Armadillo-400 シリーズ
新フラッシュメモリ移行ガイド

A410*-***Z → A411*-***Z
A420*-***Z → A441*-***Z
A440*-***Z → A441*-***Z
A460*-***Z → A462*-***Z
A461*-***Z → A463*-***Z
AB111-***Z → AB121-***Z

Version 1.1.0
2018/07/10
linux-3.14-at, linux-2.6.26-at 対応
Armadillo-420/440/410/460 対応

株式会社アットマークテクノ [http://www.atmark-techno.com]

Armadillo サイト [http://armadillo.atmark-techno.com]

Armadillo-400 シリーズ新フラッシュメモリ移行ガイド

株式会社アットマークテクノ

Version 1.1.0
2018/07/10

目次

図目次

表目次

1. はじめに

本書では、Armadillo-400 シリーズのフラッシュメモリ変更について、変更内容と、ソフトウェアの影響範囲について説明します。

本書で説明するフラッシュメモリに対応したソフトウェアは下記のバージョンです。

表 1.1 新フラッシュメモリ適用品対応ソフトウェア

項目	Linux 3.14	Linux 2.6.26
ブートローダー	v3.10.0 以降	v2.3.0 以降 v2.5.0 以降 Armadillo-460 新フラッシュメモリ適用品に対応
カーネル	at10 以降(標準イメージ:2.05 以降)	at28 以降(標準イメージ:1.18 以降) at30 以降(標準イメージ:1.10 以降) Armadillo-460 新フラッシュメモリ適用品に対応

1.1. 対象製品

従来製品と、その後継にあたる新フラッシュメモリ適用品の関係を次に示します。Armadillo-420 の後継は Armadillo-440 の相当品となります。

従来品 型番	モデル名	新フラッシュメモリ適用品 型番
A4601-D00Z	Armadillo-460 ベーシックモデル開発セット	A4621-D00Z
A4100-D00Z	Armadillo-410 液晶モデル開発セット	A4110-D00Z
A4100-U00Z	Armadillo-410 量産ボード	A4110-U00Z
A4200-D00Z	Armadillo-420 ベーシックモデル開発セット	A4410-D00Z
A4202-D00Z	Armadillo-420 WLAN モデル開発セット （AWL13 対応）	A4412-D00Z
A4400-D00Z	Armadillo-440 液晶モデル開発セット	A4411-D00Z
A4400-U00Z	Armadillo-440 量産ボード（リード部品実装済）	A4410-U00Z
A4400-B00Z	Armadillo-440 量産ボード（リード部品未実装・部品付）	A4410-B00Z
A4400-N00Z	Armadillo-440 量産ボード（リード部品未実装・部品無）	A4410-N00Z
A4200-U00Z	Armadillo-420 量産ボード（リード部品実装済）	A4410-U00Z
A4200-B00Z	Armadillo-420 量産ボード（リード部品未実装・部品付）	A4410-B00Z
A4200-N00Z	Armadillo-420 量産ボード（リード部品未実装・部品無）	A4410-N00Z
A461*-U00Z	Armadillo-460 量産ボード 3.3V 高速拡張バス （標準ラインアップ品、 およびその他カスタマイズ品）	A463*-U00Z
A4611-B00Z	Armadillo-460 量産ボード 3.3V 高速拡張バス(ベアボード) （標準ラインアップ品、 およびその他カスタマイズ品）	A4631-B00Z
A460*-U00Z	Armadillo-460 量産ボード PC/104 バス （標準ラインアップ品、 およびその他カスタマイズ品）	A462*-U00Z
A4601-B00Z	Armadillo-460 量産ボード PC/104 バス(ベアボード) （標準ラインアップ品、 およびその他カスタマイズ品）	A4621-B00Z

1.2. 変更通知に関して

新フラッシュメモリ適用品での変更内容、新フラッシュメモリの型番等については、変更通知を参照してください。

アットマークテクノ ユーザーズサイト

Armadillo-410/440/IoT ゲートウェイ G2/Box WS1 搭載フラッシュメモリの変更について

https://users.atmark-techno.com/node/2726

2. 変更による影響: フラッシュメモリのメモリマップ

ここでは、フラッシュメモリのメモリマップの変更に関して説明します。

電気的特性、外観仕様、寸法仕様、機能・性能に関する影響につましては「1.2. 変更通知に関して」に記載されている変更通知文書を参照してください。

2.1. メモリマップ変更点(hermit-at-v3 + linux-3.14-at の場合)

従来品のフラッシュメモリのメモリマップを「表 2.1. 従来品 フラッシュメモリ メモリマップ」に示します。

表 2.1 従来品 フラッシュメモリ メモリマップ

物理アドレス	パーティション名	サイズ	工場出荷状態で書き込まれているソフトウェア
0xA0000000 | 0xA001FFFF	bootloader	128kByte	Hermit-At ブートローダーイメージ
0xA0020000 | 0xA041FFFF	kernel	4MByte	Linux カーネルイメージ
0xA0420000 | 0xA1EFFFFF	userland	26.875Mbyte	Atmark Dist ユーザーランドイメージ
0xA1F00000 | 0xA1FFFFFF	config	1MByte	アプリケーションの設定情報など

新フラッシュメモリ適用品では「表 2.2. 新フラッシュメモリ適用品 フラッシュメモリ メモリマップ」に変更となります

表 2.2 新フラッシュメモリ適用品 フラッシュメモリ メモリマップ

物理アドレス	パーティション名	サイズ	工場出荷状態で書き込まれているソフトウェア
0xA0000000 | 0xA003FFFF	bootloader	256kByte	Hermit-At ブートローダーイメージ
0xA0040000 | 0xA043FFFF	kernel	4MByte	Linux カーネルイメージ
0xA0440000 | 0xA1EFFFFF	userland	26.75Mbyte	Atmark Dist ユーザーランドイメージ
0xA1F00000 | 0xA1FFFFFF	config	1MByte	アプリケーションの設定情報など

bootloader 領域は 128kByte から 256kByte に拡張にされ、その影響で、kernel と userland 領域の先頭アドレスが 128kByte ずつ後方にずれます。また、ユーザーランド領域が 128kByte 減少しています。

2.2. メモリマップ変更点(hermit-at + linux-2.6.26-at の場合)

