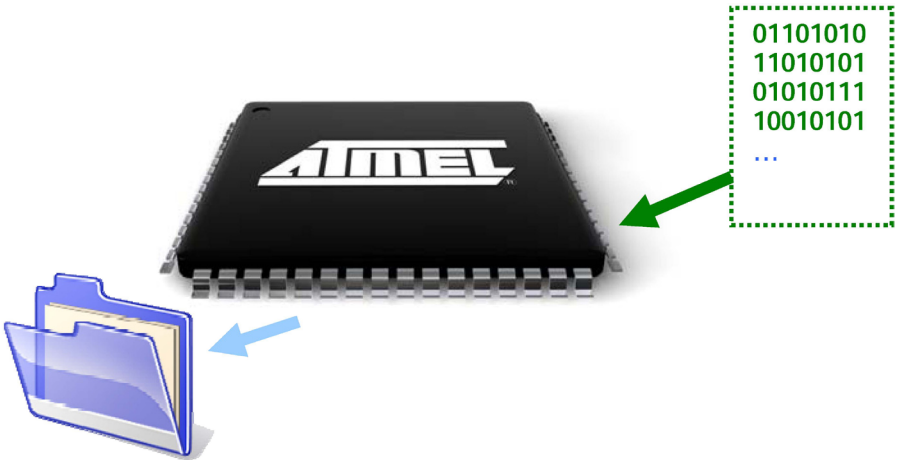
AVR115 : ATmega32U4に於ける Atmelファイル システムでのデータ記録



8ビット AVR® マイクロ コントローラ

応用記述

1. 序説

Atmel®はAT90USBxとATmegaxxUxデバイス用のファイル システム管理を提供します。このファイル システムはデータ記録操作実行を許し、この応用記述はATmeha32U4とEVK527基板を用い、この機能の実装方法を示します。"EVK527-series4-datalogging"がAVR®115と連携するファームウェア一括です。

読者はこれを読む前にAVR114応用記述を習熟すべきです。

2. ハードウェア必要条件

データ記録例応用は以下のハードウェアが必要です。

- 以下を含むATEVK527 AVR USB評価基板
  - ATmega32U4
  - DataFlash® (32Mビット)
  - SD/MMCコネクタ
- USBケーブル (標準A-ミニB)
- Windows® (98SE, ME, 2000, XP)、Linux®または、USB 1.1か2.0ホストを持つMAC® OS

3. 実装書き込みとデバイス ファームウェア更新

デバイスをプログラミングするのに以下の方法の1つを使うことができます。

- JTAGICEmk IIを使うJTAGインターフェース
- AVRISPmk IIとJTAGICEmk IIを使うSPIインターフェース
- 工場DFUブートローダとFLIP(注1)ソフトウェアによるUSBインターフェース
- STK®500またはSTK600を使う並列プログラミング

USBドライバをインストールし、USBインターフェースを通してデバイスをプログラミングする方法を知るにはFLIP(注1)のヘルプ内容を参照してください。

注1: FLIPは工場DFUブートローダによってUSBインターフェース(外部ハードウェア不要)を通してAtmelのデバイスのプログラミングを使用者に許すことをAtmelによって提供されるソフトウェアです。

注: JTAGICEmk IIに於いてAVR Studio®での"erase before programming"チェック枠に注意してください。チェックされた場合、プログラミングに先立ってDFUブートローダが削除されます。

