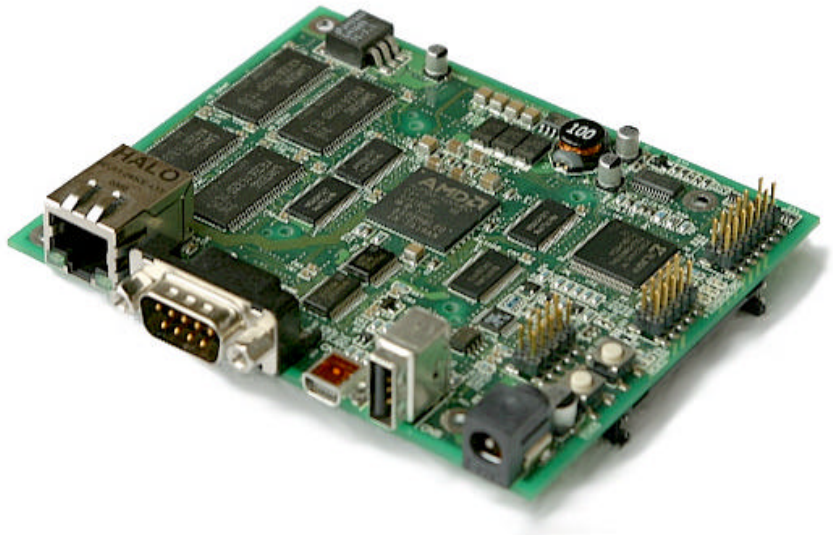
株式会社デバイスドライバーズ

組込みLinuxボード開発手法

E!Kit-1100

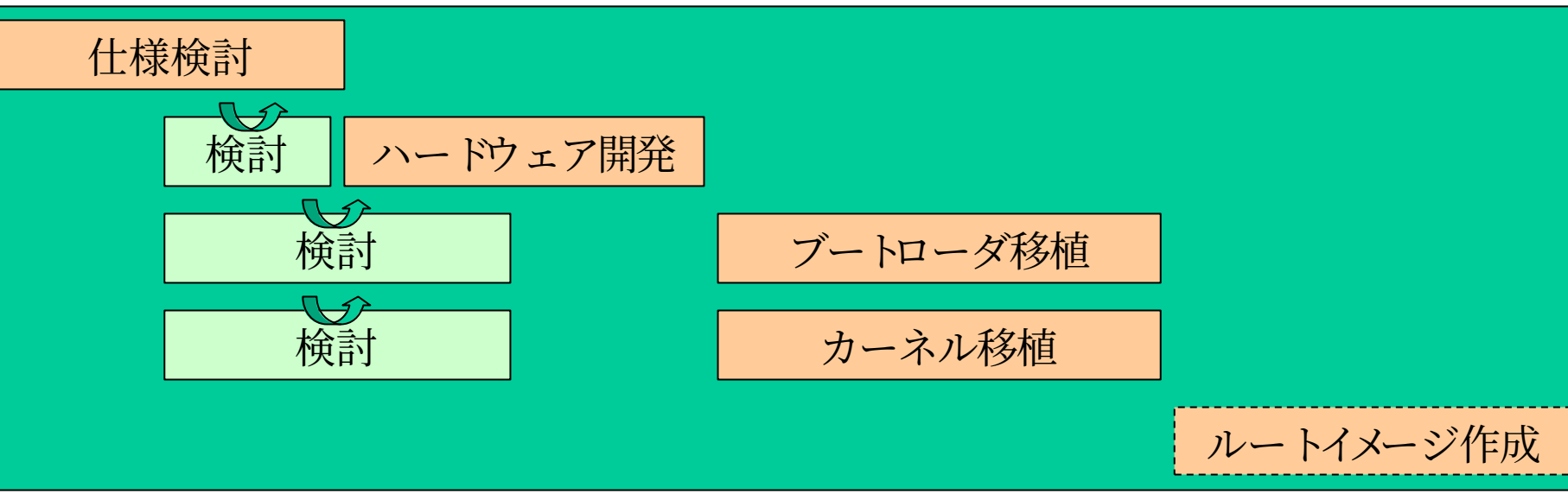
- Au1100™-400MHz
- シリアル、USB、CF、LAN標準搭載
- LCD、AC97、SDなど拡張コネクタで増設可能
- 110×90mm（名刺2枚分）
- EJTAG搭載



AMD Alchemy™ Au1100™

- MIPS32™CPUコア
- 高集積システムペリフェラル
  - SDRAM/SRAM/Flash/PCMCIAコントローラ
  - LCDコントローラ
  - 10/100Ethernetコントローラ
  - USBホスト/デバイス
  - UART
  - GPIO(48本)など
- 低消費電力（400MHz時で250［mW］以下）

Linuxボード開発



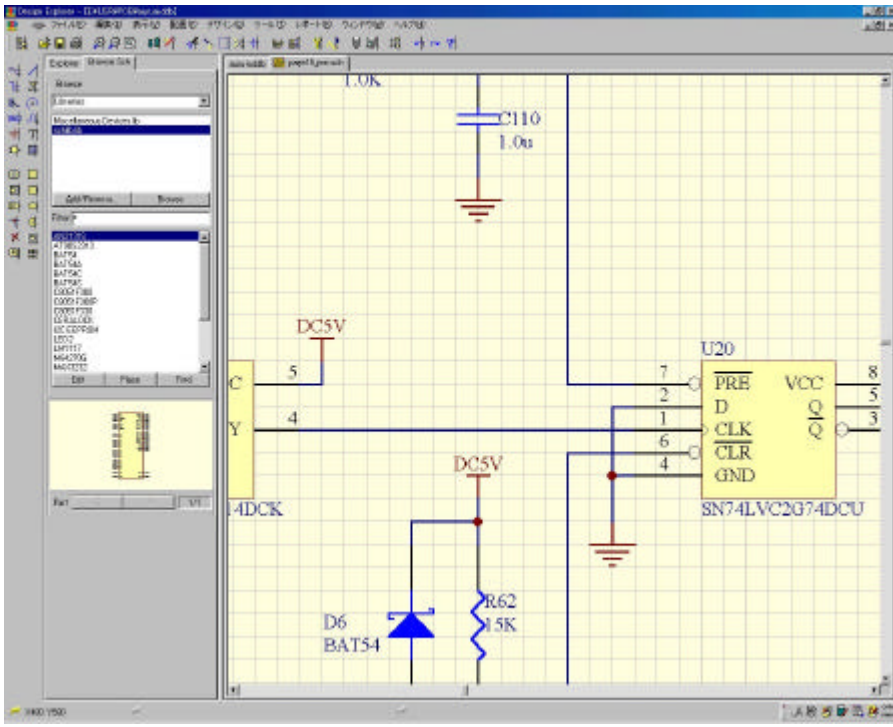
ブートローダやカーネルの検討を、仕様、ハードウェアの段階で行いフィードバックする！

仕様検討

- スペックの検討
  - CPU（種類、動作周波数・・・）
  - Flashメモリ（容量、バス幅・・・）
  - メインメモリ（容量、バス幅・・・）
  - 搭載機能（LAN、USB・・・）
  - ボードサイズ
- 部品の検討
  - 価格
  - 入手性