従来品のフラッシュメモリのメモリマップを「表 2.3. 従来品 フラッシュメモリ メモリマップ」に示します。

表 2.3 従来品 フラッシュメモリ メモリマップ

物理アドレス	パーティション名	サイズ	工場出荷状態で書き込まれているソフトウェア
0xA0000000 | 0xA001FFFF	bootloader	128kByte	Hermit-At ブートローダーイメージ
0xA0020000 | 0xA021FFFF	kernel	2MByte	Linux カーネルイメージ
0xA0220000 | 0xA1FDFFFF	userland	29.75Mbyte	Atmark Dist ユーザーランドイメージ
0xA1FE0000 | 0xA1FFFFFF	config	128kByte	アプリケーションの設定情報など

新フラッシュメモリ適用品では「表 2.4. 新フラッシュメモリ適用品 フラッシュメモリ メモリマップ」に変更となります

表 2.4 新フラッシュメモリ適用品 フラッシュメモリ メモリマップ

物理アドレス	パーティション名	サイズ	工場出荷状態で書き込まれているソフトウェア
0xA0000000 | 0xA003FFFF	bootloader	256kByte	Hermit-At ブートローダーイメージ
0xA0040000 | 0xA023FFFF	kernel	2MByte	Linux カーネルイメージ
0xA0240000 | 0xA1FDFFFF	userland	29.625Mbyte	Atmark Dist ユーザーランドイメージ
0xA1FD0000 | 0xA1FFFFFF	config	128kByte	アプリケーションの設定情報など

bootloader 領域は 128kByte から 256kByte に拡張にされ、その影響で、kernel と userland 領域の先頭アドレスが 128kByte ずつ後方にずれます。また、ユーザーランド領域が 128kByte 減少しています。

2.3. フラッシュメモリ メモリマップの変更が必要になった経緯

従来製品のフラッシュメモリは Erase Block の構成が、top 32kByte が 4 個、残りが bottom 128kByte となっています。 32kByte の Erase Block を 4 つ、合計 128kByte を bootloader 領域として割り当て、4 つ目の Erase Block を、ブートローダー:hermit-at の setenv や setbootdevice コマンドのパラメータを保存する「パラメータ領域」として使用していました。

しかし、新フラッシュメモリでは、top 32kByte の Erase Block はなくなり、すべてが、128kByte Erase Block の構成となっております。 このため、新フラッシュメモリ適用品では、128kByte Erase Block を 2 つ bootloader 領域として割り当て、2 つめの 128kByte Erase Block を「パラメータ領域」として使用するように変更しました。

なお、フラッシュメモリ全体のサイズは変更はありません。

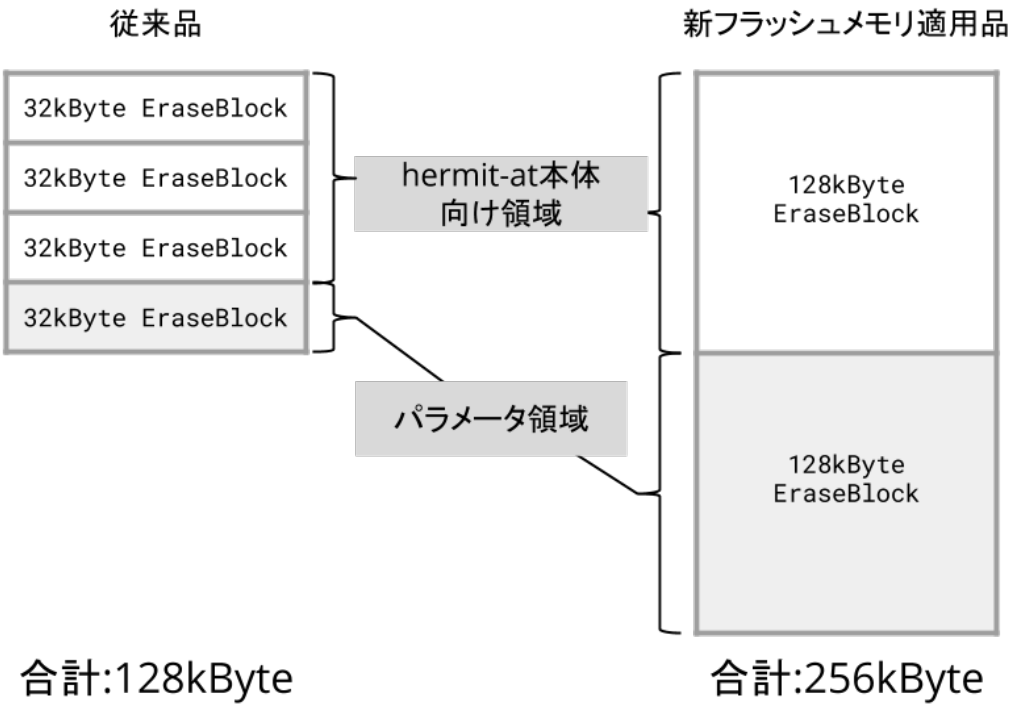

図 2.1 Erase Block サイズの違いによる bootloader 領域の違い

3. ソフトウェアの移行に関して

3.1. 対応ソフトウェア

新フラッシュメモリ適用品を動作させるには、「表 1.1. 新フラッシュメモリ適用品対応ソフトウェア 」に記載したソフトウェアを使用してください。

従来品で上記の新フラッシュメモリ適用品対応ソフトウェアを使用した場合、フラッシュメモリメモリマップは従来と変らず 「表 2.1. 従来品 フラッシュメモリ メモリマップ」に記載した仕様で動作します。

3.2. お客様が開発したソフトウェアの移行方法について

ここでは、従来品をベースに開発したソフトウェア(Hermit-At, Linux カーネル)を、どのように新フラッシュメモリ適用品に移行すべきか考えられるパターンを記載します。

本章で示すソースコードの変更箇所の例は hermit-at-v3 と Linux-3.14-at について示しています。hermit-at(v2.x.x) と linux-2.6.26-at の場合を含む、全ての差分はパッチファイルとしてダウンロードすることができます。

新フラッシュメモリ適用品向けアップデートの差分ダウンロード

https://download.atmark-techno.com/misc/new_flash_memory/

ここで記載するのはあくまで移行方法の例であり、「2. 変更による影響: フラッシュメモリのメモリマップ」、「3.3. 新フラッシュメモリ適用に伴うソフトウェア変更箇所」、「3.4. ソフトウェアから新フラッシュメモリ適用品か判別する方法」を十分に確認し、お客様の開発状況に合わせ適切な方法を検討・適宜選択してください。

