して消費電力を定めていますが、これらの数値を測定する手順は異なります。このため、これらの値の解釈に違いがあります。

Symbol	Meaning
I_{CC1}	Operating current in active mode
I_{CC2}	Precharge standby current
I_{CC3}	No operating/ standby current
I_{CC4}	Operating current in burst mode / all banks activated
I_{CC5}	Auto-refresh current
I_{CC6}	Self-refresh current

表4.消費電力測定値のシンボル

7.2 SDRAM 消費電力を削減するこつ

SDRAM 消費電力を削減するこつを次に示します。

- 使用する SDRAM 数をできるだけ少なくします。
- 1.8Vまたは 2.5V のモバイル SDRAMを使います (すべての Blackfin プロセッサで使用可能、SDRAMの概要を参照)。
- 低温環境ではリフレッシュ・レートを低くします。ワーストケース(高温)で 64 ms のリフレッシュ・レートが規定されています。このため、標準環境で動作させる場合にはまだ低下させる余裕があります。
- メモリ・バンク間 (メモリ・バンク内ではなく)でデータ転送を行います。
- EBIU_SDGCTL レジスタのセルフ・リフレッシュ・ビット (SRFS)をイネーブルします。このモードで、SDRAM の消費電力は最小になります。

7.3 モバイル SDRAM

消費電力要求が大きい組込み型アプリケーションに対してはモバイル SDRAM または低消費電力 SDRAM を使用してください。これらのアプリケーションでは、SDRAM はほとんど使用されず、大部分の時間はアイドル状態にあります。モバイル SDRAM は、SDRAM がアイドル・モードにあるとき消費電力を削減する特別なモードを提供しています。Blackfin プロセッサでサポートしている機能は、温度補償されたセルフ・リフレッシュ (TCSR) とパーシャル・アレイ・セルフ・リフレッシュです。温度補償されたセルフ・リフレッシュ機能を使うと、45°C 以下の温度でアイドル状態にあるときセルフ・リフレッシュ周波数を下げることができます。メモリ・セルのリークは温度に強く依存します。温度が高いほど、リークが大きくなります。標準 SDRAM のセルフ・リフレッシュ・レートは、高温でワーストケース値に設定され、SDRAM の仕様が規定されています。Blackfin プロセッサでは、`EBIU_SDGCTL` レジスタの `TCSR` ビットを使って温度を設定することができます。`TCSR` の値は温度境界を指定します (たとえば 45°C は 45°C を下回る動作を意味します)。

多くのアプリケーションでは、大部分の SDRAM デバイスをデータ配列のバッファ(処理後)に使用します。 これらのアプリケーションでは、パーシャル・アレイ・セルフ・リフレッシュ機能が省電力のために優れた手法になっています。この機能を使うと、アプリケーションからアイドル・モード時にリフレッシュするメモリ・バンクを選択できるようになります。SDRAM 上にコードを配置する場合は、SDRAM のバンク 0 とバンク 1 に配置してください。そうしないと、コードが失われてしまいます。

このアプリケーションでは、低電圧 (1.8V または 2.5V) Blackfin プロセッサを使用してください。

7.4 ハイバネートの開始と回復

ADSP-BF537、ADSP-BF54x、ADSP-BF52x の各プロセッサは、プロセッサがハイバネートモードになっても SDRAM の値を保持しています。このため、`VR_CTL` レジスタの `CKELOW` ビットに 1 を設定して、ハイバネート中 `CKE` 信号をロー・レベルに維持する必要があります。これにより、SDRAM のデータが失われるのが防止されます。内部メモリとすべてのレジスタ(ただし `VR_CTL` レジスタは除く) の値は失われます。このため、プログラムはすべてのレジスタ設定値および内部メモリ値を SDRAM へ書き込む必要があります。

ハイバネートモードにし、その後データを回復するため、次のステップが使われます。

ステップ 1

すべての重要レジスタを SDRAM に待避させます。

内部メモリの重要データを SDRAM に待避させます。

`VR_CTL` レジスタの `CKELOW` ビットと `SDGCTL` レジスタのセルフ・リフレッシュ・ビットがセットされていることを確認します。

プロセッサをハイバネートモードにします (図 47と リスト 11)。

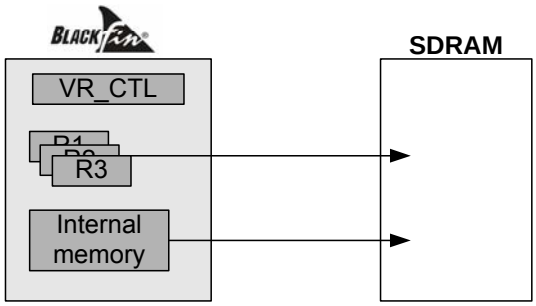


図47.ハイバネートモードの開始

```
    SaveTheMemory();
    SaveTheRegister();
    //Let's setup the RTC to wake up
    SetupRTC();
    IntMask = cli();
    *pVR_CTL = (       WAKE    | //wake by
                       CKELOW  | //keeps the content of the SDRAM
                       CANWE   | //ensures that the CAN RX can wake up the BF
                       (*pVR_CTL & ~FREQ)); // Send to hibernate
```