ハードウェア開発

- 回路設計
  - 部品、回路
- ボード設計
  - 外形、AW
- 製造
- 評価
  - 電圧、波形確認
  - 機能確認

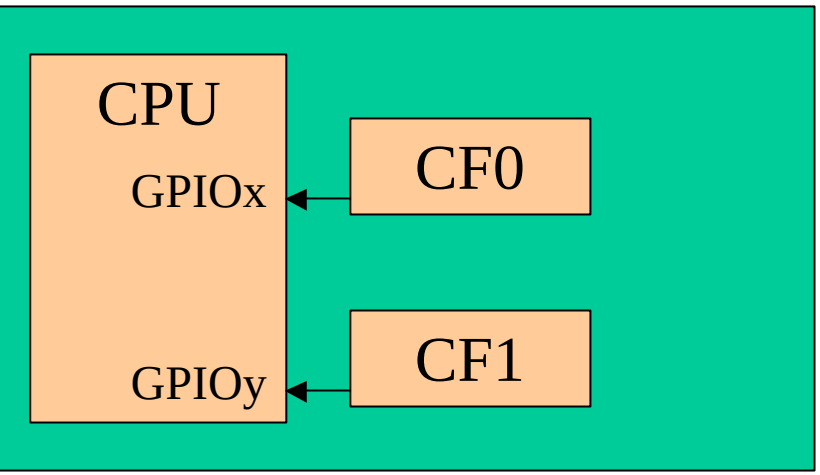


部品選定(Flashメモリ)

- 容量
- データ幅（x8、x16、x32）
- インタフェース（バースト、ページ・・・）
- セクタタイプ（ユニフォーム、ブート）
- 制御コマンド（AMD、Intel互換・・・）
- 動作電圧、アクセス速度・・・

回路設計(CFソケット割込み)

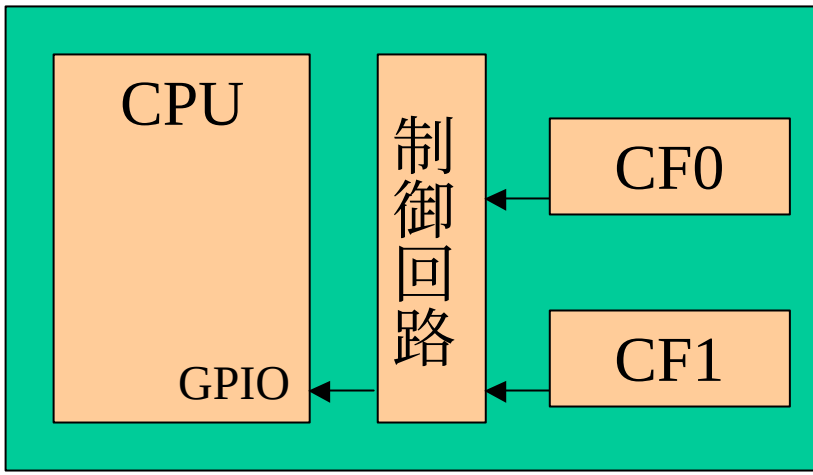
割込み信号が分離



GPIOの消費は2本

Linuxの移植が容易

割込み信号が共用



GPIOの消費は1本

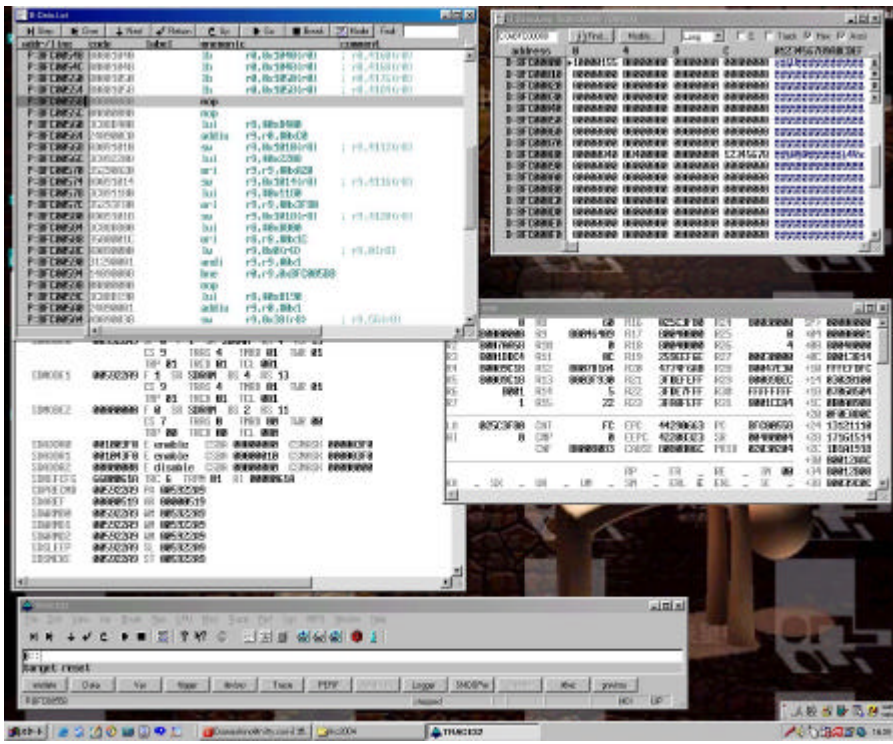
Linuxの移植が困難

drivers/pcmcia/au1000_ekit1100.c

```
static int ekit1100_pcmcia_init(struct pcmcia_init *init)
static int ekit1100_pcmcia_shutdown(void)
static int ekit1100_pcmcia_socket_state(unsigned sock, struct pcmcia_state *state)
static int ekit1100_pcmcia_get_irq_info(struct pcmcia_irq_info *info)
{
        if(info->sock > PCMCIA_MAX_SOCK) return -1;
        if(info->sock == 0)
                info->irq = AU1000_GPIO_0;
        else
                info->irq = AU1000_GPIO_1;
        return 0;
}
static int ekit1100_pcmcia_configure_socket(const struct pcmcia_configure *configure)
```

ブートローダ移植

- ソース変更
- コンパイル
- Flashメモリ書込み
- デバッグ



ブートローダ

- Yamon™
  - MIPS Technologies, Inc.
- RedBoot
  - RedHat, Inc.
- U-Boot
  - http://sourceforge.net/projects/u-boot/

init/reset/reset.S

```
        b       reset_au1xxx    /* 0xBFC00000 */
......
reset_au1xxx:
......
#ifdef EKIT1100_CONFIG
#include "../arch/init/reset_ekit1100.S"
#endif
......
yamon_le:
        la      k0, _reset_handler_le
        KSEG1A(k0)
        j       k0
```