本書は一般の方々の便宜のため有志により作成されたもので、Atmel社とは無関係であることを御承知ください。しおりのはじめにでの内容にご注意ください。

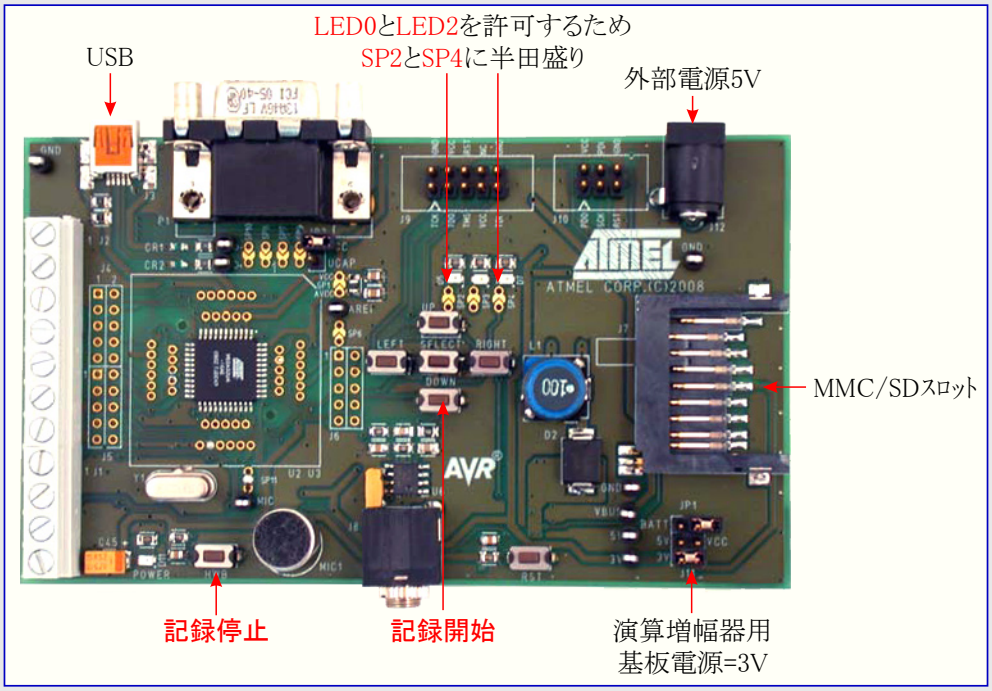
Rev. 8202A-01/09, 8202AJ2-01/21

4. 即時開始

一旦ﾃﾞﾊﾞｲｽがEVK527-series4-datalogging.hexﾌｧｲﾙで書かれると、ﾃﾞｰﾀ記録実演を開始することができます。

1. USBｹｰﾌﾞﾙを切断して電源ｹｰﾌﾞﾙ(9V)を接続してください。
2. 記録を始めるために↓ｷｰを押してください。
   - 記録ﾌｧｲﾙは存在したならばMMC/SDｶｰﾄﾞ、またはDataFlashﾒﾓﾘのどちらかに作成されます。
   - LED0は記録開始時にONにされます。
   - LED2は異常発生時(ﾃﾞｨｽｸ不在、ﾃﾞｨｽｸ満杯など)にONにされます。
3. 記録を停止するのに、HWBｷｰを押すか、またはUSBｹｰﾌﾞﾙを接続してください。
4. U-Diskを走らせて記録ﾌｧｲﾙ"ﾃﾞｨｽｸ:¥log000¥log000.bin"を読むためにPCへUSBｹｰﾌﾞﾙを接続してください。

図4-1. EVK527改訂1



注: これは半田付けされなければならないSP2とSP4を除いてEVK527の既定工場構成設定です。

既定での記録されたﾃﾞｰﾀは増加されるﾃﾞｼﾞﾀﾙ数値(16ﾋﾞｯﾄ)だけで、120µs毎に格納されます。これはdatalogging.cﾌｧｲﾙでのｿﾌﾄｳｪｱｺﾝﾊﾟｲﾙ任意選択経由で記録供給元を変更することができます。

```
#define LOG_ADCMIC_EXAMPLE          // ﾏｲｸﾛﾌｫﾝのADCからWAVﾌｧｲﾙへ
#define LOG_ADCEXT_EXAMPLE          // 他のA/D変換器入力からBINﾌｧｲﾙへ
#define LOG_PIN_EXAMPLE             // 記録ﾋﾟﾝの状態をBINﾌｧｲﾙへ
#define LOG_ENUM_EXAMPLE            // 記録計数器値をBINﾌｧｲﾙへ (既定)
```