新フラッシュメモリ適用品対応ソフトウェアにアップデートする

ユーザーランド(Atmark-Dist)のみ変更しており、Hermit-At, Linux カーネルに変更を加えていない場合や、カーネルコンフィギュレーションのみ変更した場合は、基本的に、新フラッシュメモリ適用品対応ソフトウェアにそのままアップデートすることで移行が可能です。

お客様が行った修正を、新フラッシュメモリ適用品対応ソフトウェアに入れる

お客様で Hermit-At, Linux カーネルのソースコードの変更をしているが、簡単な変更のみである、または変更量が小さい場合、この方法がスムーズである場合が多いです。

お客様のソフトウェアに、新フラッシュメモリ適用に伴うソフトウェア変更箇所 を入れる

お客様でソースコードの変更をしており、変更量が多い、最新のソフトウェアにすると動かなくなる機能がある場合、この方法がスムーズである場合が多いです。

Hermit-At, Linux カーネルの変更をしていなくても、ユーザーランドから mtd デバイスファイル(/dev/mtd*)を open し、アクセスを行うようなアプリケーションを作成している場合、フラッシュメモリメモリマップが変更になっているため、影響がある可能性があります。

3.3. 新フラッシュメモリ適用に伴うソフトウェア変更箇所

新フラッシュメモリへの対応に伴い、Hermit-At、Linux カーネルの変更を行った箇所を記載します。

主に以下の変更を行なっています。

- 新フラッシュメモリのドライバーの有効化と変更
- 新フラッシュメモリ適用品か、従来品かの判別とそれに合わせたフラッシュメモリ メモリマップの設定

3.3.1. Hermit-At の変更点

3.3.1.1. src/target/driver/flash_core.c

CFI の情報を元に、搭載フラッシュメモリの判別を行う処理を追加しました。

```
--- a/src/target/driver/flash_core.c
+++ b/src/target/driver/flash_core.c
@@ -23,6 +23,50 @@ flash_eb eb;

 static flash_protect *protect_table = 0;

+#define flash_read(addr)         cpu_to_le16(read16((addr)))
+#define flash_write(addr, b)     write16((addr), cpu_to_le16((b)))
+
+static int qry_present(const u32 base)
+{
+       u8 offs, v;
+       u8 qry[] = "QRY";
+
+       for (offs = 0; offs < 3; offs++) {
+               v = flash_read(base + ((0x10 + offs)<<1));
+               if ( v != *(qry + offs))
+                       return 0;
+       }
+       return 1; // "QRY" found
+}
+
+int flash_get_cfi_vendor_spec(const u32 base_addr)
+{
+       int type;
+       u32 base;
```

```
+       u16 primary_spec = 0;
+
+       base = (base_addr & FLASH_BASE_MASK);
+       flash_write(base, 0xF0);
+       flash_write(base, 0xFF);
+       flash_write(base, 0x98);
+
+       if (!qry_present(base))
+               flash_write(base + (0x555 << 1), 0x98);
+       primary_spec = flash_read(base + (0x13 << 1));
+       switch (primary_spec){
+       case 0x0001:
+               type = FLASH_TYPE_INTEL;
+               break;
+       case 0x0002:
+               type = FLASH_TYPE_AMD;
+               break;
+       default:
+               type = -1;
+               break;
+       }
+       return type;
+}
+
 int flash_initialize(const int type, const u32 base_addr)
 {
        memzero(&fops, sizeof(flash_ops));
@@ -77,6 +121,16 @@ int flash_initialize(const int type, const u32 base_addr)
        return 0;
 }

+int flash_initialize_cfi(const u32 base_addr)
+{
+       int type;
+
+       type = flash_get_cfi_vendor_spec(base_addr);
+       if (type < 0)
+               return -1;
+       return flash_initialize(type, base_addr);
+}
+
 int flash_register_protect_table(flash_protect *table)
 {
        protect_table = table;
```

3.3.1.2. src/target/driver/flash_amd.c

新フラッシュメモリの buffer program に対応しました。

```
--- a/src/target/driver/flash_amd.c
+++ b/src/target/driver/flash_amd.c
@@ -21,8 +21,21 @@

 static flash_cfi cfi_info;

+#define AMD_CMD_UNLOCK_START           0xAA
+#define AMD_CMD_UNLOCK_ACK             0x55
```

```
+#define AMD_CMD_WRITE_TO_BUFFER               0x25
+#define AMD_CMD_WRITE_BUFFER_CONFIRM   0x29
+#define AMD_BUFFER_PROGRAM_ALIGNED_MASK (0xf)
+
+#define FORCE_WORD_PROGRAM (0)
 #define FLASH_TIMEOUT 400000000

+#define min(x, y) ({                \
+       typeof(x) _min1 = (x);       \
+       typeof(y) _min2 = (y);       \
+       (void) (&_min1 == &_min2);   \
+       _min1 < _min2 ? _min1 : _min2;})
+
 #define flash_read(addr)         read16(addr)
 #define flash_write(addr, b)     write16((addr), (b))

@@ -73,6 +86,12 @@ static int flash_status_full_check(addr_t addr, unsigned short value1, unsigned
        return (status1 != value1 || status2 != value2) ? -1 : 0;
 }

+static void flash_amd_unlock_seq(addr_t base)
+{
+       flash_write_cmd(base + (0x555 << 1), AMD_CMD_UNLOCK_START);
+       flash_write_cmd(base + (0x2AA << 1), AMD_CMD_UNLOCK_ACK);
+}
+
 /*
  * Program the contents of the download buffer to flash at the given
  * address.  Size is also specified; we shouldn't have to track usage
@@ -86,7 +105,7 @@ static int flash_status_full_check(addr_t addr, unsigned short value1, unsigned
  * them because in the context of this command they signify bugs, and
  * we want to be extra careful when writing flash.
  */
-int flash_amd_program(const u32 from, const u32 to, const u32 size)
+static int flash_amd_word_program(const u32 from, const u32 to, const u32 size)
 {
        int status;
        unsigned int loop;
@@ -121,6 +140,53 @@ int flash_amd_program(const u32 from, const u32 to, const u32 size)
        return status;
 }

+static int flash_amd_buffer_program(const u32 from, const u32 to, const u32 size)
+{
+       u32 base;
+       u32 addr;
+       u32 end;
+       u32 data;
+       size_t max_program_size;
+
+       if (from & UNALIGNED_MASK) return -H_EALIGN;
+       if (to & UNALIGNED_MASK) return -H_EALIGN;
+       if (cfi_info.max_buf_write_size < 2) return -H_ENOCMD;
+
+       base = (to & FLASH_BASE_MASK);
+       addr = to;
+       end  = to + size;
+       data = from;
```

```
+       max_program_size = 1 << cfi_info.max_buf_write_size;
+
+       while (addr < end) {
+               u32 block_start = addr & ~(max_program_size - 1);
+               /* We must not cross write block boundaries */
+               u32 next_boundary = (addr + max_program_size) & ~(max_program_size - 1);
+               u32 program_end = min(end, next_boundary);
+
+               flash_amd_unlock_seq(base);
+               flash_write_cmd(block_start, AMD_CMD_WRITE_TO_BUFFER);
+               flash_write_cmd(block_start, ((program_end - addr) >> 1) - 1);
+
+               /* program flash memory par 16 bits word */
+               for (; addr < program_end; addr+=2, data+=2) {
+                       flash_write(addr, *((u16 *)data));
+               }
+
+               flash_write_cmd(block_start, AMD_CMD_WRITE_BUFFER_CONFIRM);
+               flash_status_wait(addr - 2, *((u16 *)(data - 2)));
+       }
+       return 0;
+}
+
+int flash_amd_program(const u32 from, const u32 to, const u32 size)
+{
+       if (FORCE_WORD_PROGRAM)
+               return flash_amd_word_program(from, to, size);
+       else
+               return flash_amd_buffer_program(from, to, size);
+}
+
```