リセットベクタ

ボード依存の初期化ルーチンのインクルード

エンディアン毎の初期化ルーチンへジャンプ

arch/init/reset_ekit1100.S

■CPUエンディアン設定

■CPU各種レジスタ初期化

■EJTAG Debugレジスタ初期化

■キャッシュ初期化

■TLB初期化

■CPU PLL初期化

■システムバス分周比初期化

■AUX PLL初期化

■32KHz発信器初期化

■スタティックメモリコントローラ初期化

■各種ペリフェラル初期化

■SDRAMコントローラ初期化

■各種信号線初期化（信号の選択、方向など）

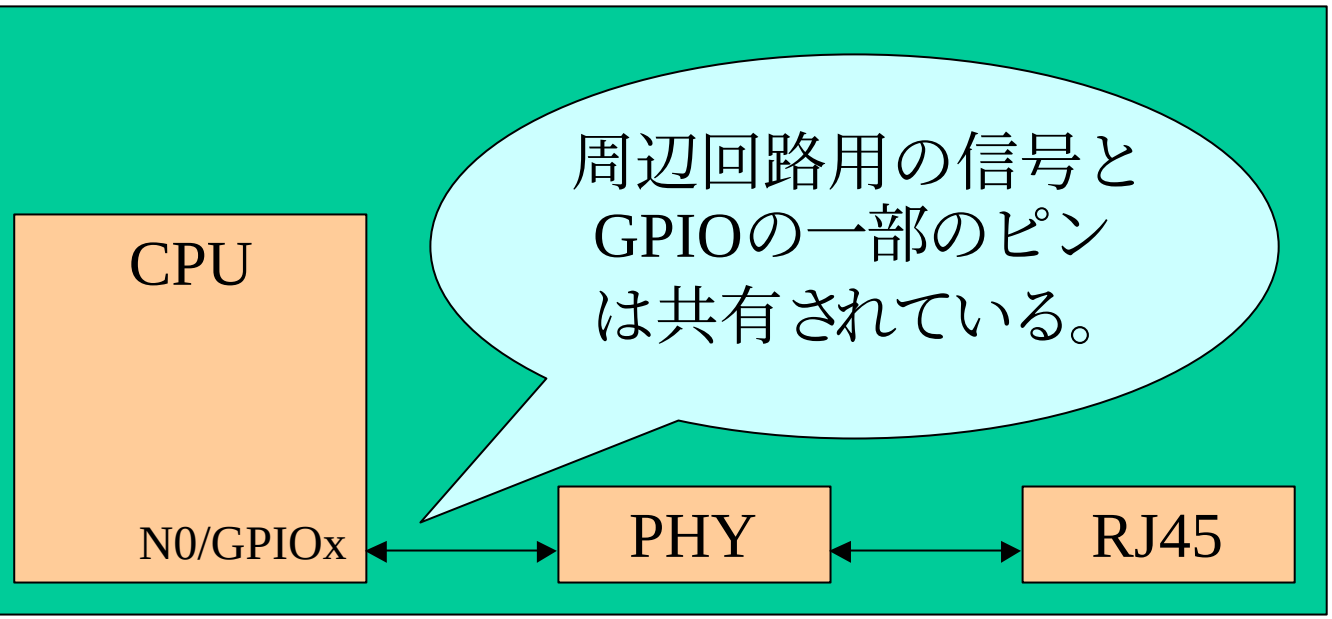
init/init.S

```
.....
        lw      t3, 0(t0)
        add     t0, 4
        sw      t3, 0(t1)
.....
        li      t0, 4*4
        subu    sp, t0
        la      t0, c_entry
        jalr    t0
```

コードや初期化データを
RAM領域にコピー

RAM領域にコピーされたCの
c_entry関数へジャンプ

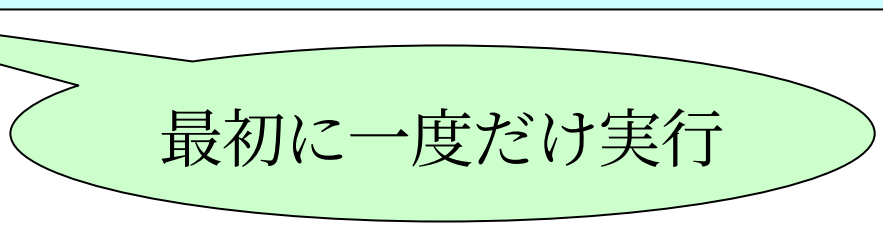
各種信号線初期化



ブートローダで初期化が必要！

Yamon™のコンパイル

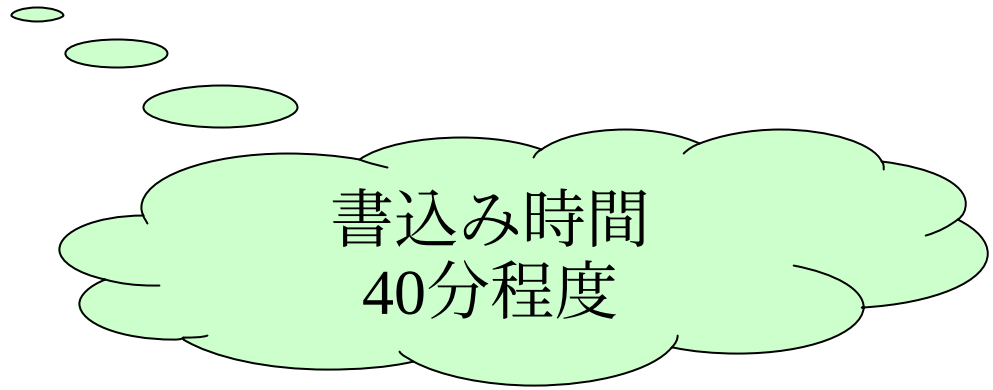
```
$ make install
mkdir EL
mkdir EB
```




```
$ make
```

Yamon™の書込み

```
FLASH.PROGRAM 0xBFC00000--0xBFCFFFFF
DATA.SET 0xA0100000--0xA01FFFFF 0xFF
DATA.LOAD.S3RECORD yamon¥bin¥yamon-02.19EKIT1100-070504.rec.m.EB
DATA.COPY 0xA0100000--0xA01FFFFF 0xBFC00000 /WORD
```



Yamon™の起動

```
IO

EXCEP

......

YAMON ROM Monitor, Revision 02.19EKIT1100-012704.
......
Flash memory size =                 8 MByte

SDRAM size =                        128 MByte

First free SDRAM address =          0x8008f0bc

......

YAMON>
```

RAMテスト

```
YAMON> test ram
Testing RAM
Memory test from 0xA008F100 to 0xA7FFFFFC, 10 loops.
Press Ctrl-C to break
Now running loop 10 |
Test passed
YAMON>
```

KSEG1領域

非キャッシュ、非TLB

Flashメモリテスト

```
YAMON> test flash
Testing Flash
Test passed
YAMON>
```

Yamon™の設定

```
YAMON> setenv ethaddr 00.0e.6c.00.00.09
YAMON> setenv ipaddr 192.168.1.150
YAMON> setenv subnetmask 255.255.255.0
```

```
YAMON> setenv bootserver 192.168.1.201
YAMON> setenv bootfile ekit1100.srec
```

