秋月電子H8/3069F基板実装 D-RAMのアクセス

今回は、秋月電子 H8/3069F USBホスト基板裏側に実装されている 2Mbyte D-RAMの アクセスの実験を行います。 以下の画像の黄色の四角で囲ったTOSHIBAの LSI が 今回のD-RAMです。 データバスは 8bitで H8マイコンの D15 ～ D8 のデータバスに接続されています。 ちなみに RAMは 基本 S-RAMとD-RAMの 2種類が あります。S-RAMは 1bitのデータを記憶するのに フリップフロップ回路を使用しています。フリップフロップ回路は トランジスタを 5個か 6個使用するようで、1bit を構成する用途では、ややトランジスタ数を 多く消費するので、記憶密度を上げる事が出来ません。 それに対し D-RAMは 半導体中に 小容量のコンデンサを構成して 1bitの 構成部品を少なくして 記憶密度を上げています。

但し、コンデンサなので そのまま放置しておくと、電荷が自然放電して、Hi か Low か 分からなくなります。 その関係で 周期的にリフレッシュ動作という Hi か Low かが、ハッキリ分かるようにする処理を 行います。 これにより 中途半端な電位に ならないようにしています。 その関係で 外付けで リフレッシュ回路が必要です。

今回は、H8/3069Fマイコン内部に D-RAMのリフレッシュコントローラが 組み込まれています。 よって別途 リフレッシュコントローラは必要ありません。 但し リフレッシュ動作は、D-RAM内の多数のコンデンサの充電を 瞬時に行うので やや電力を消費します。

そして、そのリフレッシュサイクルが ミリ秒ぐらいの周期で 割り込んでくるので D-RAMのメモリアクセスは 遅くなります。

また、CPUが D-RAMを アクセスする時は、間にバスコントローラが 入って D-RAM特有のアドレス指定である カラムと ローの 2回に分けて書き込むので この動作によっても アクセスタイムが 遅くなります。

D-RAMの短所ばかり、書きましたが、D-RAMの長所は ワンチップでの メモリ容量が 大きい事です。 よってメモリ容量が大きいので、他の短所には 目をつぶってくれ。という事なのでしょう。

因みに S-RAMの短所は 単位面積当たりの記憶容量が D-RAMと比べ 小さい事です。
S-RAMの長所は リフレッシュの必要が無いので、低消費電力、高速アクセスが可能です。

そして、D-RAMも S-RAMも 揮発性のメモリですが、S-RAMは CPUが アクセスしなければ、消費電力は極めて小さいので 電池等で約 3V程度で 電源バックアップが 出来ます。