3.3.1.3. target/armadillo4x0/Kconfig or target/armadillo-box-ws1/Kconfig

新フラッシュメモリ適用品で、強制的に従来品のフラッシュメモリで動作するコンフィギュレーション "FORCE_MTDPARTS4x0" を追加しました。

"FORCE_MTDPARTS4x0"はデフォルト無効です。

Linux カーネルが"FORCE_MTDPARTS4x0"に対応していないため、このコンフィギュレーションは現時点で使用することができません。

```
--- a/src/target/armadillo4x0/Kconfig
+++ b/src/target/armadillo4x0/Kconfig
@@ -0,0 +1,7 @@
+#
+# src/target/armadillo4x0/Kconfig
+#
+
+if PLATFORM_ARMADILLO4X0
+
+endif # PLATFORM_ARMADILLO4X0
```

3.3.1.4. configs/armadillo_4x0_defconfig or armadillo_box_ws1_defconfig

新フラッシュメモリを動作させるために、"CONFIG_FLASH_AMD"を有効化しています。

```
--- a/configs/armadillo_4x0_defconfig
+++ b/configs/armadillo_4x0_defconfig
@@ -74,7 +74,7 @@ CONFIG_CONSOLE_TTYMXC1=y
 CONFIG_DEFAULT_CONSOLE="ttymxc1"
 CONFIG_STANDARD_CONSOLE="ttymxc1"
 CONFIG_FLASH=y
-# CONFIG_FLASH_AMD is not set
+CONFIG_FLASH_AMD=y
 CONFIG_FLASH_INTEL=y
 # CONFIG_FLASH_SPI is not set
 CONFIG_ETHERNET=y
```

3.3.1.5. configs/armadillo4x0_boot_defconfig or configs/ armadillo_box_ws1_defconfig

新フラッシュメモリを動作させるために、"CONFIG_FLASH_AMD"を有効化しています。

```
--- a/configs/armadillo4x0_boot_defconfig
+++ b/configs/armadillo4x0_boot_defconfig
@@ -76,7 +76,7 @@ CONFIG_DEFAULT_CONSOLE="ttymxc1"
CONFIG_STANDARD_CONSOLE="ttymxc1"
CONFIG_TTYMXC1_DTR_DSR_DCD_RI_MX25=y
CONFIG_FLASH=y
-# CONFIG_FLASH_AMD is not set
+CONFIG_FLASH_AMD=y
CONFIG_FLASH_INTEL=y
# CONFIG_FLASH_SPI is not set
CONFIG_ETHERNET=y
```

3.3.1.6. target/armadillo4x0/board.h

新フラッシュメモリ適用品を表す Board type "BOARD_TYPE_ARMADILLO411 と "BOARD_TYPE_ARMADILLO441" を追加しました。なお、従来品の Board type は "BOARD_TYPE_ARMADILLO410" または "BOARD_TYPE_ARMADILLO440"となります。

Board type は Armadillo に搭載した EEPROM 内に書き込まれており、Hermit-At 内で EEPROM を Read し Board type の判別を行います。

```
--- a/src/target/armadillo4x0/board.h
+++ b/src/target/armadillo4x0/board.h
@@ -39,6 +39,8 @@ struct board_private {
#define BOARD_TYPE_ARMADILLO410                0x0410
#define BOARD_TYPE_ARMADILLO420                0x0420
#define BOARD_TYPE_ARMADILLO440                0x0440
+#define BOARD_TYPE_ARMADILLO411                 0x0411
+#define BOARD_TYPE_ARMADILLO441                 0x0441
#define BOARD_TYPE_ARMADILLO460                0x0460
u16 hardware;   /* Hardware ID */
};
```

3.3.1.7. include/target/machine.h

新フラッシュメモリ適用品を表す machine_nr ”MACH_ARMADILLO411" と ”MACH_ARMADILLO441" を追加しました。なお、従来品の machine_nr は "MACH_ARMADILLO410" または "MACH_ARMADILLO440" となります。

Hermit-At 内で EEPROM に書き込まれている Board type を判別し、それに対応した machine_nr の設定を行います。machine_nr は ARM Linux の ATAG という仕組みを利用し、Linux カーネル側に伝えられます。Linux カーネルはこの machine_nr を元に、製品の種類を判別します。

```
--- a/include/target/machine.h
+++ b/include/target/machine.h
@@ -12,6 +12,8 @@
#define MACH_ARMADILLO440      (2374)
#define MACH_ARMADILLO460      (3270)
#define MACH_ARMADILLO410      (4636)
+#define MACH_ARMADILLO441      (5135)
+#define MACH_ARMADILLO411      (5136)
#define MACH_ARMADILLO800EVA   (3863)
#define MACH_ARMADILLO810      (4115)
#define MACH_ARMADILLO840      (4264)
```