Linuxカーネル移植

- ソース変更
- コンパイル
- デバッグ

arch/mips/zboot/pb1xxx/head.S

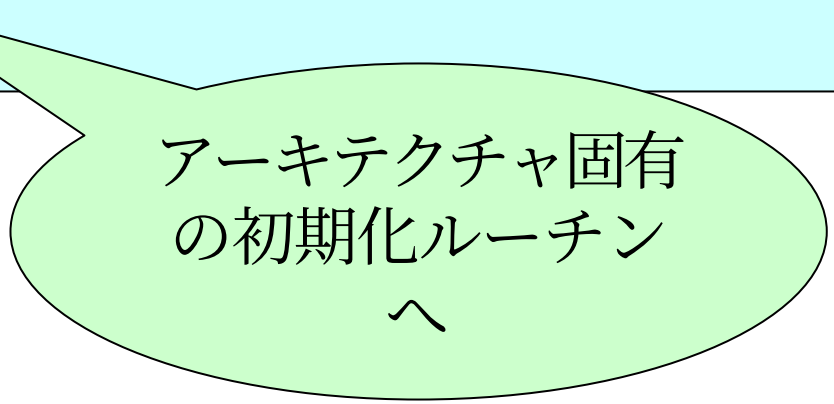
```
.....
        la      k0, decompress_kernel
        jr      k0
.....
        li      k0, KERNEL_ENTRY
        jr      k0
.....
```

圧縮カーネルの展開

展開したカーネルへジャンプ

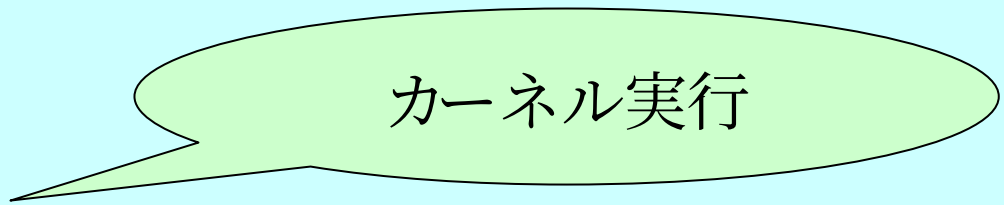
arch/mips/kernel/head.S

```
......

        jal     init_arch

......
```



arch/mips/kernel/setup.c

```
asmlinkage void __init init_arch(int argc, char**argv, char**envp, int*prom_vec)
{
        cpu_probe();
        prom_init(argc, argv, envp, prom_vec);
        cpu_report();
        load_mmu();
……
        start_kernel();
}
```



init/main.c

```
asmlinkage void __init start_kernel(void)
{
        char * command_line;
        extern char saved_command_line[];
        lock_kernel();
        printk(linux_banner);
        setup_arch(&command_line);
        printk("Kernel command line: %s¥n", saved_command_line);
        parse_options(command_line);
……
}
```

アーキテクチャ固有の
初期化ルーチン

arch/mips/kernel/setup.c

```
void __init setup_arch(char **cmdline_p)
{
......
#ifdef CONFIG_SOC_AU1X00
        case MACH_GROUP_ALCHEM
                au1x00_setup();
                break;
#endif
......
}
```

arch/mips/au1000/common/setup.c

```
void __init au1x00_setup(void)
{
......
        board_setup();
......
}
```

ボード固有の初期化
ルーチン

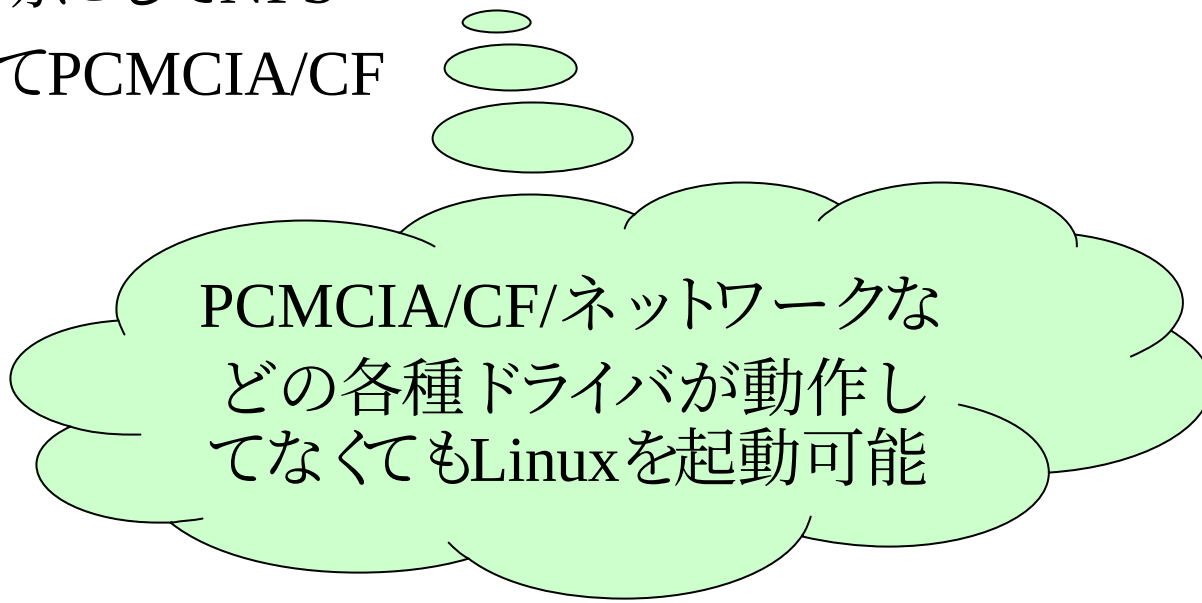
arch/mips/au1000/ekit1100/board_setup.c

```
void __init board_setup(void)
{
#if defined(CONFIG_MMC)
        au_writel((1 << 31)|(1 << 30)|(1 << 20)|(1 << 21), SYS_TRIOUTCLR);
        au_sync();
        au_writel((1 << 22)|(1 << 23), SYS_OUTPUTSET);
        au_sync();
#endif
……
        printk("Device Drivers Limited E!Kit-1100 Board¥n");
}
```

SD/MMCコネクタ用の
信号線（GPIO）初期化

initrd

- 圧縮イメージを展開しRAMDISKとしてマウント
- 必要な処理を行いルートファイルシステムを再マウント
  - ネットワークを有効にしてNFS
  - cardmgrを実行してPCMCIA/CF
  - ・・・・・・



initrdの作成

- BusyBoxのコンパイル
- ramdisk.gz作成
- Linuxカーネルへの組込み

BusyBox

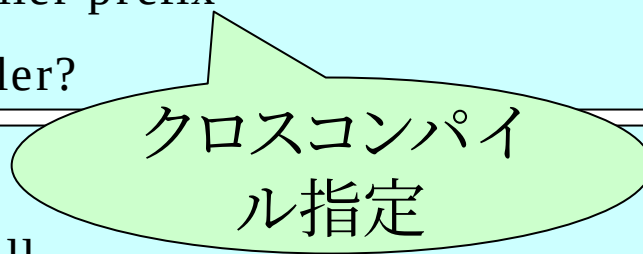
- 組込みLinux用コマンド郡
- 一つの実行ファイルで複数のコマンドに対応
- 小容量
  - 各種サーバ機能(httpdなど)を入れても約1.5MB
  - gzipで圧縮すれば約500KB

