Linux 上 Mini210S 開発キット OS なしのプログラム

開発マニュアル

株式会社日昇テクノロジー

http://www.csun.co.jp

info@csun.co.jp

作成日：2013/01/04

修正履歴

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2013/01/25

目次

第 I 章　紹介

一、 紹介

ネット技術の高速発展と伴に、性能が優れた Cortex-A8 CPU プロセッサはインテリジェント端末の最優先選択であります。携帯或いはタップレットは Cortex A8 アーキテクチャを多く使用しております。組み込み開発のビジョンから見て、これらの CPU は全て ARMV7 命令セットを使用します；コスト比べて、ARM9、ARM11 アーキテクチャ CPU の製品は段々市場からなくなりました。組み込み開発者とファン達にとって、特に初心者には基層から Cortex A8 を習得、把握するのは、工夫を掛かると思います。

そのため、弊社のエンジニア達は大量の時間と精力を掛けって、Mini210S 開発ボードを基づき、本マニュアルを編成し、組み込み開発に興味がある方に短時間で S5PV210 の操作、使用でき、そして OS なしのプログラムから Cortex A8 を学べるようにと努力している方々の学習能力が違いので、本マニュアルについての技術指導や質問解決なとのサービスは提供いたしませんので予めご了承お願いします。本マニュアルも S5PV210 を基づきの Tiny210/Tiny210V2 などの開発ボードプラットフォームに適用します。

二、開発環境

1) 前提条件：C 言語や ARM アセンブリ言語の基礎があります。
2) 開発：windows xp + 仮想マシン fedora15、source insight でソースコードを書く
3) クロスコンパイラ　：arm-linux-gcc-4.5.1

注：クロスコンパイラのインストール方法では、開発ボードの Linux マニュアルをご参照ください。

4）開発ボードサポート：すべてのプログラムは Mini210S で成功実行しましたので、そして他の 210 シリーズの開発ボードにも大部分のコードは依然として適用します。

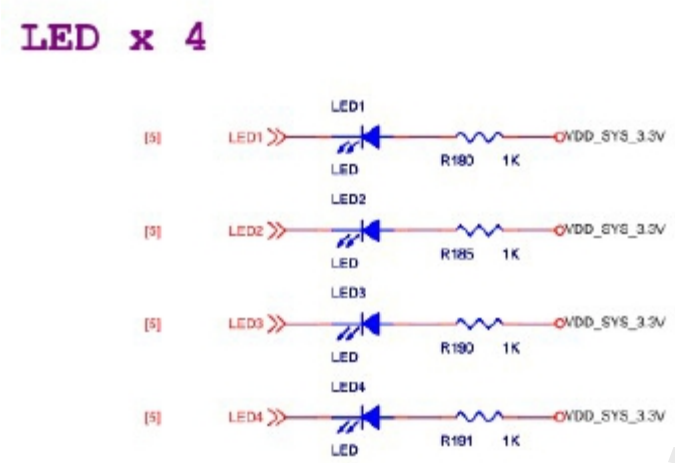
三、ファイルに関わった OS なしのプログラムについて

まず、2 から 8 章では最初基本なものから始めます：LED 点灯、sd カードでロードの仕方、S5PV210 の起動・ブートプロセスとコードのロケーションなど、ステップバイステップで S5PV210 作動、操作プロセスをユーザーに示します。これらの必要な知識があれば、以降の章では、我々は同じ方法でより複雑な OS なしのプログラムを編集することができます。ユーザーがブックマーク機能でドキュメント上ハードウェアモジュールの相関情報を閲覧、理解することができます、本書には合計 23 件の OS なしプログラムがあります。そしてバージョンは更新し続きますので、ユーザーに良い経験・優れた OS なしプログラムを作成できましたら、私たちにもフィードバックしてください。

第 II 章　ASM 言語で LED を点灯

第一節　回路図確認

Mini210S ボードには 4 つのユーザプログラマブル LED を提供しています。下記の回路図をご覧下さい。



LEDの回路図

回路図ピン〝LED1〟を検索して、結果は：

LEDピンダイアグラム

LED1、2、3、4はCPUポートリソース　GPJ2_0、1、2、3を使用します。

第二節　プログラム説明

完全なコードは、ディレクトリ 1.led_s を参照してください。　（注：このチュートリアルでは、ユーザーの分かり易い習得するために、すべてのコードが詳細なコメントを含みます。）

1. start.S

概略から見て、Mini210S の 4 つの LED を点灯するには次の 2 のステップがあります。:

ステップ 1：レジスタ GPJ2CON を設定して、GPJ2_0/1/2/3 4 ピンを出力機能に調整します。

ステップ 2：レジスタ GPJ2DAT に 0 を書き込み、GPJ2_0/1/2/3 4 ピンはロー出力にしで、4 つの LED ランプが点灯します ; 逆に、登録する GPJ2DAT に 0 を書き込むと、GPJ2_0/1/2/3 4 はハイ出力にしで、4 つの LED ランプが消します。