3.3.1.8. src/target/armadillo4x0/board.c or src/target/armadillo-box-ws1/board.c

EEPROM 内に記載されているボード情報から、machine_nr を登録します。

machine_nr の値に応じて、新フラッシュメモリ適用品のフラッシュメモリメモリマップを使用するか、従来品のフラッシュメモリメモリマップを使用するかを判別します。

コンフィギュレーション"CONFIG_FORCE_MTDPARTS4x0"を設定した場合は、machine_nr の値に関係なく、従来品のフラッシュメモリメモリマップで動作します。新フラッシュメモリ適用品で従来品のメモリマップを利用した場合、setenv よって記録されたデータ構造が 32kByte まで到達した時点で bootloader 領域が消去されるので、試験目的以外ので利用は避ける事をお勧めします。

```
--- a/src/target/armadillo4x0/board.c
 +++ b/src/target/armadillo4x0/board.c
 @@ -27,11 +27,18 @@
 #include < mx25_esdhc.h >
 #include ”board.h”

 +#if defined(CONFIG_FORCE_MTDPARTS4x0)
 +int force_mtdparts4x0 = 1;
 +#else
 +int force_mtdparts4x0 = 0;
 +#endif
 +
 char target_name[256];
 char *target_profile = target_name;

 static struct board_private board_priv;
 static struct memory_device mdev_param;
 +static struct memory_map armadillo4x1_memory_map;

 #define FLASH_ADDR(offset) (FLASH_START + (offset))
 #define RAM_ADDR(offset) (DRAM_START + (offset))
```

```
@@ -371,6 +378,12 @@ static void armadillo4x0_setup_private_data(struct platform_info *pinfo)
case BOARD_TYPE_ARMADILLO460:
pinfo->machine_nr = MACH_ARMADILLO460;
break;
+case BOARD_TYPE_ARMADILLO411:
+pinfo->machine_nr = MACH_ARMADILLO411;
+break;
+case BOARD_TYPE_ARMADILLO441:
+pinfo->machine_nr = MACH_ARMADILLO441;
+break;
case BOARD_TYPE_ARMADILLO420:
/* FALL THROUGH */
default:
@@ -566,14 +579,27 @@ static void armadillo4x0_setup_flash(struct platform_info *pinfo)
{
int val;

-flash_initialize(FLASH_TYPE_INTEL, FLASH_START);
+if ((pinfo->machine_nr == MACH_ARMADILLO411 ||
+     pinfo->machine_nr == MACH_ARMADILLO441) &&
+    !force_mtdparts4x0)
+pinfo->map = &armadillo4x1_memory_map;
+
+flash_initialize_cfi(FLASH_START);

val = flash_get_size(FLASH_START);
pinfo->map->flash.size = 1 << val;
-hsprintf(pinfo->default_mtdparts,
- "armadillo4x0-nor:%p(all)ro,0x20000@0(bootloader)ro,"
- "0x400000(kernel),%p(userland),-(config)",
- pinfo->map->flash.size, pinfo->map->flash.size - 0x520000);
+if ((pinfo->machine_nr == MACH_ARMADILLO411 ||
+     pinfo->machine_nr == MACH_ARMADILLO441) &&
+    !force_mtdparts4x0)
+hsprintf(pinfo->default_mtdparts,
+ "armadillo4x0-nor:%p(all)ro,0x40000@0(bootloader)ro,"
+ "0x400000(kernel),%p(userland),-(config)",
+ pinfo->map->flash.size, pinfo->map->flash.size - 0x540000);
+else
+hsprintf(pinfo->default_mtdparts,
+ "armadillo4x0-nor:%p(all)ro,0x20000@0(bootloader)ro,"
+ "0x400000(kernel),%p(userland),-(config)",
+ pinfo->map->flash.size, pinfo->map->flash.size - 0x520000);
}

static void armadillo4x0_setup_map(struct platform_info *pinfo)
@@ -655,9 +681,11 @@ static void armadillo4x0_rev_fixup(struct platform_info *pinfo)
*   Rev.B 0x02xx
*   Rev.C 0x03xx
*/
-if (pinfo->system_rev < 0x0300) {
-memcpy(& mmcsd1_pwren_pin[0], &mmcsd1_pwren_pin_revb[0],
-       sizeof(struct iomux_info));
+if (pinfo->machine_nr != MACH_ARMADILLO411 &&
+    pinfo->machine_nr != MACH_ARMADILLO441) {
+if (pinfo->system_rev < 0x0300)
+    memcpy(&mmcsd1_pwren_pin[0], &mmcsd1_pwren_pin_revb[0],
+   sizeof(struct iomux_info));
```

```
}
}

@@ -727,6 +755,25 @@ static struct memory_map armadillo4x0_memory_map = {
.free= { RAM_ADDR(0x03000000), 0x01000000 },
};

+static struct memory_map armadillo4x1_memory_map = {
+.flash= { FLASH_ADDR(0x00000000), 0},
+.param= { FLASH_ADDR(0x00020000), 0x00008000 },
+
+/* default memory map: RAM = 64MB, (FLA = 16MB) */
+.ram= { RAM_ADDR(0x00000000), 0 },
+.boot_param= { RAM_ADDR(0x00000100), 0x00000f00 },
+.mmu_table= { RAM_ADDR(0x00004000), 0x00004000 },
+.kernel= { RAM_ADDR(0x00008000), 0x007f8000 },
+/*.hermit= { RAM_ADDR(0x00800000), 0x00300000 }, */
+/*.fec_desc= { RAM_ADDR(0x00b00000), 0x00100000 }, */
+.gunzip= { RAM_ADDR(0x00c00000), 0x00100000 },
+/*.stack(irq)= { RAM_ADDR(0x00d00000), 0x00100000 }, */
+/*.stack(svc)= { RAM_ADDR(0x00e00000), 0x00100000 }, */
+/*.vector= { RAM_ADDR(0x00f00000), 0x00100000 }, */
+.initrd= { RAM_ADDR(0x01000000), 0x02000000 },
+.free= { RAM_ADDR(0x03000000), 0x01000000 },
+};
+
static void armadillo4x0_init(void)
{
struct platform_info *pinfo = &platform_info;
@@ -750,8 +797,14 @@ static void armadillo4x0_init(void)

update_target_profile();

-memdev_add(flash, &mdev_param,
-   "hermit/param", FLASH_ADDR(0x00018000), 0x00008000);
+if ((pinfo->machine_nr == MACH_ARMADILLO411 ||
+     pinfo->machine_nr == MACH_ARMADILLO441) &&
+    !force_mtdparts4x0)
+memdev_add(flash, &mdev_param,
+   "hermit/param", FLASH_ADDR(0x00020000), 0x00008000);
+else
+memdev_add(flash, &mdev_param,
+   "hermit/param", FLASH_ADDR(0x00018000), 0x00008000);
}
arch_initcall(armadillo4x0_init);
```