5. 応用

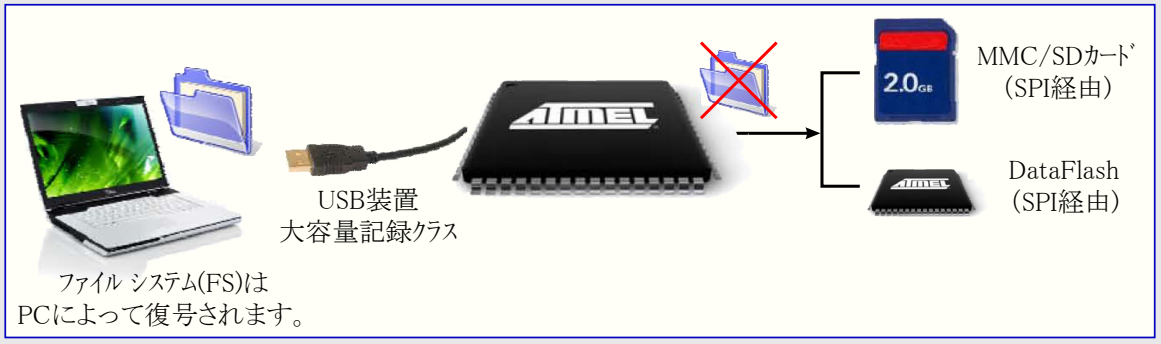
5.1. 動き

試供応用は2つの動作形態を提供します。

ﾀﾞｳﾝﾛｰﾄﾞ動作形態：使用者はﾒﾓﾘ(DataFlashまたはSD/MMCｶｰﾄﾞ)に書かれた記録ﾌｧｲﾙにｱｸｾｽし得るために、ｷｯﾄをPCに接続しなければなりません。この動作形態ではﾒﾓﾘのｱｸｾｽに組み込まれたAtmelのﾌｧｲﾙ ｼｽﾃﾑは許されません。

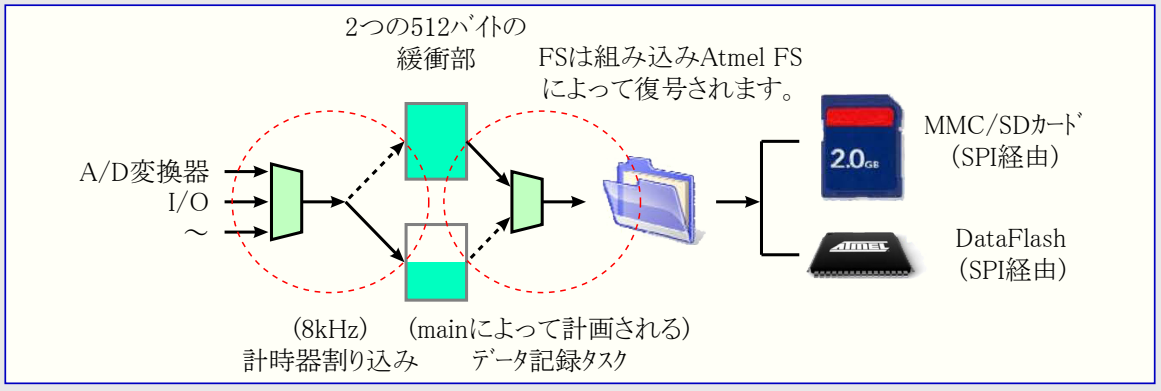
注: ﾃﾞｰﾀの不正を防ぐため、一時に1つのﾌｧｲﾙ ｼｽﾃﾑ管理だけを活性にすることができます。

図5-1. ﾀﾞｳﾝﾛｰﾄﾞ動作形態(U-Disk)