BusyBoxのコンパイル

```
[root@pen4 rootfs]# make menuconfig
```

```
Build Options  --->
        [*] Build BusyBox as a static binary (no shared libs)
        [ ] Build with Large File Support (for accessing files > 2 GB)
        [*] Do you want to build BusyBox with a Cross Compiler?
        (/export/local/bin/mipsel-linux-) Cross Compiler prefix
        ()  Any extra CFLAGS options for the compiler?
```

```
[root@pen4 rootfs]# make
[root@pen4 rootfs]# make PREFIX=linux/initrd install
```



initrd/etc/init.d/rcS

```
#!/bin/sh

/bin/mount -t proc /proc /proc

exit 0
```

ramdisk.gz作成

```
/bin/dd if=/dev/zero of=ramdisk bs=1k count=4k
/sbin/mke2fs -F ramdisk
/bin/mount -o loop ramdisk /misc
/bin/cp -a initrd/* /misc/
/bin/umount /misc
/usr/bin/gzip -f -9 ramdisk
```

```
mv ramdisk.gz linux/arch/mips/ramdisk/.
```

カーネルへの組込み

```
[root@pen4 rootfs]# make menuconfig
```

```
Block devices  --->
        <*> RAM disk support
        (4096)   Default RAM disk size
        [*]   Initial RAM disk (initrd) support
MIPS initrd options  --->
        [*]   Embed root filesystem ramdisk into the kernel
        Filename of gziped ramdisk image: "ramdisk.gz"
```

```
[root@pen4 rootfs]# make zImage
[root@pen4 rootfs]# cp linux/arch/mips/zboot/images/ekit1100.srec /tftpboot/.
```

カーネルのロード

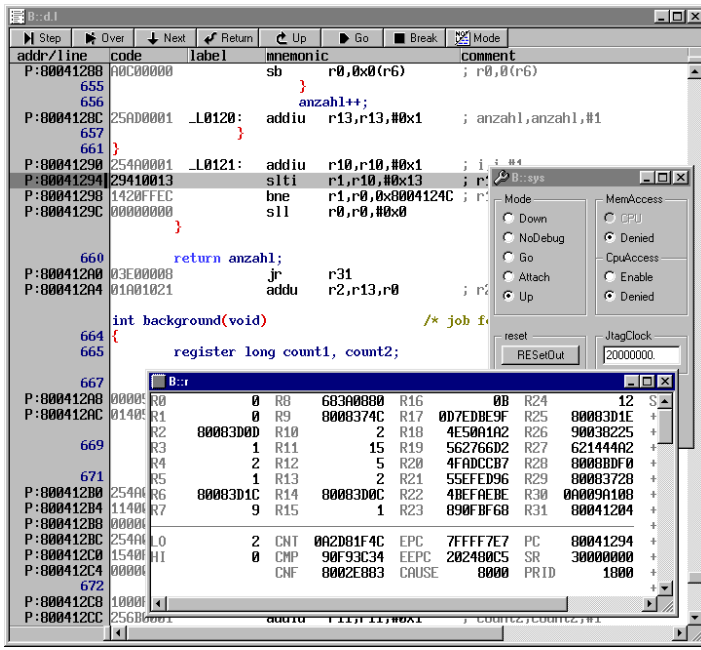
```
YAMON> load
About to load tftp://192.168.1.201/ekit1100.srec
Press Ctrl-C to break
Start dump from terminal program
.....
.....
.....
Start = 0x81000000, range = (0x81000000,0x810d6fff), format = SREC
YAMON>
```

カーネルの起動

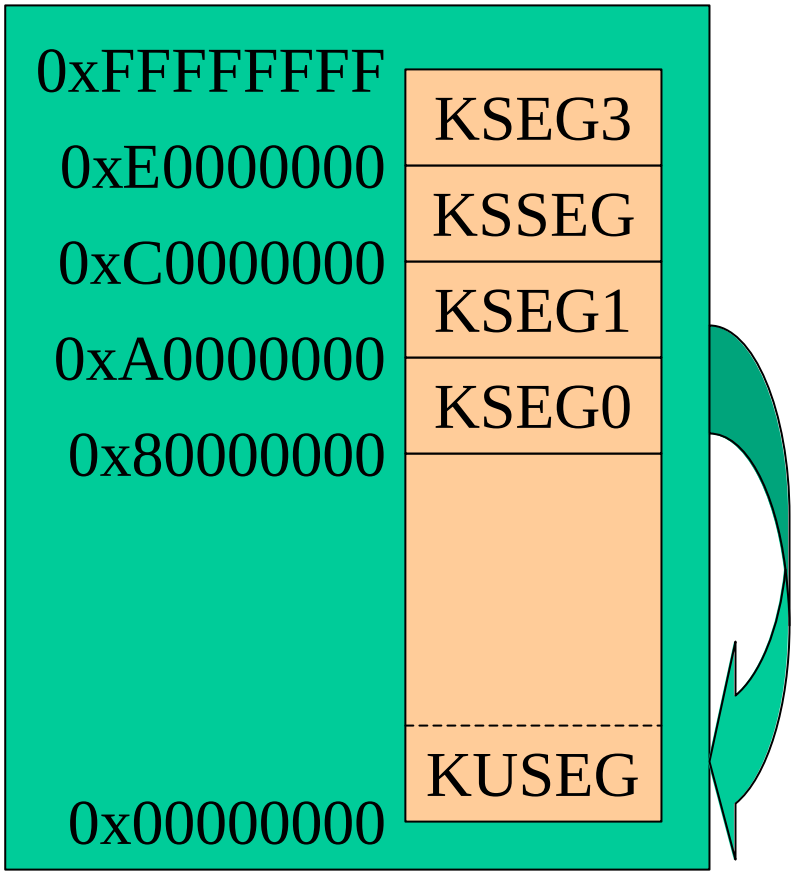
```
YAMON> go
CPU revision is: 02030204
Primary instruction cache 16kB, physically tagged, 4-way, linesize 32 bytes.
......
VFS: Mounted root (ext2 filesystem) readonly.
Freeing unused kernel memory: 112k freed
Algorithmics/MIPS FPU Emulator v1.5
Please press Enter to activate this console.
BusyBox v1.00-pre10 (2004.06.14-08:56+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.
-sh: can't access tty; job control turned off
/ #
```

補足 :JTAGデバッガ

- 独ロータバッハ社製JTAG Debuggerシリーズ
- 対応CPU
  - ARM, MIPS, PPC, SH, H8, i186, 各種DSP, FPGA
- 提供機能
  - ハードウェアデバッグ
  - ソフトウェア・シミュレータ
  - ROMエミュレータ
  - **オンボードFlashメモリの書込み**