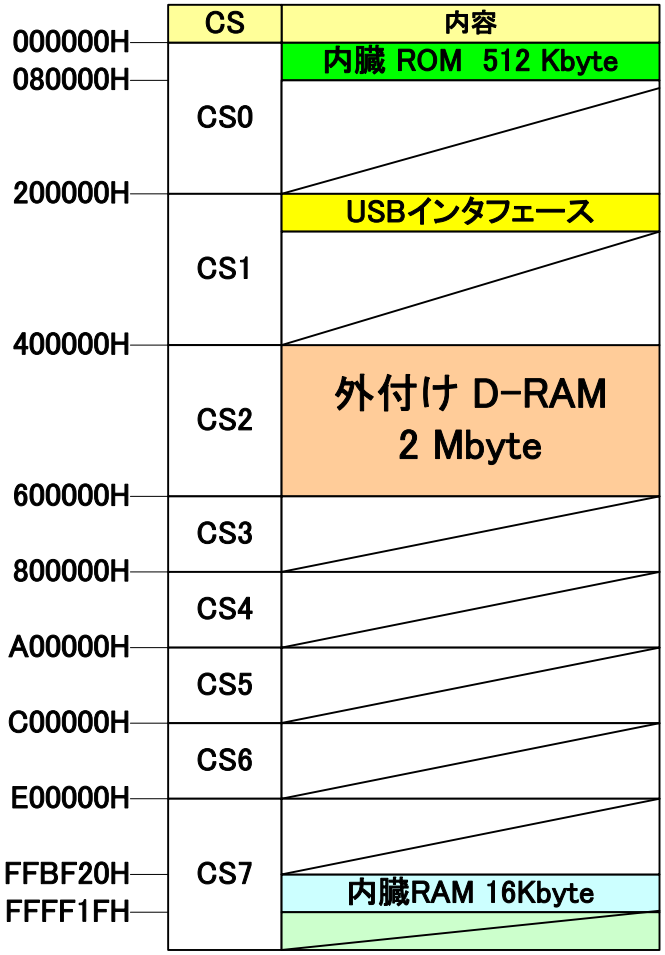
という事で、D-RAMと S-RAMの 違いの説明でした。

AE-3069USB基板のメモリマップ

AE-3069USB基板のメモリマップを示し、基板裏側に実装される外付け D-RAMの　アドレス範囲を　確認します。
　右のメモリマップを見れば分かりますが、D-RAMの配置アドレスは、400000h ～ 5FFFFFh です。　CS2の領域を　フルに　使用してます。　ちなみに　CS0～CS7は7種類の　チップセレクト信号と　考えて下さい。

　それと、先ほども書きましたが 使用されるD-RAMのデータバス幅は 8bitです。よって、一回のアクセスで　読み書き出来るデータの 幅は 8bitのみです。　16bit整数、または　32bit整数を　読み書きする事は　可能ですが、16bit整数の場合 2回連続アクセス、32bit整数の場合、4回連続アクセスを行います。よって、その分 遅くなるという事です。　内蔵の RAMメモリと比べると　遅いです。