リスト 11.ハイバネートモードの開始

ステップ 2

プロセッサはハイバネートモードになります (図 48)。VR_CTL 以外のすべてのレジスタの値は失われます。データはSDRAMに保存されます。

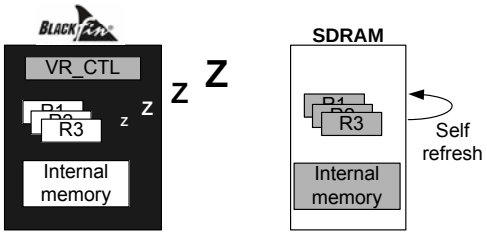


図48.ハイバネートモード

ステップ 3

ハイバネートモードからプロセッサがウェイクアップすると (図 49)、プロセッサは初期化ファイルからブートします。プロセッサはCKELOW ビットをチェックして、ハイバネートまたはリセットのいずれから抜け出たのかを調べます。Blackfin プロセッサがリセットから抜け出た場合は、プロセッサはブート処理を続け、その他の場合には、内部メモリとレジスタを回復するルーチンを呼び出して、実行コードへ分岐します。

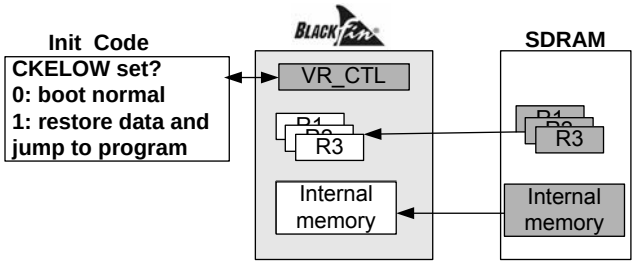


図49.ハイバネートモードからの回復

7.5 低消費電力用のデータ構成

プリチャージ動作とアクティベート動作は時間がかかるだけでなく、消費電力にも大きな影響を与えます。このため、これらの動作回数をできるだけ削減してください。

アプリケーション内のプリチャージ動作は、ページ・ブレークとSDRAM リフレッシュで実行されます。したがって、ページ・ブレーク回数ができるだけ少なくなる方法でデータを構成してください。例えば、システムのSDRAM 性能の強化 を参照してください。

補足

補足 A: 用語集

アクセス・タイム	1 つのデバイス・アクセスの開始から次のアクセスが開始できるまでの時間。
配列	データ格納用のメモリ領域。配列は行と列から構成され、アドレスが交差する位置に各メモリ・セルが存在します。メモリ内の各ビット、行と列の座標で検索されます。
非同期	独立に動作が進行する処理。
自動プリチャージ	バースト動作の後にページを閉じる SDRAM の機能。
自動リフレッシュ	内部発振器がリフレッシュ・レートを決定し、内部カウンタがリフレッシュするアドレスを記録しているモード。
バンク	バンクは、SDRAM モジュール上の物理バンク数 (行数と同じ) を意味します。各 SDRAM デバイス内の内部論理バンク数 (今日は通常 4 バンク)を意味することもあります。
バースト・モード	バーストは、一連のメモリ・セル・ロケーションへのデータの急速転送を意味します。
バイパス・コンデンサ	特に隣接するデバイスまたは回路の電源電圧の安定化を主な機能とするコンデンサ。
バス・サイクル	メモリ・デバイスと Blackfin プロセッサのシステム・ドメインとの間のシングル・トランザクション。
CAS	列アドレス・ストローブ。列アドレスを SDRAM コントロール・レジスタへラッチするコントロール信号。
CAS-before-RAS (CBR)	行アドレス・ストローブの前の列アドレス・ストローブ。CBR はリフレッシュする次の行を記録する高速なリフレッシュ機能です。
列	メモリ配列の一部分。ビットは行と列が交差する位置に保存されます。
クロストーク	別の配線またはパターンの電流によって 1 つの配線またはパターンに誘導される信号。
DDR	ダブル・データ・レート。データが、クロックの立ち上がりエッジと立ち下がりエッジで転送されます。アドレス・ラインが同じ状態を維持するため、シーケンシャルなアドレスのデータが転送されます。
DQM	書き込みサイクルでマスクする際に使用されるデータ・マスク信号。8 本の I/O ごとに 1 本の DQM 信号があります。
DRAM	Dynamic Random Access Memory の略。コンピュータ・システムのマス・ストレージとして使用されるメモリ・デバイスの 1 つのタイプ。ダイナミックという用語は、一定のリフレッシュによりメモリがデータを保持することを意味しています。
EBIU	External Bus Interface Unit の略。主に PC100 と PC133 規格に準拠する SDRAM に対する同期外部メモリ・インターフェースと、SRAM、ROM、FIFO、フラッシュ・メモリ、FPGA/ASIC デザインに対する非同期インターフェースを提供します。
EMC	EMI を制御する規制に対する適合性。