ﾃﾞｰﾀ記録動作形態：この動作形態の間、ﾃﾞｰﾀ記録が実行されます。ｷｯﾄはUSBﾎｽﾄから切断されなければならず、記録活動の開始/停止はEVK527釦によって管理されます(図4-1. EVK527改訂1をご覧ください)。ﾃﾞｰﾀ記録開始時にﾌｧｲﾙがﾒﾓﾘに作成されます。8kHz割り込みの計時器がA/D変換器または別の供給元からの値を採取してそれらを緩衝部に書きます。満杯の緩衝部はﾃﾞｰﾀ記録ﾀｽｸによってﾌｧｲﾙに転送されます。

図5-2. ﾃﾞｰﾀ記録動作形態



5.2. ファームウェア

この項はファイル システム管理ｺｰﾄﾞだけを説明し、USB単位部は説明されません。以下のｺｰﾄﾞ例は"EVK527-series4-datalogging"一括からのdata_logging.cファイルから抽出されています。

5.2.1. 組み込みFSの許可/禁止

ファイル システム単位部の初期化と抜け出しはdatalogging_task()によって管理されます。

ﾁｯﾌﾟがUSB装置動作形態から抜け出す時に、組み込みAtmelファイル システムを初期化するのに"nav_reset()"を呼ぶことができます。

組み込みAtmelファイル システムの停止を望む、またはUSB装置動作を開始する時に、ﾒﾓﾘ内のｷｬｯｼｭ情報を吐き出すために"nav_exit()"が呼ばれなければなりません。

■ datalogging_task()ルーチンの例

```
void datalogging_task(void)
{
   // ﾃﾞｰﾀ記録状態変更
   if( Is_joy_down()               // 使用者によるﾃﾞｰﾀ記録開始で、
   && (!Is_device_enumerated())) // 且つUSB装置動作が停止なら、
   {
      if( !g_b_datalogging_running )
      {
         // ﾃﾞｰﾀ記録開始
         nav_reset();              //** ファイル システム初期化
         g_b_datalogging_running = datalogging_start();
      }
   }
   if( Is_hwb()                    // 使用者によるﾃﾞｰﾀ記録停止、
   ||  Is_device_enumerated() )  // またはUSB装置動作開始なら、
   {
      if( g_b_datalogging_running )
      {
         // ﾃﾞｰﾀ記録停止
         g_b_datalogging_running = FALSE;
         datalogging_stop();
         nav_exit();               //** ファイル システム単位部抜け出し
      }
   }

   // ﾃﾞｰﾀ記録背後ﾀｽｸ実行
   if( g_b_datalogging_running )
   {
      g_b_datalogging_running = datalogging_file_write_sector();
      if( !g_b_datalogging_running )
      {
         datalogging_stop();
         nav_exit();               //** ファイル システム単位部抜け出し
      }
   }
}
```

5.2.2. ディスク オープン

この例では、応用が同時に1つだけのﾃﾞｨｽｸを調査して1つのﾌｧｲﾙだけを開くので、1つの誘導部ﾊﾝﾄﾞﾙ(注)だけが必要とされます。その場合に、我々は既定誘導部ﾊﾝﾄﾞﾙ0"nav_select(0)"を選びました。

注: 誘導部ﾊﾝﾄﾞﾙについてのより多くの情報に関してはAVR114応用記述をご覧ください。

ﾃﾞｨｽｸを開くための算法は接続されるﾃﾞｨｽｸ数に依存します(conf_access.hﾌｧｲﾙ内の構成設定をご覧ください)。既定によって例はDataFlashとMMC/SDのﾄﾞﾗｲﾊﾞと以下の算法を使います。

MMC/SDﾃﾞｨｽｸの搭載と必要ならばﾌｫｰﾏｯﾄを試みます。
異常(ﾃﾞｨｽｸ不在、失敗など)なら、
DataFlashﾃﾞｨｽｸの搭載と必要ならばﾌｫｰﾏｯﾄを試みます。
異常(ﾃﾞｨｽｸ不在、失敗など)なら、
ﾃﾞｰﾀ記録中止