メモリマップ



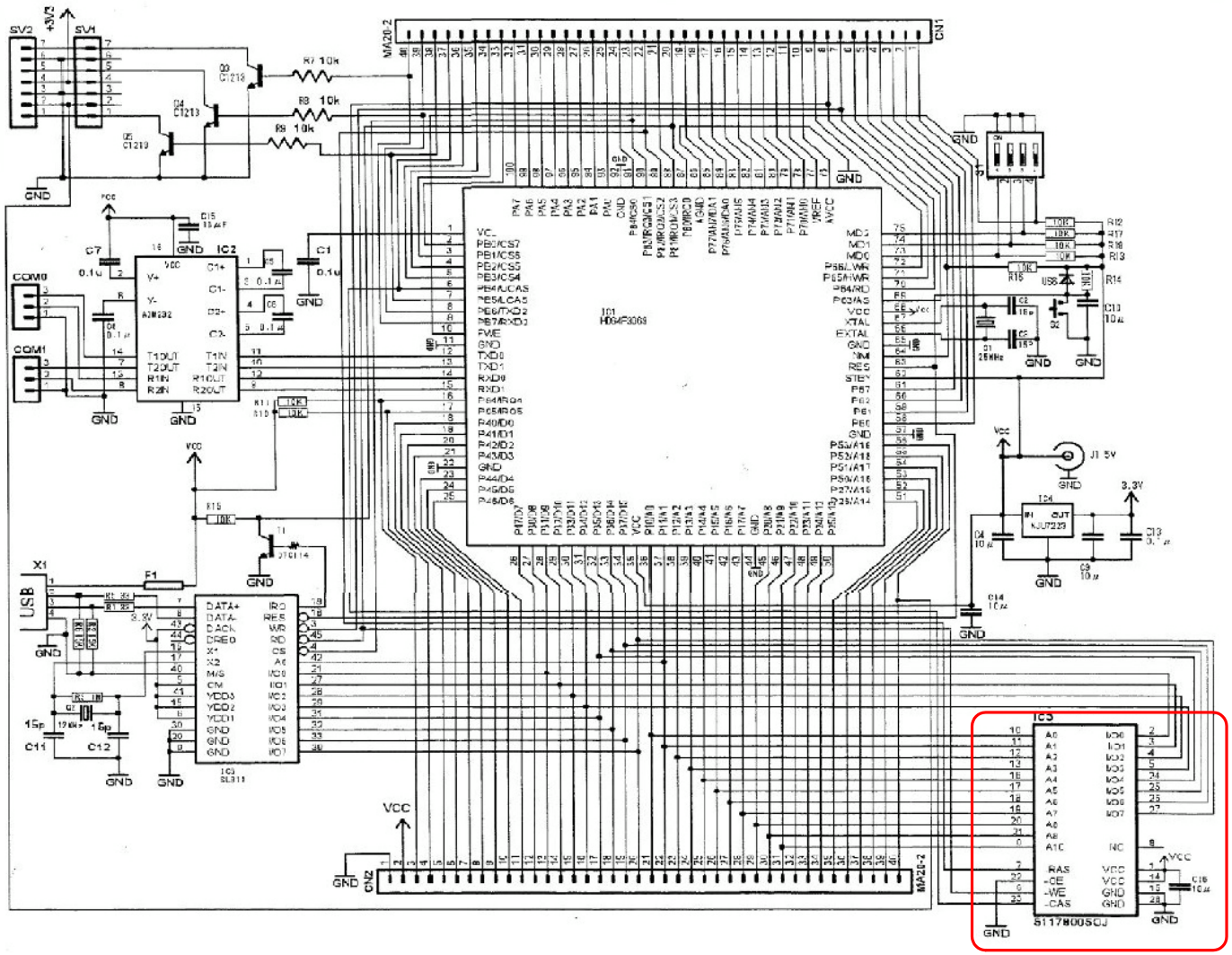
AE-3069USB基板の 回路図

回路図は、細かくて詳細を見るのは無理と思います。

大雑把に右下の 赤枠内 が D-RAMです。

秋月電子の基板を購入すると 説明書に回路図も 記載されてます。その印刷物を 画像としてスキャナで取り込んだ事もあり 荒い画像となっています。

バス線は 2つの 40ピンコネクタで、外部に引き出せるので、他の 外部素子も インタフェース仕様が合えば 接続する事も可能と思います。



D-RAM、ソフトからのアクセス

AE-3069USB基板裏面の D-RAMですが、今度は ソフトからの アクセスについて説明します。 まず、初期化が 必要です。

幸い、秋月の基板キットの説明書に D-RAMを 使用する上で 必要となる I/O レジスタにコマンドを出すシーケンスが書いてありました。
右の 赤枠で 囲った部分です。

但し、Cのソースでは ありませんので、移植時ひと手間掛かります。

初期化が、終われば通常のメモリとして
自由にアクセスできます。

尚、右上の赤枠上の文言は 秋月電子基板キットの説明書に 記載されていた文面です。

付属D-RAMの使い方
付属D-RAMは モード5で 使用します。
電源ON直後の状態では、正しく D-RAMを 使う事が 出来ません。以下のように設定します。

```
P1DDR ← FFH
P2DDR ← FFH
P8DDR ← 1CH
RTCOR ← 0AH
RTMCSR ← 18H
DRCRB ← 90H
DRCRA ← 30H
```

P1DDR、P2DDR、P8DDRは HEWの C上に簡単に移植出来ると 思いますが、RTCOR、RTMCSR、DRCRB、DRCRA は ちょっと悩むと思います。 参考にするのは iodefine.h と H8/3069の ハードウェアデータシートです。
次ページに 移植結果をお見せします。

```
P1DDR = 0xFF;            // P1 は  Address Bus A7 ～ A0
P2DDR = 0xFF;            // P2 は  address Bus A15 ～ A8
P8DDR = 0x1C;            // P8.b2=CS2(D-RAMの区画)

BSC.RTCOR = 0x0A;        // リフレッシュタイム コンスタント レジスタ
BSC.RTMCSR.BYTE = 0x18;// リフレッシュタイマコントロール／ステータスレジスタ
BSC.DRCRB.BYTE = 0x90;  // DRAMコントロールレジスタB
BSC.DRCRA.BYTE = 0x30;  // DRAMコントロールレジスタA

wait_ms( 10 );           // 約 10ms D-RAM 安定時間待ち
```

上のソースに BSC という名称が出てますが H8/3069の バスコントローラの事です。
以下に、ハードウェアデータシートに書かれている概要を 転載しました。

本 LSI は バスコントローラ(BSC)を内蔵しており、外部アドレス空間を 8 つのエリアに分割して管理します。 各エリアでは、バス幅、アクセスステート数などのバス仕様を独立に 設定することが可能であり、複数のメモリを 容易に接続することができます。
また、バスコントローラは バス調停権機能を持っており、内部バスマスタである CPU、DMAコントローラ(DMAC)及び DRAM インタフェースの 動作を制御すると共に
外部に バス権を解放することが できます。 との事です。

D-RAM、ソフトからのアクセス

今回、用意した関数は H8_300H_iocs.hに
プロトタイプ宣言を 行っています。

H8_300H_iocs.h

```
//   D-RAM アドレス値
// ------------------------------------------------------------------------
#define DRAM_TOP_ADR (volatile _UBYTE *)0x400000     // D-RAM 先頭アドレス
#define DRAM_BTM_ADR (volatile _UBYTE *)0x5FFFFF     // D-RAM 最終アドレス
#define DRAM_BYTE_SIZE    0x200000          // D-RAMの メモリサイズ( byte )

～～～～～～～～～～～～～～～～（途中省略）～～～～～～～～～～～～～～～～～～～～

//  ★ 秋月電子 AE-3069USB基板 実装 D-RAM アクセス処理 ( dram_acc.c )
// ------------------------------------------------------------------------
short  init_dram( void );                    // D-RAM初期化 ( メモリゼロクリアも行う )
void clear_dram( _UBYTE *adr, _UDWORD len );  // D-RAM領域のゼロクリア( dram_sub.src )
void test_wr_dram( _UBYTE *adr, _UDWORD len ); // 00h～FFhのテストデータを書き込む
_UBYTE  *dump_page( _UBYTE *adr );           // 16byte x 16行 ダンプ表示
```