EMI	電磁放射による干渉。
外部バッファ	負荷が 50 pF を超える場合に、外部バッファは SDRAM コマンド、クロック、クロック・イネーブル、アドレス・ピンを駆動するために必要です。この容量は、入力ピン数と SDRAM 入力ピンの容量の積 (SDRAM の 1 本の入力ピンには約 4.5 pF があります [SDRAM データ・シートを参照])に PCB パターンの容量を加算した値になります。
FBBRW	Fast back-to-back read-to-write の略。標準的なアプリケーションでは、書き込みコマンドが読み出しコマンドが処理された後に 1 クロック・サイクル遅延させられます。**FBBRW** では、書き込みが読み出しコマンドの処理後 1 サイクルの遅延なしで直ちに実行されます。データの読み出しとデータの書き込みの間にデータ・バスを迅速に切り替える必要があるため、この機能はデータ・バスの容量に強く依存します。これには、SDRAM チップ数とデータ・バス回路パスのデザインが含まれます。
FPM	高速ページ・モード。一般的な SDRAM データ・アクセス方式。
ハイバネート	最小の消費電力を提供する、Blackfin プロセッサの特別な電源モード。I/O 電源は維持されますが、コアはパワーダウンします。Blackfin プロセッサは複数の外部イベントでウェイクアップすることができます。
インターリービング	SDRAM 内の複数のページからデータの読み出し/書き込みを交互に行う処理。
JEDEC	Joint Electron Device Engineering Council の略。
遅延	メモリ・ロケーションに対する読み出し要求から実際にデータが読み出されるまでの時間で、クロック・サイクル数で表されます。
メモリ・サイクル・タイム	メモリ動作全体(読み出しまたは書き込み)に要する時間。
マイクロストリップ	信号パターンの片面にのみリファレンス・プレーンが存在するパターン構成。
OTP メモリ	One-Time-Programmable Memory の略。特にセットアップ値と Blackfin セキュリティ方式のキーを保存するために使用する内蔵メモリ。
ページ	1 つの行アドレスでアクセスできるバイト数。
ページ・モード	RAS がロー・レベルになり、かつ列アドレスがストローブされたときに発生する動作。 SDRAM は直前の行アドレスを記憶しているためその行に留まり、新しい列アドレスへ移動します。
PCB	Printed circuit board の略。
RAS	行アドレス・ストローブ。行アドレスを SDRAM へラッチするコントロール信号。列アドレスと組み合わせて使い、各メモリ・ロケーションを選択します。
RAS—CAS 間遅延	行アクセス・ストローブと列アドレス・ストローブとの間の時間。
読み出し時間	行アドレスと列アドレスが有効になった後にデータが出力に現れるまでに要する時間。読み出し時間はアクセス・タイムとも呼ばれます。
リフレッシュ	データを保持するために必要な SDRAM セル電荷の周期的な再生。
リフレッシュ・サイクル	SDRAM の 1 行のリフレッシュを行う周期。
リフレッシュ周期	SDRAM 内の各行がリフレッシュされる必要がある最小の時間。
行	メモリ配列の一部分。ビットは行と列が交差する位置に保存されます。

SDR	Single data rate の略。データが 1 クロック・サイクルで転送されます。この定義は、この SDRAM を DDR から区別するために使用されています。
SDRAM セルフ・リフレッシュ	通常の動作モードでは、自動リフレッシュ・コマンドを送信することにより、Blackfin プロセッサがデータ・セルのリフレッシュを制御しますが、アプリケーションが SDRAM の制御を諦める必要がある場合(例えば、プロセッサがハイバネートモードを開始する場合またはマルチプロセッサ・アプリケーションの場合)、SDRAM はデータの一貫性を維持する役割を受け継ぐ必要があります。SDRAM をセルフ・リフレッシュにするもう 1 つのケースは、消費電力を削減するときです。Blackfin プロセッサがセルフ・リフレッシュ・コマンドを送信すると、SDRAM は自分のクロックを使ってセルフ・リフレッシュ・サイクルを実行します。欠点は、セルフ・リフレッシュで SDRAM をアクセスするときに発生する遅延です。
ストローブ	データを同期して SDRAM へラッチする入力コントロール信号。
ストリップライン	信号パターンが 2 つのリファレンス・プレーン間に配置された回路ボード構成。
同期メモリ	リファレンス・クロックに同期させた信号を持つメモリ・デバイス。
終端	信号反射を制御するために伝送ラインと組み合わせて使用される 1 個または複数の部品。
書き込み時間	データが SDRAM にラッチされてから実際にメモリ・ロケーションへ保存されるまでの時間。

補足 B: コード例、回路図、解説

初期化コード (Chapter 2)