3.3.2. Linux カーネルの変更点

3.3.2.1. arch/arm/configs/armadillo4x0_defconfig or arch/arm/configs/armadillo-box-ws1_defconfig

新フラッシュメモリを動作させるために、コンフィギュレーション"CONFIG_MTD_CFI_AMDSTD"を有効化しました。

```
--- a/arch/arm/configs/armadillo4x0_defconfig
+++ b/arch/arm/configs/armadillo4x0_defconfig
```

```
@@ -99,6 +99,7 @@ CONFIG_MTD_RO_IF=y
CONFIG_MTD_BLOCK=y
CONFIG_MTD_CFI=y
CONFIG_MTD_CFI_INTELEXT=y
+CONFIG_MTD_CFI_AMDSTD=y
CONFIG_MTD_PHYSMAP=y
CONFIG_BLK_DEV_LOOP=y
CONFIG_BLK_DEV_RAM=y
```

3.3.2.2. arch/arm/tools/mach-types

新フラッシュメモリ適用品を表す mach-types ”ARMADILLO411"と "ARMADILLO441" を追加しました。なお、従来品の mach-types は従来品は"ARMADILLO410" または "ARMADILLO440" となります。

この情報は、bootloader から ARM Linux の ATAG という仕組みを利用し、Linux カーネルに伝えられます。

```
-- a/arch/arm/tools/mach-types
+++ b/arch/arm/tools/mach-types
@@ -1010,3 +1010,5 @@ eukrea_cpuimx28sd MACH_EUKREA_CPUIMX28SD  EUKREA_CPUIMX28SD       4573
domotab                         MACH_DOMOTAB            DOMOTAB                 4574
pfla03                  MACH_PFLA03             PFLA03                  4575
armadillo410            MACH_ARMADILLO410       ARMADILLO410            4636
+armadillo441            MACH_ARMADILLO441       ARMADILLO441            5135
+armadillo411            MACH_ARMADILLO411       ARMADILLO411            5136
```

3.3.2.3. arm/mach-imx/mach-armadillo4x0.c or arm/mach-imx/mach-armadillo-box-ws1.c

__machine_arch_type(mach-types)の値に応じて、新フラッシュメモリ適用品のフラッシュメモリメモリマップを使用するか、従来品のフラッシュメモリメモリマップを使用するかを判別します。

```
--- a/arch/arm/mach-imx/mach-armadillo4x0.c
+++ b/arch/arm/mach-imx/mach-armadillo4x0.c
@@ -417,6 +417,30 @@ static struct mtd_partition armadillo4x0_nor_flash_partitions_32m[] = {
},
};

+static struct mtd_partition armadillo4x1_nor_flash_partitions_32m[] = {
+{
+.name= ”nor.bootloader”,
+.offset= 0x00000000,
+.size= 2 * SZ_128K,
+.mask_flags= MTD_WRITEABLE,
+}, {
+.name= ”nor.kernel”,
+.offset= MTDPART_OFS_APPEND,
+.size= 32 * SZ_128K,
+.mask_flags= 0,
+}, {
+.name= ”nor.userland”,
```

```
+.offset= MTDPART_OFS_APPEND,
+.size= 214 * SZ_128K,
+.mask_flags= 0,
+}, {
+.name= "nor.config",
+.offset= MTDPART_OFS_APPEND,
+.size= 8 * SZ_128K,
+.mask_flags= 0,
+},
+};
+
static const struct physmap_flash_data
armadillo4x0_nor_flash_pdata_16m __initconst = {
.width= 2,
@@ -431,6 +455,13 @@ static const struct physmap_flash_data
.nr_parts= ARRAY_SIZE(armadillo4x0_nor_flash_partitions_32m),
};

+static const struct physmap_flash_data
+armadillo4x1_nor_flash_pdata_32m __initconst = {
+.width= 2,
+.parts= armadillo4x1_nor_flash_partitions_32m,
+.nr_parts= ARRAY_SIZE(armadillo4x1_nor_flash_partitions_32m),
+};
+
static const struct resource
armadillo4x0_nor_flash_resource __initconst = {
.flags= IORESOURCE_MEM,
@@ -502,7 +533,10 @@ static void __init armadillo420_init_mtd(void)
#if defined(CONFIG_MACH_ARMADILLO440) || defined(CONFIG_MACH_ARMADILLO410)
static void __init armadillo440_init_mtd(void)
{
-armadillo4x0_init_mtd(armadillo4x0_nor_flash_pdata_32m);
+if (machine_is_armadillo441() || machine_is_armadillo411())
+armadillo4x0_init_mtd(armadillo4x1_nor_flash_pdata_32m);
+else
+armadillo4x0_init_mtd(armadillo4x0_nor_flash_pdata_32m);
}
#endif

@@ -584,6 +618,18 @@ MACHINE_START(ARMADILLO410, "Armadillo-410")
.init_machine= armadillo440_init,
.restart= mxc_restart,
MACHINE_END
+
+MACHINE_START(ARMADILLO411, "Armadillo-410")
+/* Maintainer: Atmark Techno, Inc.  */
+.atag_offset= 0x100,
+.map_io= mx25_map_io,
+.init_early= imx25_init_early,
+.init_irq= mx25_init_irq,
+.handle_irq= imx25_handle_irq,
+.init_time= armadillo4x0_timer_init,
+.init_machine= armadillo440_init,
+.restart= mxc_restart,
+MACHINE_END
#endif
```

```
#if defined(CONFIG_MACH_ARMADILLO420)
@@ -612,4 +658,16 @@ MACHINE_START(ARMADILLO440, "Armadillo-440")
.init_machine= armadillo440_init,
.restart= mxc_restart,
MACHINE_END
+
+MACHINE_START(ARMADILLO441, "Armadillo-440")
+/* Maintainer: Atmark Techno, Inc.  */
+.atag_offset= 0x100,
+.map_io= mx25_map_io,
+.init_early= imx25_init_early,
+.init_irq= mx25_init_irq,
+.handle_irq= imx25_handle_irq,
+.init_time= armadillo4x0_timer_init,
+.init_machine= armadillo440_init,
+.restart= mxc_restart,
+MACHINE_END
#endif
```