補足 :MIPS32™の論理アドレス

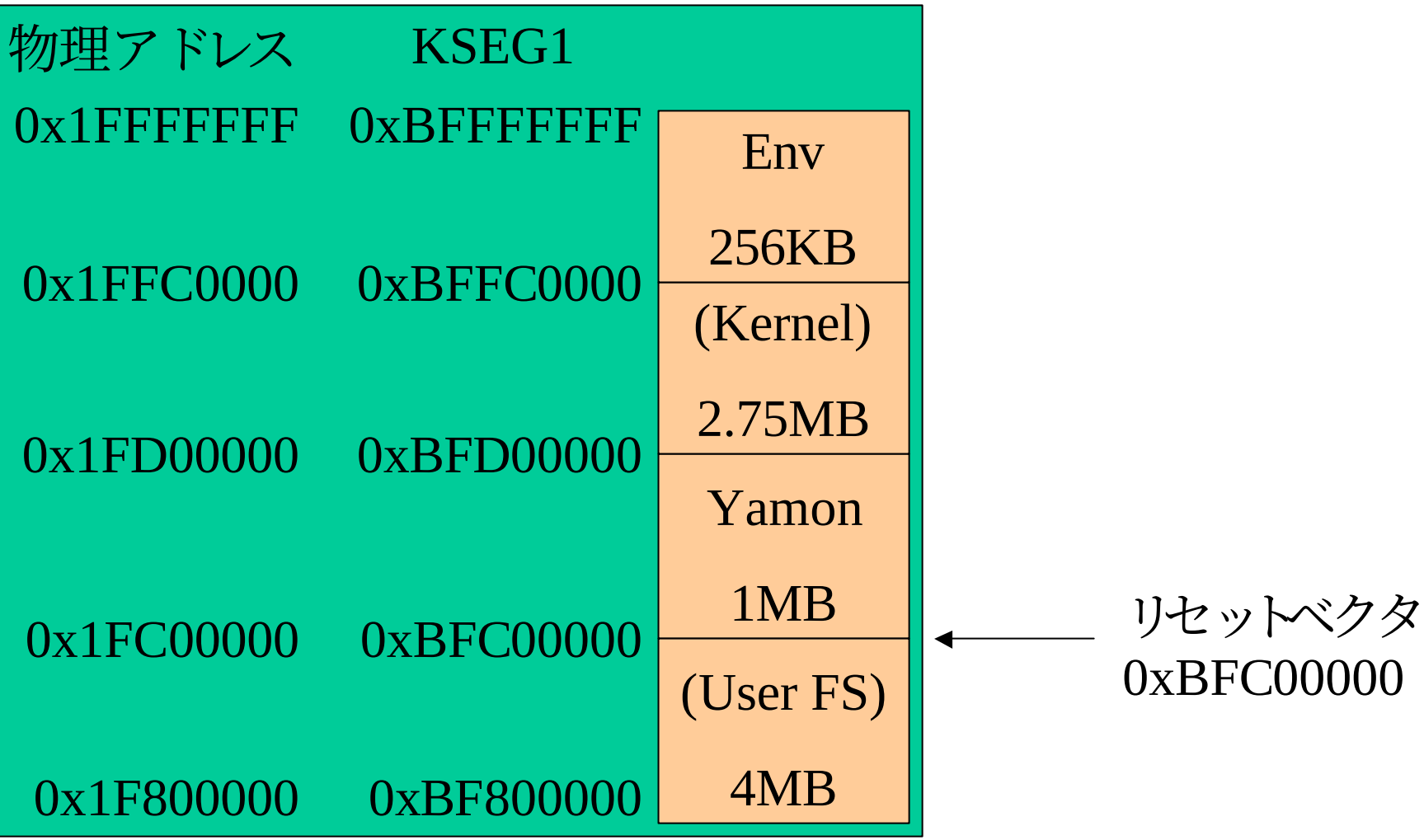


領域	キャッシュ	TLB
KSEG1	×	×
KSEG0	○	×
KUSEG	○	○

KSEG0、KSEG1は物理アドレス0x00000000 - 0x1FFFFFFFにマップ

Au1100™はKSSEG、KSEG3が予約領域になっている

補足 :Flashメモリマップ



補足 :情報のポインタ

- E!Kit-1100
  - http://e-kit.jp/products/E!Kit1100/
- Linux/MIPS
  - http://www.linux-mips.org/
  - http://www.linux.or.jp/JF/JFdocs/MIPS-HOWTO.html
- BusyBox
  - http://www.busybox.net/

補足 :GPIOアサイン

GPIO0	CF0割込み	GPIO8	拡張コネクタ(I2SDI)
GPIO1	CF1割込み	GPIO9	UART3(U3CTS)
GPIO2	拡張コネクタ(EXTCLK0)	GPIO10	UART3(U3DSR)
GPIO3	CPLDクロック(EXTCLK1)	GPIO11	UART3(U3DCD)
GPIO4	拡張コネクタ(DMA_REQ0)	GPIO12	UART3(U3RI)
GPIO5	拡張コネクタ(DMA_REQ1)	GPIO13	UART3(U3RTS)
GPIO6	CPLD(SMROMCKE)	GPIO14	UART3(U3DTR)
GPIO7	ソフトウェアリセット	GPIO15	拡張コネクタ(IRFIRSEL)

他の信号とピンが共有

専用のピン

補足 :GPIOアサイン(続き)

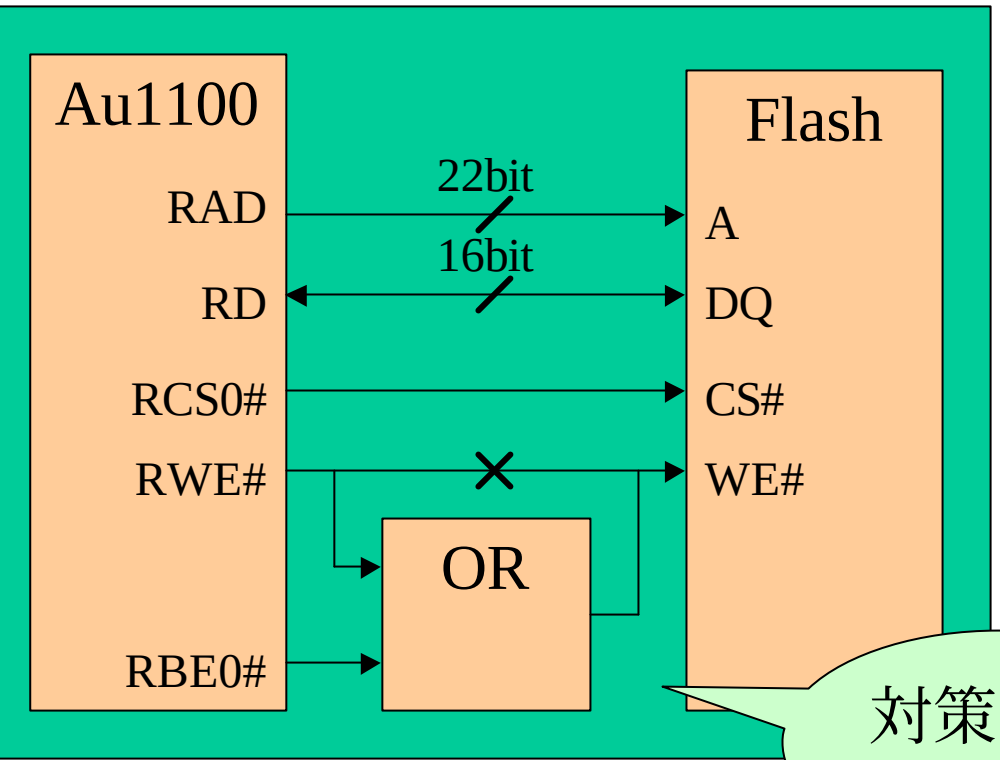
GPIO16	CF電源	GPIO24	PHY(N0TXEN)
GPIO17	CF電源	GPIO25	PHY(N0TXD0)
GPIO18	CF電源	GPIO26	PHY(N0TXD1)
GPIO19	CF電源	GPIO27	PHY(N0TXD2)
GPIO20	拡張コネクタ	GPIO28	PHY(N0TXD3)
GPIO21	拡張コネクタ	GPIO29	拡張コネクタ(I2SDIO)
GPIO22	拡張コネクタ	GPIO30	拡張コネクタ(I2SCLK)
GPIO23	拡張コネクタ	GPIO31	拡張コネクタ(I2SWORD)