リスト 12 に、アセンブリ言語による初期化の例を示します。

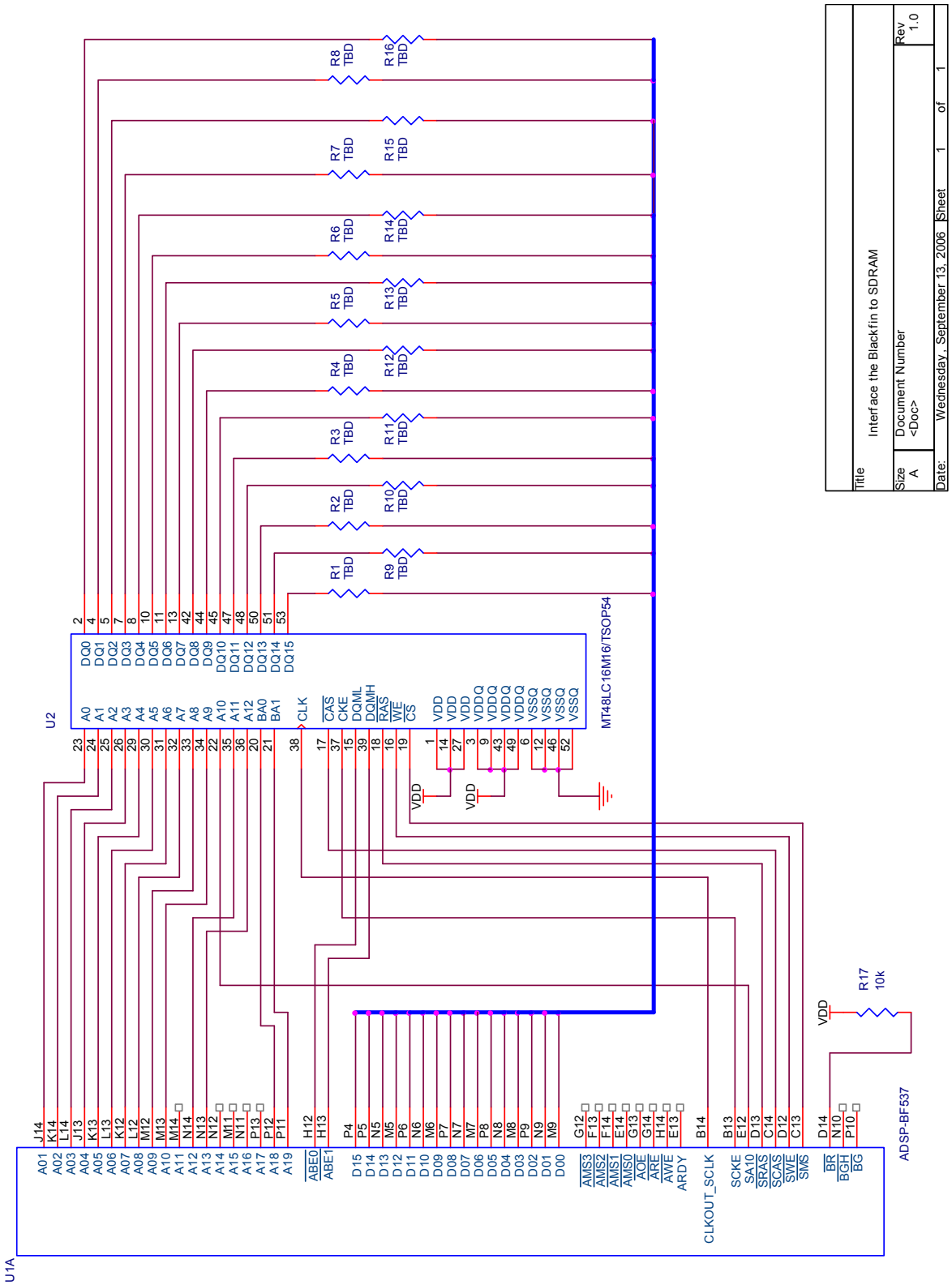
```
//SDRAM Refresh Rate Setting
    P0.H = hi(EBIU_SDRRC);
    P0.L = lo(EBIU_SDRRC);
    R0 = 0x406 (z);
    w[P0] = R0;
//SDRAM Memory Bank Control Register
    P0.H = hi(EBIU_SDBCTL);
    P0.L = lo(EBIU_SDBCTL);
    R0 =          EBCAW_9   | //Page size 512
                  EBSZ_64   | //64 MB of SDRAM
                  EBE;        //SDRAM enable
    w[P0] = R0;
//SDRAM Memory Global Control Register
    P0.H = hi(EBIU_SDGCTL);
    P0.L = lo(EBIU_SDGCTL);
    R0.H=         hi(    ~CDDBG     & // Control disable during bus grant off
                         ~FBBRW     & // Fast back to back read to write off
                         ~EBUFE     & // External buffering enabled off
                         ~SRFS      & // Self-refresh setting off
                         ~PSM       & // Powerup sequence mode (PSM) first
                         ~PUPSD     & // Powerup start delay (PUPSD) off
                         TCSR       | // Temperature compensated self-refresh at 85
                         EMREN      | // Extended mode register enabled on
                         PSS        | // Powerup sequence start enable (PSSE) on
                         TWR_2      | // Write to precharge delay TWR = 2 (14-15 ns)
                         TRCD_3     | // RAS to CAS delay TRCD =3 (15-20ns)
                         TRP_3      | // Bank precharge delay TRP = 2 (15-20ns)
                         TRAS_6     | // Bank activate command delay TRAS = 4
                         PASR_B0    | // Partial array self refresh Only SDRAM Bank0
                         CL_3       | // CAS latency
                         SCTLE      ); // SDRAM clock enable

    R0.L=         lo(    ~CDDBG     & // Control disable during bus grant off
                         ~FBBRW     & // Fast back to back read to write off
                         ~EBUFE     & // External buffering enabled off
                         ~SRFS      & // Self-refresh setting off
                         ~PSM       & // Powerup sequence mode (PSM) first
                         ~PUPSD     & // Powerup start delay (PUPSD) off
                         TCSR       | // Temperature compensated self-refresh at 85
                         EMREN      | // Extended mode register enabled on
                         PSS        | // Powerup sequence start enable (PSSE) on
                         TWR_2      | // Write to precharge delay TWR = 2 (14-15 ns)
                         TRCD_3     | // RAS to CAS delay TRCD =3 (15-20ns)
                         TRP_3      | // Bank precharge delay TRP = 2 (15-20ns)
                         TRAS_6     | // Bank activate command delay TRAS = 4
                         PASR_B0    | // Partial array self refresh Only SDRAM Bank0
                         CL_3       | // CAS latency
                         SCTLE    ) ; // SDRAM clock enable
    [P0] = R0;
```