3.4. ソフトウェアから新フラッシュメモリ適用品か判別する方法

3.4.1. Hermit-At のソースコード内で判別する方法

src/target/armadillo4x0/board.c の以下の処理を参考にしてください。"pinfo->machine_nr == MACH_ARMADILLO411" あるいは "pinfo->machine_nr == MACH_ARMADILLO441" が真であれば新フラッシュメモリ適用品となります。

```
static void armadillo4x0_init(void)
{
※ 省略 ※
    if ((pinfo->machine_nr == MACH_ARMADILLO411 ||
         pinfo->machine_nr == MACH_ARMADILLO441) &&
        !force_mtdparts4x0)
        memdev_add(flash, &mdev_param,
               "hermit/param", FLASH_ADDR(0x00020000), 0x00008000);
    else
        memdev_add(flash, &mdev_param,
               "hermit/param", FLASH_ADDR(0x00018000), 0x00008000);
}
```

3.4.2. 保守モードの Hermit-At コマンドで判別する方法

Hermit-At の info コマンド、または memmap コマンドから判別が可能です。

新フラッシュメモリ適用品で info コマンドを実行すると、次のように"Board Type"が 0x00000411 もしくは 0x00000441 となります。

```
hermit> info
 Board Type: 0x00000411
Hardware ID: 0x00000100
    DRAM ID: 0x00000002
     Jumper: 0x00000000
    Tact-SW: 0x00000001
```

```
 ORIG MAC-1: 00:11:0c:27:00:03
Base Board Gen: 0x00000001
```

新フラッシュメモリ適用品で memmap コマンドを実行すると、次のように"bootloader"領域が"0xa0000000:0xa003ffff"となります。hermit-at-2.x.x の場合は「2.2. メモリマップ変更点(hermit-at + linux-2.6.26-at の場合)」の値となります。

```
hermit> memmap
0xa0000000:0xa1ffffff FLA all bf:8K bl:256x128K/l
0xa0000000:0xa003ffff FLA bootloader bf:8K bl:2x128K/l
0xa0040000:0xa043ffff FLA kernel bf:8K bl:32x128K
0xa0440000:0xa1efffff FLA userland bf:8K bl:214x128K
0xa1f00000:0xa1ffffff FLA config bf:8K bl:8x128K
0x80000000:0x87ffffff RAM dram-1
```

従来品で info コマンドを実行すると、次のように"Board Type"が 0x00000410 もしくは 0x00000440 となります。

```
hermit> info
 Board Type: 0x00000410
Hardware ID: 0x000003ff
    DRAM ID: 0x00000002
     Jumper: 0x00000000
    Tact-SW: 0x00000001
 ORIG MAC-1: 00:11:0c:18:0b:ff
Base Board Gen: 0x00000001
```

従来品で memmap コマンドを実行すると、次のように"bootloader"領域が"0xa0000000:0xa001ffff"となります。hermit-at-2.x.x の場合は「2.2. メモリマップ変更点(hermit-at + linux-2.6.26-at の場合)」の値となります。

```
hermit> memmap
0xa0000000:0xa1ffffff FLA all bf:8K bl:4x32K/l,255x128K/l
0xa0000000:0xa001ffff FLA bootloader bf:8K bl:4x32K/l
0xa0020000:0xa041ffff FLA kernel bf:8K bl:32x128K
0xa0420000:0xa1efffff FLA userland bf:8K bl:215x128K
0xa1f00000:0xa1ffffff FLA config bf:8K bl:8x128K
0x80000000:0x87ffffff RAM dram-1
```

3.4.3. Linux カーネルのソースコード内で判別する方法

arch/arm/mach-imx/mach-armadillo4x0.c あるいは mach-armadillo-box-ws1.c の処理を参考にしてください。"machine_is_armadillo411()" あるいは "machine_is_armadillo441()" が真であれば新フラッシュメモリ適用品となります。

3.4.4. アプリケーションから判別する方法

/proc/mtd の内容を確認し、bootloader 領域の size から判別が可能です。

新フラッシュメモリ適用品では次に示すように、"nor.bootloader"の"size"が 00040000 となります。

```
[root@armadillo440-0 (ttymxc1) ~]# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00040000 00020000 "nor.bootloader"
mtd1: 00400000 00020000 "nor.kernel"
mtd2: 01ac0000 00020000 "nor.userland"
mtd3: 00100000 00020000 "nor.config"
```

従来品では次に示すように、"nor.bootloader"の"size"が 00020000 となります。

```
[root@armadillo440-0 (ttymxc1) ~]# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00020000 00008000 "nor.bootloader"
mtd1: 00400000 00020000 "nor.kernel"
mtd2: 01ae0000 00020000 "nor.userland"
mtd3: 00100000 00020000 "nor.config"
```

4. 付録 A 新フラッシュメモリ適用品向けアップデートの差分の適用例

ここでは、以下の Web ページからダウンロードできるパッチファイルの中から、例として hermit-at-2.1.5 にパッチを適用する例を紹介します。

新フラッシュメモリ適用品向けアップデートの差分ダウンロード

https://download.atmark-techno.com/misc/new_flash_memory/

一連の操作は ATDE の GNOME 端末上で行います。また、ATDE や関連するソフトウェア(シリアル通信ソフトウェアなど)については、設定が完了している前提で説明しています。各製品の製品マニュアル、ソフトウェアマニュアルをご覧いただき、関連ソフトウェアの設定を完了してから操作を行ってください。

4.1. ソースコードのバックアップ

後でパッチを当てたことによって、実際にどのような変更が行われたのかを把握しやすくするため、ソースコードのバックアップします。

```
[ATDE ~]$ cp -r hermit-at-2.1.5-source hermit-at-2.1.5-source.backup
```