■ datalogging_open_disk()ルーチン

```
Bool datalogging_open_disk(void)
{
   U8 u8_i;
   nav_select(FS_NAV_ID_DEFAULT);

#if( (LUN_2 == ENABLE) && (LUN_3 == ENABLE)) // 構成設定はconf_access.hで設定
   // MMC/SDﾃﾞｨｽｸ(lun 1)またはDataFlashﾃﾞｨｽｸ(lun 0)の選択と搭載試行
   // MMC/SDを先に試行
   for( u8_i=1; u8_i!=0xFF; u8_i-- )
#else
   // MMC/SDまたはDataFlash(lun 0)の1つだけのﾒﾓﾘが存在
   for( u8_i=0; u8_i!=0xFF; u8_i-- )
#endif
   {
      if( nav_drive_set(u8_i) )              // ﾄﾞﾗｲﾊﾞ選択(ﾃﾞｨｽｸではない)
      {
         // ﾄﾞﾗｲﾊﾞが利用可能なら、それを搭載
         if( !nav_partition_mount() )
         {
            // 搭載中に異常なら、異常状況検査
            if( FS_ERR_NO_FORMAT != fs_g_status )
               continue;                     // ﾃﾞｨｽｸ失敗(不在、ﾊｰﾄﾞｳｪｱ異常、ｼｽﾃﾑ異常など)
            // ﾃﾞｨｽｸが未ﾌｫｰﾏｯﾄなら、それをﾌｫｰﾏｯﾄ
            if( !nav_drive_format(FS_FORMAT_DEFAULT) )
               continue;                     // ﾌｫｰﾏｯﾄ失敗
         }
         return TRUE;                        // ここでﾃﾞｨｽｸ搭載済み
      }
   }
   return FALSE;                             // 有効なﾃﾞｨｽｸ未発見
}
```

5.2.3. パス ファイル作成

この部分は以下のパス ファイル"ディスク:¥logxxx¥logxxx.bin"を作成します。

datalogging_create_path_file()ルーチンは"¥logxxx"ディレクトリを作成します。"nav_setcwd()"ルーチンが検索して結局、パスを作成します(第3引数はTRUEでなければなりません)。

■ datalogging_create_path_file()ルーチン

```
Bool datalogging_create_path_file(void)
{
   char ascii_name[15];
   U16 u16_dir_num = 0;

   if( !nav_dir_root() )
      return FALSE;            // FS異常

   while( u16_dir_num < 30 )   // ディレクトリ数の制限は単に例です。
   {
      sprintf( ascii_name, "./log%03d/", u16_dir_num);

      // 副ディレクトリ移行、不在なら結局作成
      if( !nav_setcwd( ascii_name, FALSE, TRUE) )
         return FALSE;         // FS異常

      // ファイル作成
      if( datalogging_create_file() )
         return TRUE;          // ファイル作成

      // ここで、ディレクトリが一杯なら、次の副ディレクトリを作成するために親ディレクトリへ行きます。
      if( !nav_dir_gotoparent() )
         return FALSE;         // FS異常
      u16_dir_num++;
   }
   return FALSE;               // 記録ディレクトリとファイルが多すぎ
}
```

datalogging_create_file()ルーチンは"¥logxxx.bin"ファイルを作成します。これは多数のファイルを持つディレクトリの調査が遅すぎるために、記録ディレクトリ内の記録ファイル数を10に制限します。

■ datalogging_create_file()ルーチン

```
Bool datalogging_create_file(void)
{
   char ascii_name[15];
   U16 u16_file_num = 0;

   while( u16_file_num < 10 )
   {
      // ファイル作成
      sprintf( ascii_name, "log%03d.bin", u16_file_num);
      if( nav_file_create( ascii_name ))
         return TRUE;    // ここで、ファイルが作成され、閉じられ、空にされます。
      // 作成中に異常なら、異常を調査
      if( fs_g_status != FS_ERR_FILE_EXIST )
         return FALSE;   // FS異常(ディスク満杯、ディレクトリ満杯など)
      // 既にファイル存在なら、名前の数値進行
      u16_file_num++;
   }
   return FALSE;          // 現在のディレクトリで記録ファイルが多すぎ
}
```