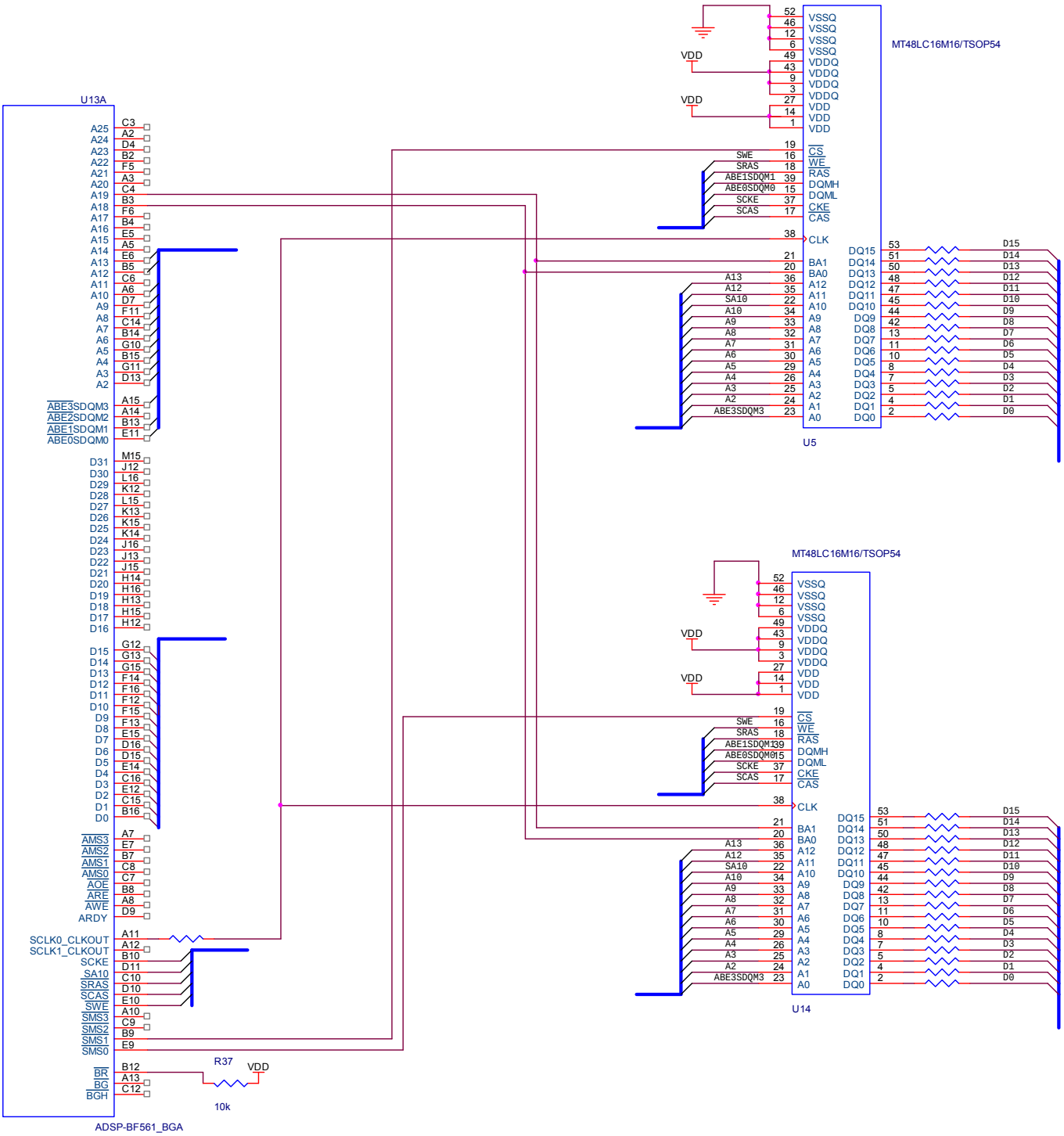
リスト 12.アセンブリ言語による初期化の例

SDRAM と Blackfin プロセッサとのインターフェースの回路図 (Chapter 4)