D-RAMアドレス値は、H8_300H_iocs.h 先頭に 宣言しています。
D-RAMアクセス関数プロトタイプ宣言は H8_300H_iocs.h 下部に あります。

今回は、関数の中身の細かい説明は省略します。

D-RAMの初期化処理の使い方の説明をします。

```
//*************************
//**  D-RAM 初期化処理    **
//** ------------------- **
//**  関数値： 1 = 正常    **
//**          0 = 異常    **
//*************************
short  init_dram( void )
```

関数値は、1であれば、正常(D-RAMが正常に機能している)で、0 であれば、異常(D-RAMが機能して無い)事を 意味します。尚、正常であれば D-RAM を 先頭 byte から 終端 byteまで ゼロクリアします。

D-RAM内容を ゼロクリアする関数

```
; ***************************************
; **  D-RAM メモリクリア                **
; ** ------------------------------- **
; **  ER0 : 先頭アドレス (_UBYTE *adr ) **
; **  ER1 : カウンタ  ( long len )      **
; **  関数値：無し                      **
; ***************************************
    .export  _clear_dram
_clear_dram:
```

大容量のメモリを高速で ゼロクリアするためにアセンブラで 出来ています。 その関係で 引数1は ER0 レジスタで 引数2は ER1 レジスタとなります。

C言語から この関数を呼ぶ場合は 先頭の _ を取り除いた関数名で 引数は カッコ内の _UBYTE *adr と、long len の様に 呼び出して下さい。

D-RAM全体のゼロクリアは init_dram関数で 行っているので 部分的な消去に 使用して下さい。

```
; ****************************************
; **  D-RAM全エリアに テストデータを     **
; **  書き込み                          **
; **  テストデータは                    **
; **  00h ～ FFを 繰り返し書き込み      **
; ** ---------------------------------- **
; **  ER0 : 先頭アドレス ( _UBYTE *adr ) **
; **  ER1 : カウンタ ( long len )       **
; **  関数値 : 無し                     **
; ****************************************
     .export _test_wr_dram
_test_wr_dram:
```

これも アセンブラの関数です。
C言語からの呼び出し方は 前の関数と同じです。
指定されたアドレスから lenの バイト数分
00h ～ FFh を 繰り返し書き込みます。
D-RAMの テスト用の関数です。

```
//****************************************
//**  16byte x 16行 ダンプ表示          **
//** ---------------------------------- **
//**  adr :  ダンプ表示したいデータの先頭 **
//**         アドレス                   **
//**  関数値 :                          **
//**  引数 adr に 256 を 足した値です。  **
//**  メモリアドレスを 連続的に ダンプし **
//**  続ける場合は、関数値を 次のダンプ  **
//**  処理の引数として使用して下さい。   **
//****************************************
_UBYTE  *dump_page( _UBYTE *adr )
```

この関数は C で 出来ています。
指定された先頭アドレスから 256 byteのデータを
横 16byte分の HEX表示と 表示可能であれば
ASCII文字で 表示します。それを 16 行 縦に
表示して 計 256 byteの ダンプ表示を行います。
　目的のデータが　メモリの所定箇所に 正しく
入っているか、確認の用途で使用して下さい。

まず最初に、init_dram関数を 呼び出して下さい。 通常の データの書き込み読み出しは ポインタ変数を使って行って下さい。

D-RAMの先頭アドレスは、H8_300H_iocs.h 内の先頭に宣言してある DRAM_TOP_ADR を 使用して下さい。 例）2byte整数の場合：

```
short  *ptr;
ptr = (short *)DRAM_TOP_ADR;
```

として下さい。

他に

```
DRAM_BTM_ADR  // D-RAM 最終アドレス
DRAM_BYTE_SIZE // D-RAM全体のバイトサイズ
```

が、あります。

では、D-RAM全領域に 00h ～ FFhのテストデータを 書き込み、HEXダンプを行う実験を行います。