5.2.4. ファイル空間割り当て

この項は必須ではありませんが、データ記録の帯域の増加を許します。より多くの情報についてはAVR114応用記述の6.5.をご覧ください。

注: データ記録の最後で、割り当てられた残りメモリ(割り当てられた容量-最終容量)はファイルが閉じられた時に開放されます。

■ datalogging_alloc_space()ルーチン

```
Bool datalogging_alloc_file_space(void)
{
   Fs_file_segment g_recorder_seg;

   // 作成したファイルを書き込み形態で開く
   if( !file_open(FOPEN_MODE_W))
      return FALSE;

   // 割り当てるセグメント量を定義(512バイト単位)
   // 注: 総容量を知らない場合はより多くを割り当てることができます。
   g_recorder_seg.u16_size = FILE_ALLOC_SIZE;

   // セグメント容量よりも小さいか等しいクラスター一覧でFATに割り当て
   if( !file_write( &g_recorder_seg ))
   {
      file_close();
      return FALSE;
   }

   // 望むなら、割り当てた最小容量を調べることができます。
   if( g_recorder_seg.u16_size < FILE_ALLOC_SIZE_MIN )
   {
      file_close();
      nav_file_del(FALSE);
      return FALSE;
   }

   // 容量をリセットするためにファイルを閉じて/開いてください。
   // 注: この手順は直前のFAT割り当てを取り去りません。
   file_close();                              // ファイルを閉じる
   if( !file_open(FOPEN_MODE_W))              // 容量を0に強制した書き込み形態でファイルを開く
     return FALSE;
   return TRUE;    //** ファイル オープンとFAT割り当て
}
```

5.2.5. ファイルを満たす

"file_write_buf()"はファイルを満たすための最良のルーチンです。メモリ インターフェースが512バイトの塊を使うため、緩衝部容量と現在のファイル位置として512バイトの倍数を使うことが最適な速度性能を与えます。

緩衝部(512バイト×2)は計時器0割り込みルーチンによって満たされます。8MHzでのATmega32U4の最大データ記録帯域は18Kバイト/sです。この値は以下の例で測定されました。

■ datalogging_file_write_sector()ルーチン

```
Bool datalogging_file_write_sector(void)
{
   // !!!! 注:   // 書かれた緩衝部容量が512バイトの倍数で、
   // 且つ、現在のファイル位置が512バイトの倍数なら、
   // "file_write_buf()"ルーチンは非常に効率的です。
   if( g_b_buf_full[g_u8_cur_buf] )
   {
      if( !file_write_buf( &g_data_buf[g_u8_cur_buf*FS_SIZE_OF_SECTOR], FS_SIZE_OF_SECTOR ) )
         return FALSE;  // 書き込み異常
      g_b_buf_full[g_u8_cur_buf] = FALSE;
      // 新しい緩衝部待ち
      g_u8_cur_buf++;
      if( NB_DATA_BUF == g_u8_cur_buf )
        g_u8_cur_buf = 0;
   }
   return TRUE;
}
```

本社

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
TEL 1(408) 441-0311
FAX 1(408) 487-2600

国外営業拠点

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
TEL (852) 2245-6100
FAX (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
TEL (33) 1-30-60-70-00
FAX (33) 1-30-60-71-11

Atmel Japan
104-0033 東京都中央区
新川1-24-8
東熱新川ﾋﾞﾙ 9F
ｱﾄﾒﾙ ｼﾞｬﾊﾟﾝ株式会社
TEL (81) 03-3523-3551
FAX (81) 03-3523-7581

製品窓口

ｳｪﾌﾞｻｲﾄ
www.atmel.com

技術支援
avr@atmel.com

販売窓口
www.atmel.com/contacts

文献請求
www.atmel.com/literature

本応用記述はAtmelのAVR115応用記述(doc8202.pdf Rev.8202A-01/09)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。