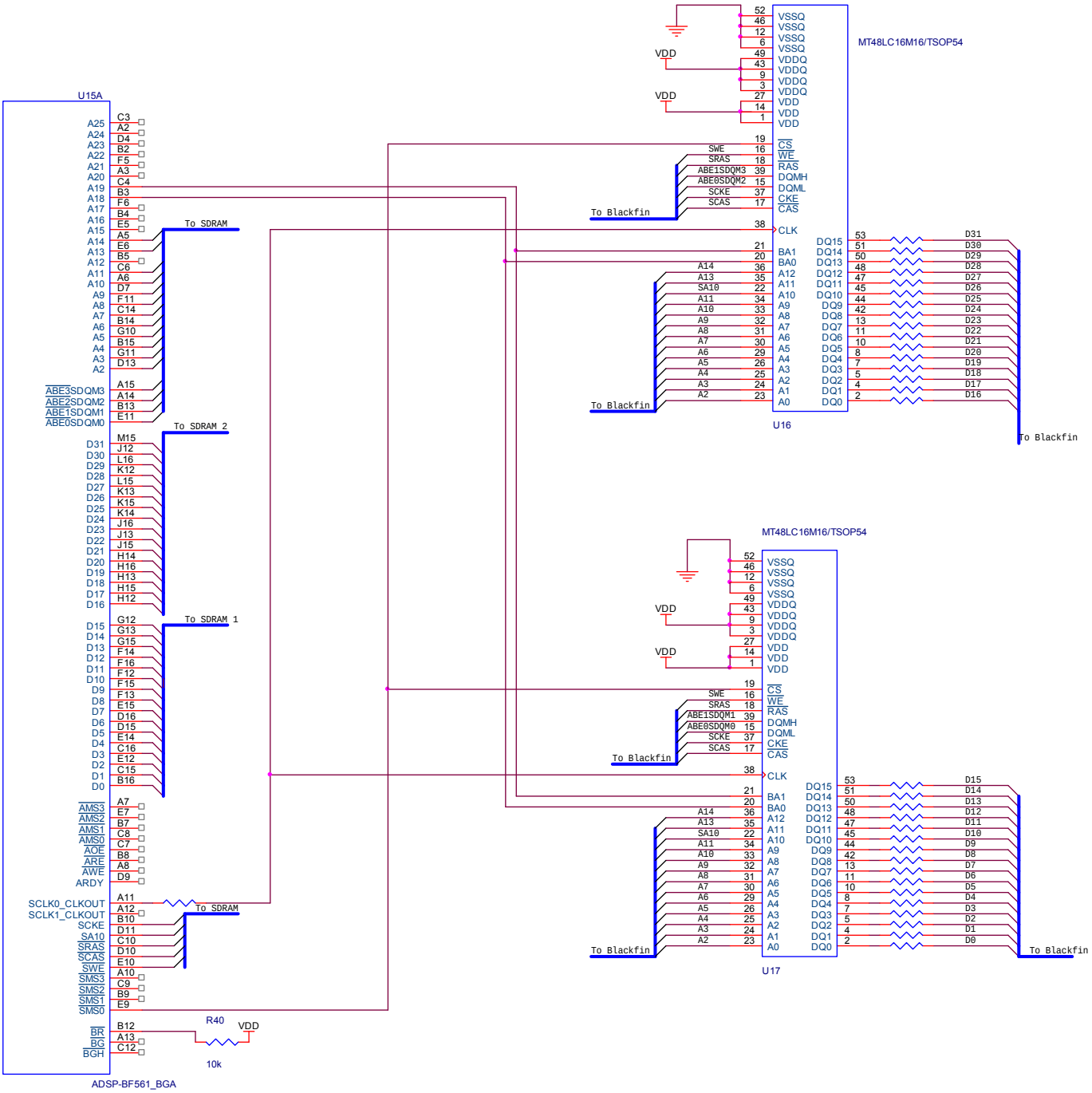
次の数ページに実施例を示します。これらの回路図は、SDRAM と Blackfin プロセッサとの間の正しい接続 (信号インテグリティの手法としてではなく)を示すためのものです。



16 ビット・モードのADSP-BF561



32 ビット・モードのADSP-BF561



解説: Z_0 の計算 (Chapter 4)

このセクションでは Telegrapher の式による方法を説明します。

SDRAM 接続は伝送ラインです。これが、構造に沿った電流と電圧の伝搬を正確にモデル化するtelegrapherの式を使う理由です。パターンは理想的な導体でないため、パターン自体の影響を含む等価回路図 (図 50)を使う必要があります。

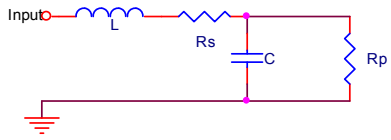


図50.パターン自体の影響を含む等価回路

ただし、パターンはこれらの構造の 1 つから構成されるだけではありません。接続ライン (図 51) をこれらのほぼ無限の接続と見なすことができます。

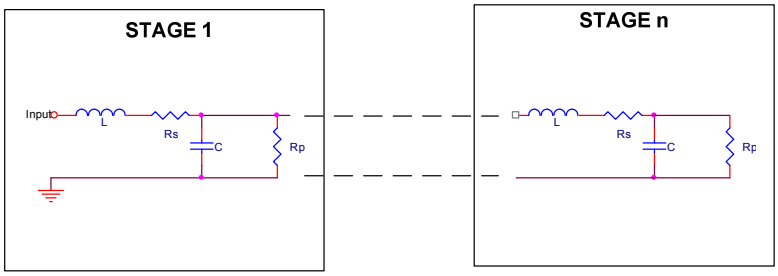


図51.無限のパターンで構成される接続ライン

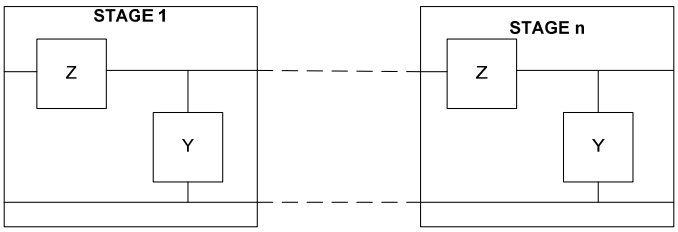
変数 Z_C は、インピーダンスの関数で周波数に依存します。数学的に正しく書く場合は、 $Z_C(\omega)$と書く必要があります。