図 4.1 hermit-at-2.1.5-source のバックアップ

4.2. パッチの適用

4.2.1. パッチコマンドの実行

パッチファイルの適用には patch コマンドを使用します。まずは以下のように patch コマンド実行します。

```
[ATDE ~]$ ls
hermit-at-2.1.5-source          new_flash_memory_hermit-at-2.x.x.patch
hermit-at-2.1.5-source.backup
[ATDE ~]$ cd hermit-at-2.1.5-source
[ATDE ~/hermit-at-2.1.5-source]$ patch -p 1 < ../new_flash_memory_hermit-at-2.x.x.patch
patching file configs/armadillo4x0_boot_defconfig
patching file configs/armadillo4x0_defconfig
patching file include/target/flash.h
patching file include/target/machine.h
Hunk #1 FAILED at 12.
1 out of 1 hunk FAILED -- saving rejects to file include/target/machine.h.rej
patching file src/target/armadillo4x0/Kconfig
patching file src/target/armadillo4x0/board.c
Hunk #2 succeeded at 370 (offset -3 lines).
Hunk #3 succeeded at 572 (offset -3 lines).
Hunk #4 succeeded at 674 (offset -3 lines).
Hunk #5 succeeded at 772 (offset -3 lines).
patching file src/target/armadillo4x0/board.h
Hunk #1 succeeded at 38 with fuzz 1 (offset -1 lines).
patching file src/target/driver/flash_amd.c
patching file src/target/driver/flash_core.c
```

図 4.2 hermit-at-2.1.5 へのパッチの適用

「図 4.2. hermit-at-2.1.5 へのパッチの適用」では patch コマンドが失敗していることがわかります。

4.2.2. パッチを適用できなかった箇所の確認

patch コマンドが失敗した場合、パッチを適用できなかった箇所を、テキストエディタで適切に修正する必要があります。

パッチを適用できなかった箇所は、「パッチが適用されるファイル名」.rej というファイルに記述されています。

```
[ATDE ~/hermit-at-2.1.5-source]$ cat include/target/machine.h.rej
--- include/target/machine.h
+++ include/target/machine.h
@@ -12,5 +12,7 @@
 #define MACH_ARMADILLO440    (2374)
 #define MACH_ARMADILLO460    (3270)
 #define MACH_ARMADILLO410    (4636)
+#define MACH_ARMADILLO441    (5135)
+#define MACH_ARMADILLO411    (5136)

 #endif /* _HERMIT_TARGET_MACHINE_H_ */
```

図 4.3 include/target/machine.h.rej の内容

「図 4.3. include/target/machine.h.rej の内容」と include/target/machine.h を見比べると、"#define MACH_ARMADILLO410 (4636)" の行が無かった(Armadillo-410 対応前のソースコードだった)ため、パッチを適用できなかったことがわかります。

4.2.3. パッチを適用できなかった箇所のソースコードの修正

パッチを適用できなかった箇所がある場合、生成された rej ファイルに記述された差分を確認し、ソースコードに適用すべきかを判断する必要があります。

hermit-at-2.1.5-source は、Armadillo-410 対応前のソースコードです。そのため、"#define MACH_ARMADILLO410 (4636)"が存在しなくても問題はありません。したがって、include/target/machine.h を以下のように修正します。

```
[ATDE ~/hermit-at-2.1.5-source]$ vi include/target/machine.h
#ifndef _HERMIT_TARGET_MACHINE_H_
#define _HERMIT_TARGET_MACHINE_H_

#define MACH_ARMADILLO        (83)
#define MACH_ARMADILLO9        (386)
#define MACH_ARMADILLOJ        (416)
#define MACH_ARMADILLO300    (473)
#define MACH_ARMADILLO500    (1260)
#define MACH_ARMADILLO500FX    (1918)
#define MACH_ARMADILLO420    (2373)
#define MACH_ARMADILLO440    (2374)
#define MACH_ARMADILLO460    (3270)
#define MACH_ARMADILLO441    (5135) // 追加
#define MACH_ARMADILLO411    (5136) // 追加

#endif /* _HERMIT_TARGET_MACHINE_H_ */
```

図 4.4 include/target/machine.h の修正

4.3. 変更内容の確認

patch コマンドが成功していたとしても、意図したとおりにソースコードが変更されているかはわかりません。そのため、パッチ適用後のソースコードが「3.3. 新フラッシュメモリ適用に伴うソフトウェア変更箇所」および使用したパッチファイルの意図したとおりになっているか確認してください。

変更されたファイルは「図 4.2. hermit-at-2.1.5 へのパッチの適用」に表示されています。

パッチ適用前後のソースコードの差分は、diff コマンドで確認できます。patch コマンドにより、実際のソースコードには .orig ファイルと .rej ファイルが含まれていますが、ビルドには影響しないため無視してください。

```
[ATDE ~/hermit-at-2.1.5-source]$ cd ../
[ATDE ~]$ diff -ur hermit-at-2.1.5-source.backup hermit-at-2.1.5-source
```

図 4.5 パッチ適用前後のソースコードの差分

正しく適用されたのかどうか判断ができない場合は、ソースコードのバージョンと「図 4.5. パッチ適用前後のソースコードの差分」の実行結果を添付して Armadillo フォーラムへお問い合せください。

Armadillo フォーラム

https://users.atmark-techno.com/forum/armadillo

4.4. イメージファイルのビルド

ビルド方法については、各製品の製品マニュアル、ソフトウェアマニュアルをご覧ください。

新フラッシュメモリ対応のイメージをビルドする上で、手順の追加はありません。ただし、次の点にご注意ください。

4.4.1. デフォルトコンフィギュレーションを適用してからビルドする

新フラッシュメモリ対応のイメージをビルドする際は、必ずデフォルトコンフィギュレーションを適用してから make を実行してください。

新フラッシュメモリ対応ではデフォルトコンフィギュレーションの変更を行っています。そのため、ソースコードの修正後、デフォルトコンフィギュレーションを適用せずにイメージをビルドすると、新フラッシュメモリ非対応のイメージが生成されてしまいます。

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2017/12/09	・初版発行
1.1.0	2018/06/28	・Armadillo-460 に対応 ・「4. 付録 A 新フラッシュメモリ適用品向けアップデートの差分の適用例」を追加

Armadillo-400 シリーズ新フラッシュメモリ移行ガイド
Version 1.1.0
2018/07/10