補足 :GPIOアサイン(続き)

GPIO200	拡張コネクタ(LRD0#)	GPIO208	拡張コネクタ(S0DOUT)
GPIO201	拡張コネクタ(LRD1#)	GPIO209	拡張コネクタ(S0CLK)
GPIO202	拡張コネクタ(LWR0#)	GPIO210	拡張コネクタ(S0DEN)
GPIO203	拡張コネクタ(LWR1#)	GPIO211	拡張コネクタ(IRDATX)
GPIO204	CF(PREG#)	GPIO212	拡張コネクタ(U0TXD)
GPIO205	CF(PCE1#)	GPIO213	拡張コネクタ(U1TXD)
GPIO206	CF(PCE2#)	GPIO214	UART3(U3TXD)
GPIO207	CF(PWE#)	GPIO215	PHY(N0MDC)

補足 :デバッグ事例

E!Kit-1100のFlashメモリ回路

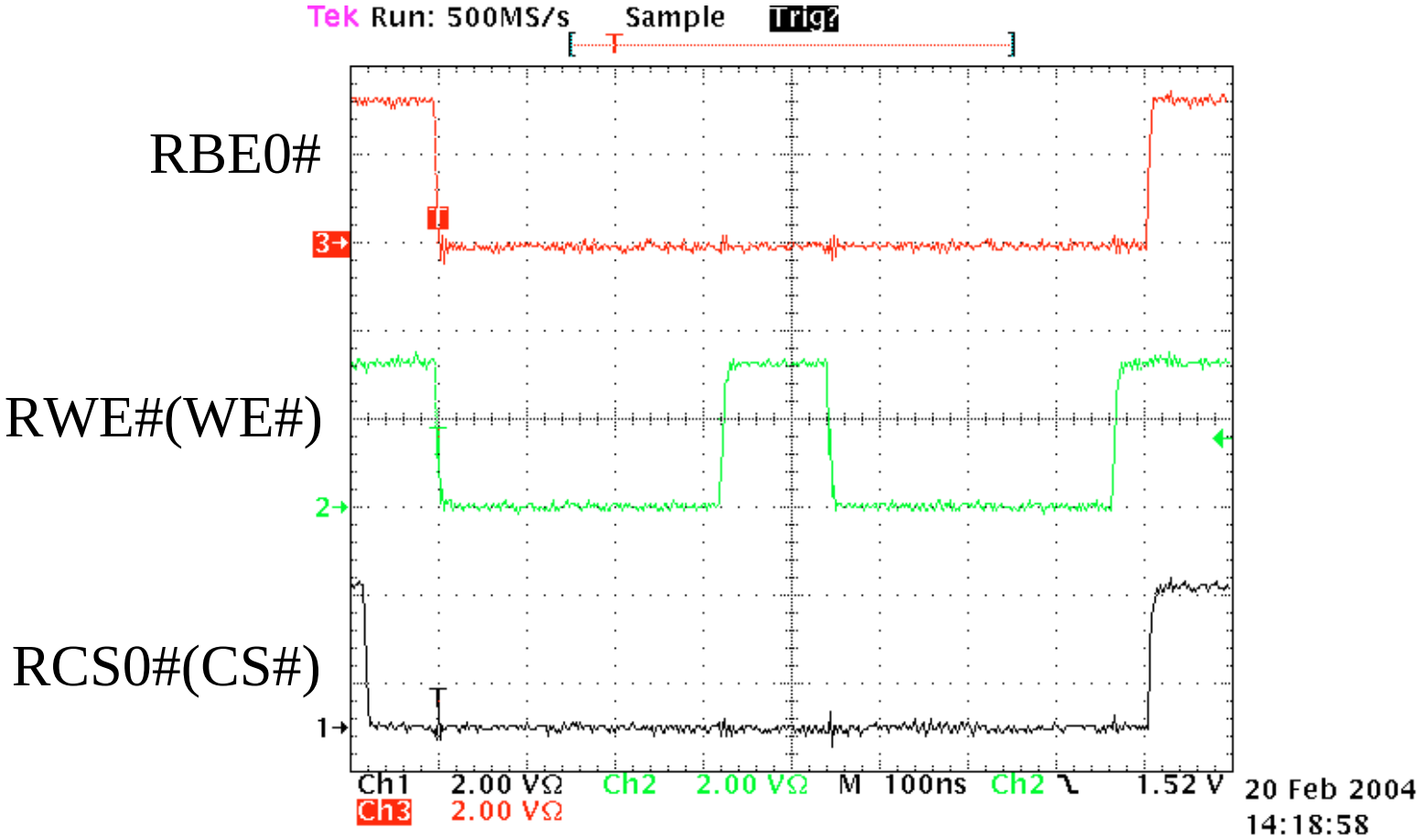


しかしAu1100の設定の不具合だった

対策回路

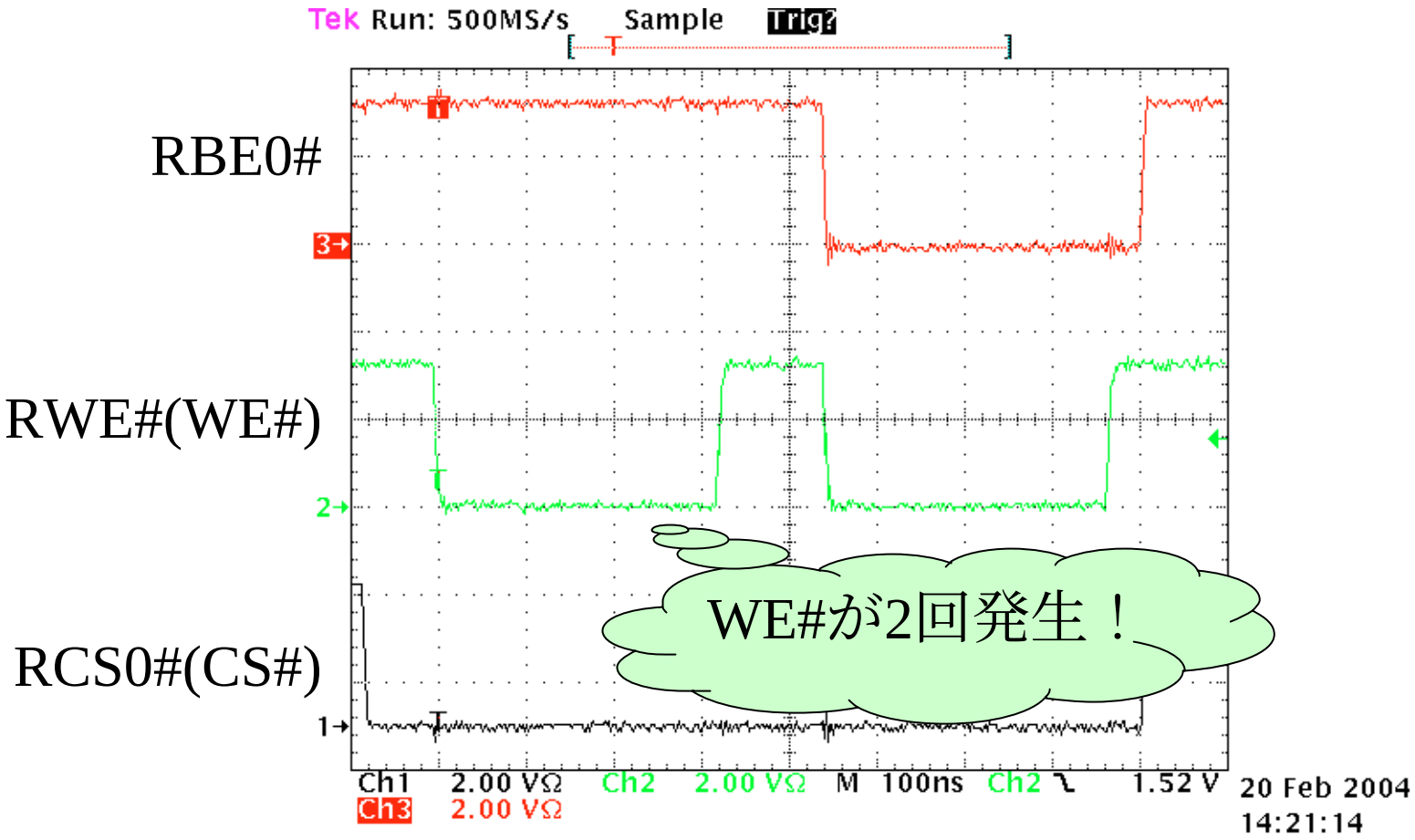
補足 :デバッグ事例(続き)

32bitアクセス



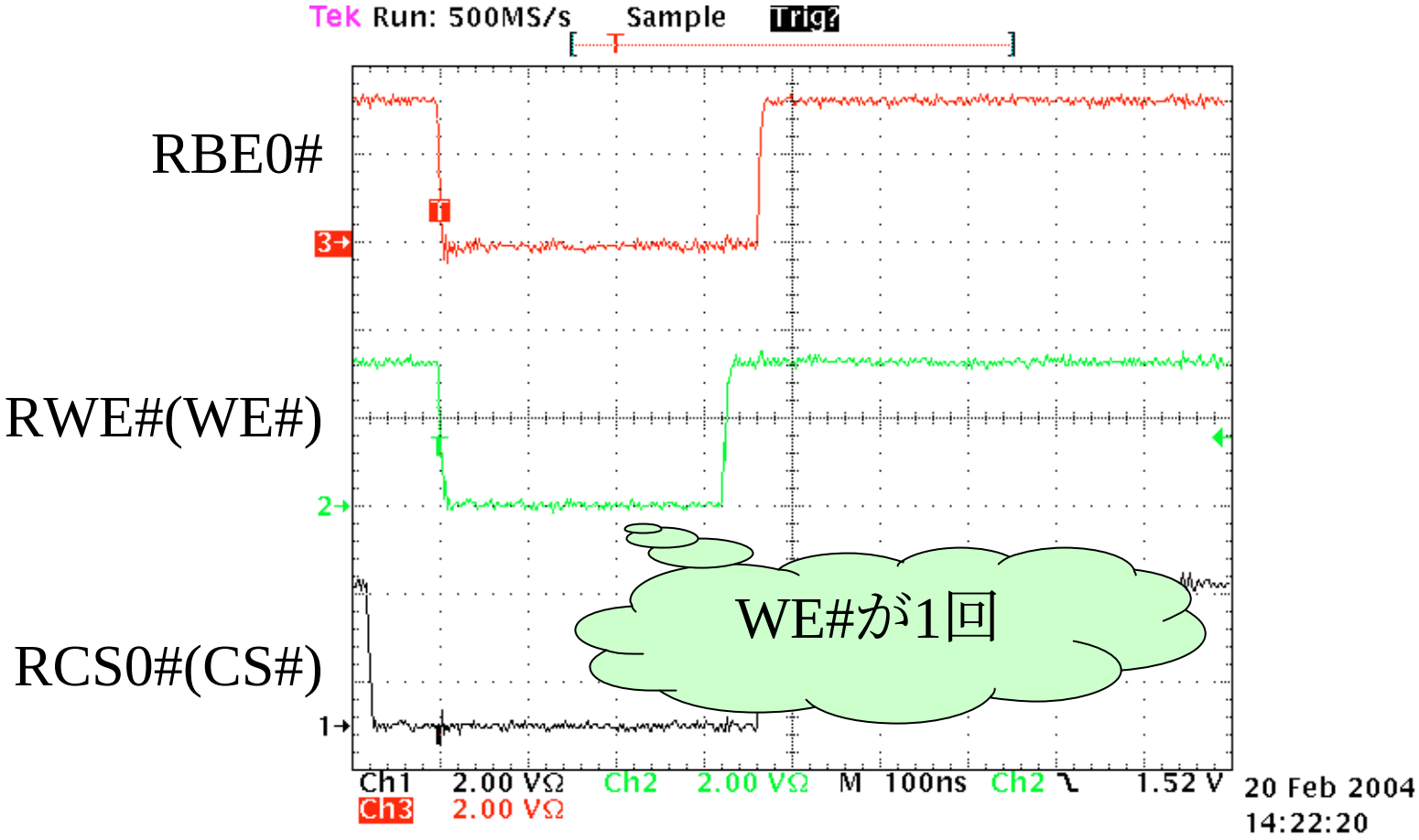
補足 :デバッグ事例(続き)

16bitアクセス



補足 :デバッグ事例(続き)

16bitアクセス（Au1100設定対策後）



連絡先

株式会社デバイスドライバーズ
川本 泰久
info@devdrv.co.jp