変数 Z_0は、特定の周波数ω_0での特性インピーダンス値を表す定数です。

次のようにインピーダンス Y と Z を定義してモデルを簡素化します。

$Z = j\omega L+Rs$

$Y = j\omega C+1/Rp$



この結果得られるインピーダンスは、インピーダンス Z、インピーダンス Y、並列なすべての他のステージの和です。ここで、同じ Rp、Rs、C、L を持つ n 個の エレメントを持つとします。

$$\tilde{Z}_c = \frac{Z}{n} + \frac{1}{\frac{1}{\tilde{Z}_c} + \frac{Y}{n}}$$

両辺を $(1+\frac{Y}{n}\tilde{Z}_c)$ 倍すると、

$$\tilde{Z}_c(1+\frac{Y}{n}\tilde{Z}_c)=\frac{Z}{n}(1+\frac{Y}{n}\tilde{Z}_c)+\tilde{Z}_c \Leftrightarrow$$

$$\tilde{Z}_c{}^2=\frac{Z}{Y}+\frac{Z}{n}\tilde{Z}_c \Leftrightarrow$$

$$\tilde{Z}_c=\sqrt{\frac{Z}{Y}+\frac{Z}{n}\tilde{Z}_c}$$

伝送ラインは長さ 0 の無限エレメントで構成されるとすると、

$$Z_c=\lim_{n\to\infty}\sqrt{\frac{Z}{Y}+\frac{Z}{n}\tilde{Z}_c}=\sqrt{\frac{Z}{Y}}=\sqrt{\frac{j\omega L+Rs}{j\omega X+1/Rp}}$$

周波数を上げると、R と G の項はそれぞれ jω L と jω C に比べて実質的に無視できるため、インピーダンスは平坦になります。誘導性インピーダンス jω L と容量性アドミタンス jω C の間でバランスさせると、高周波でインピーダンスを一定にすることができます。この一定のインピーダンス平坦性は、伝送ラインの終端が 1 本の抵抗で可能になるため、高速デジタル回路のデザインに大いに役立ちます。この平坦部での特性インピーダンス値は Z0 と呼ばれます。

$$Z_0=\lim_{\omega\to\infty}(Z_C(\omega))=\sqrt{\frac{L}{C}}$$

マイクロトリップ・パターンのインピーダンスの計算

これらの値を測定することが推奨されますが、次の式を使って値を計算することができます。

$$L\cong 5\cdot\ln\left(\frac{2\pi\cdot h}{w}\right)$$

ここで、

L は nH/インチで表したインダクタンス

h はプレーンからの高さ (mils)

w はパターンの幅 (mils)

容量の計算

$$C=\varepsilon_0 * \varepsilon_r * \frac{A}{d}$$

ここで、

A はパターンの長と幅の積

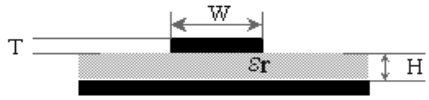
d はグラウンドとパターン間の距離

これは、各グラウンド・プレーンに対して計算する必要があります。

Z_0についてのIPCとDouglas Brooksの手法 (Chapter 4)

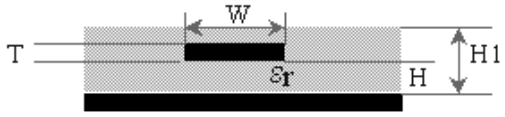
IPC と Douglas Brooks は、使い易い PCB に対する幾つかの式を提供しています。

マイクロトリップ・パターン



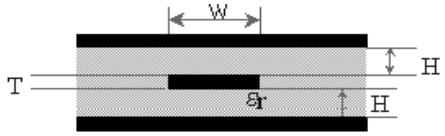
$$Z_0 = \frac{87}{\sqrt{\varepsilon_r + 1.41}} \ln\left(\frac{5.98 \cdot H}{0.8 \cdot W + T}\right)$$

埋め込みマイクロトリップ・パターン



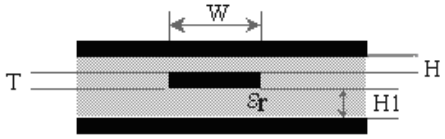
$$Z_0 = \frac{60}{\sqrt{\varepsilon_r\left[1 - e^{\frac{(-1.55 \cdot H1)}{H}}\right]}} \ln\left(\frac{5.98 \cdot H}{0.8 \cdot W + T}\right)$$

ストリップライン・パターン



$$Z_0 = \frac{60}{\sqrt{\varepsilon_r}} \ln\left(\frac{1.9 \cdot (2H + T)}{0.8 \cdot W + T}\right)$$

非対称ストリップライン・パターン



$$Z_0 = \frac{80}{\sqrt{\varepsilon_r}} \ln\left(\frac{1.9 \cdot (2H + T)}{0.8 \cdot W + T}\right)\left(1 - \frac{H}{4 \cdot H1}\right)$$

ここで、H1 > H

プリント回路ボード (PCB)の誘電率 (Chapter 4)

次の表は「*A Survey and Tutorial of Dielectric Materials Used in the Manufacture of Printed Circuit Boards*[7]」から得たものです。

市販のガラス織強化ラミネート

Material	Tg	e_{r*}	Tan (f)	DBV (V/mil)	WA, %
Standard FR-4 Epoxy Glass	125C	4.1	0.02	1100	0.14
Multifunctional FR-4	145C	4.1	0.022	1050	0.13
Tetra Functional FR-4	150C	4.1	0.022	1050	0.13
Nelco N4000-6	170C	4	0.012	1300	0.10
GETEK	180C	3.9	0.008	1100	0.12
BT Epoxy Glass	185C	4.1	0.023	1350	0.20
Cyanate Ester	245C	3.8	0.005	800	0.70
Polyimide Glass	285C	4.1	0.015	1200	0.43
Teflon	N/A	2.2	0.0002	450	0.01
		* Measured with a TDR using velocity method. Resin content 55%			

Tg = glass transition temperature
er = relative dielectric constant
Tan (f) = loss tangent
DBV = dielectric breakdown voltage
WA = water absorption

All materials with woven glass reinforcement except teflon.

非織またはガラス成分が非常に少ないラミネート材料のリスト

Material	Tg	e_{r*}	Tan (f)	DBV (V/mil)	WA, %
Speedboard N	140C	3	0.02	N/A	N/A
Speedboard C	220C	2.7	0.004	N/A	N/A
Rogers Ultralam C	280C	2.5	0.0019	N/A	N/A
Rogers 5000	280C	2.3	0.001	N/A	N/A
Rogers 6002	350C	3	0.0012	N/A	N/A
Rogers 6006	325C	6 to 10	0.002	N/A	N/A
Rogers RO3003	350C	3	0.0013	N/A	N/A
Rogers RO3006	325C	6 to 10	0.003	N/A	N/A
Teflon	N/A	2.2	0.0002	450	0.01
		Information from manufacturer's data sheets.			

Tg = glass transition temperature
er = relative dielectric constant
DBV = dielectric breakdown voltage
WA = water absorption

参考文献

[1] *The ABCs of SDRAM (EE-126)*, Rev. 1, March 2002. Analog Devices, Inc.

[2] *System Optimization Techniques for Blackfin Processors (EE-324)*, Rev. 1, July 2007. Analog Devices, Inc.

[3] *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 1 of 2) Revision 0.31 (Preliminary).* May, 2008. Analog Devices, Inc.

[4] *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 2 of 2) Revision 0.3 (Preliminary).* September, 2007. Analog Devices, Inc.

[5] *ADSP-BF533 Blackfin Processor Hardware Reference.* Rev 3.2, July 2006. Analog Devices, Inc.

[6] *ADSP-BF537 Blackfin Processor Hardware Reference.* Rev 3.0, December 2007. Analog Devices, Inc.

[7] *ADSP-BF561 Blackfin Processor Hardware Reference.* Rev 1.1, February 2007. Analog Devices, Inc.

[8] *ADSP-BF522/523/524/525/526/527 Blackfin Embedded Processor Preliminary Data Sheet.* Rev PrE, August 2008. Analog Devices, Inc.

[9] *ADSP-BF512/ADSP-BF514/ADSP-BF516/ADSP-BF518 Blackfin Embedded Processor Preliminary Data Sheet.* Rev PrC, October 2008. Analog Devices, Inc.

[10] *ADSP-BF531/ADSP-BF532/ADSP-BF533 Blackfin Embedded Processor Data Sheet.* Rev F, 2008. Analog Devices, Inc.

[11] *ADSP-BF534/ADSP-BF536/ADSP-BF537 Blackfin Embedded Processor Data Sheet.* Rev E, March 2008. Analog Devices, Inc.

[12] *ADSP-BF538/ADSP-BF539 Blackfin Embedded Processor Data Sheet.* Rev B, 2008. Analog Devices, Inc.

[13] *ADSP-BF561 Blackfin Embedded Processor Data Sheet.* Rev C, 2007. Analog Devices, Inc.

[14] *A Survey and Tutorial of Dielectric Materials Used in the Manufacture of Printed Circuit Boards* - By Lee W. Ritchey, Speeding Edge, for publication in November 1999 issue of Circuitree magazine. Copyright held by Lee Ritchey of Speeding Edge, September 1999.

[15] *Writing Efficient Floating-Point FFTs for ADSP-TS201 TigerSHARC® Processors (EE-218)*, Rev. 2, March 2004

[16] *Micron Technical Note 48-09 LVTTL DERATING FOR SDRAM SLEW RATE VIOLATIONS*

[17] *High Speed Digital Design – A Handbook of Black Magic* by Howard W. Johnson and Martin Graham, 1993 PTR Prentice Hall, ISBN 0-13-395724-1

[18] *High-Speed Signal Propagation – Advanced Black Magic* by Howard W. Johnson and Martin Graham, 2003, PTR Prentice Hal, ISBN 0-13-084408-X

[19] *EMV-Design Richtlinien* by Bernd Föste and Stefan Öing, 2003 Franzis' Verlag GmbH, ISBN 3-7723-5499-8

資料

[20] *ADSP-BF53x/ADSP-BF56x Blackfin Processor Programming Reference.*Rev 1.2, February 2007. Analog Devices, Inc.

ドキュメント改訂履歴

Revision	Description
Rev 2 – December 11, 2008 by Fabian Plepp	Provides a more detailed description of the SDRAM initialization. Also, adds information on drive strength control for ADSP-BF52x devices. Incorporates support for ADSP-BF51x processors.
Rev 1 – May 12, 2008 by Fabian Plepp	Initial public release. Adds Low Power section, OTP and ADSP-BF52x processors.
Rev 0 – August 21, 2006 by Fabian Plepp	Internal version (draft).