上記の 2 つの手順は、start.S のメイン内容であります、もっと詳しいコアアセンブリ命令は GNU アセンブラ命令セットをご参照ください。

2. Makefile

コード：

```
led.bin: start.o
arm-linux-ld -Ttext 0x0 -o led.elf $^
arm-linux-objcopy -O binary led.elf led.bin
```

```
arm-linux-objdump -D led.elf > led_elf.dis
gcc mkv210_image.c -o mkmini210
./mkmini210 led.bin 210.bin
%.o : %.S
arm-linux-gcc -o $@ $< -c
%.o : %.c
arm-linux-gcc -o $@ $< -c
clean:
rm *.o *.elf *.bin *.dis  -f
```

次は Makefile について説明します。ユーザーは Makefile のディレクトリ下で make コマンドを実行すると、システムは次のように操作を実行します：

ステップ 1　arm-linux-gcc -o $@ $< -c を実行し、当ディレクトリ内のアセンブリファイルと C ファイルを.o ファイルにコンパイルします。

ステップ 2 arm-linux-ld -Ttext 0x0 -o led.elf $^を実行し、.o ファイルを elf ファイルとコンパイルします、-Ttext 0x0 はプログラムが 0x0 で実行するため、現在編集するコードがアドレスと関係がないため、任意的なアドレスでも実行できます；

ステップ 3　arm-linux-objcopy -O binary led.elf led.bin を実行し、elf ファイルを抽出し、開発ボードで実行できる bin ファイルに変更します；

ステップ 4　arm-linux-objdump -D led.elf > led_elf.dis を実行し、elf ファイルを逆アセンブルして DIS ファイルに保存し、デバッガの時は使用できます；

ステップ 5　mkmini210 で led.bin ファイルを処理して、mkmini210 は mkv210_image.c で編成します。詳しい情報は mkv210_image.c 関連説明をご参照下さい；

3. mkv210_image.c

サムスン提供する S5PV210 文書《S5PV210_iROM_ApplicationNote_Preliminary_20091126.pdf》とチップのマニュアル 《S5PV210_UM_REV1.1.pdf》を参照し、S5PV210 起動後は先に内部 IROM 中のコードを実行して必要な初期化を行います；完了後自動的ハードウェアの NAND Flash または SD カードのブートデバイスの最初の 16K のデータを IRAM 中に読み取ります、この 16K のデータの初位 16 バイトで一つのチェックサム値があります、データをコピーする時 S5PV210 は bin ファイル中の'1 'の数を統計し、次にチェックと比較します。等しい場合はプログラムを実行し続け、それ以外の場合は停止します。そのために、S5PV210 で実行する bin ファイルは全部 16 バイトの頭を持ちます、初位 16 バイトははチェックサム情報が含まれます。mkv210_image.c は ARM9 フォーラムのファンが提供する工具で、bin ファイルに 16 バイトの頭を添付すると使用します。 mkv210_image.c コア機能は次のようになります：

ステップ 1　　16K バッファの配置；

ステップ 2　led.bin を buffer の始まりの 16 バイトまで読み込みます；

ステップ 3　チェックサムを計算し、チェックサムを buffer の 8～11 に保存します；

ステップ 4　16k の buffer を 210.bin にコピーします；

16K バッファー(buffer)（210.bin）

16byte ヘッド	led.bin

チェックサムと統計キーコードは下記の通りです：

```
a = Buf + SPL_HEADER_SIZE;
for(i = 0、  checksum = 0; i < IMG_SIZE - SPL_HEADER_SIZE; i++)
checksum += (0x000000FF) & *a++;
```

コード mkv210_image.c には既に非常に豊富な解釈が含めます、ユーザー自習で簡単に読み取ることができます。

第三節　コードコンパイルとプログラミング・実行

SD カードを PC にアクセスして、Fedora 端末で下記のコマンドを実行します：

```
# cd 1.led_s
# make
# chmod 777 write2sd
# ./write2sd
```

make を実行後、210.bin ファイルは生成します、./write2sd 実行し、.bin は sd カードセクタ 1 にプログラミングします、sd カードの初起セクタは 0 で、セクタサイズは 512 バイト、SD 起動時、IROM の硬化コードはセクタ 1 からコードをコピーします。 write2sd はスクリプトファイルで、内容は次のようになります：

```
#!/bin/sh
sudo dd iflag=dsync oflag=dsync if=210.bin of=/dev/sdb seek=1
```

dd は書き込みと読み取りコマンドで、if は輸入、ｏｆは出力、ｓｅｅｋはセクタ 1 から書き込みと意味します。

注意：SDB 文書 SD カードのデバイスノードとします、ユーザーのニーズに応じてノートを変更することができます。後述はしません。

第四節　実験現象

SD カードを Mini210S に挿入して、SD カードを起動・通電します。次の現象を確認できます：LED が通常点滅します、これは最初のコンパイルプログラムですべての LED が点灯することが正常に作動しました。ここでの要点は、CPU が始まったばかりの時、ウォッチドッグ· リセット機能により、CPU がリセットされるはずですが、コード 1.led_s 中手動でウォッチドッグをオフにすること

もしないのに、、なぜプログラムが正常に作動しましたかのことです。この問題について、次の章でご説明致します。

第 III 章　S5PV210 の起動プロセス

第一節　IROM と IRAM の初期の認識

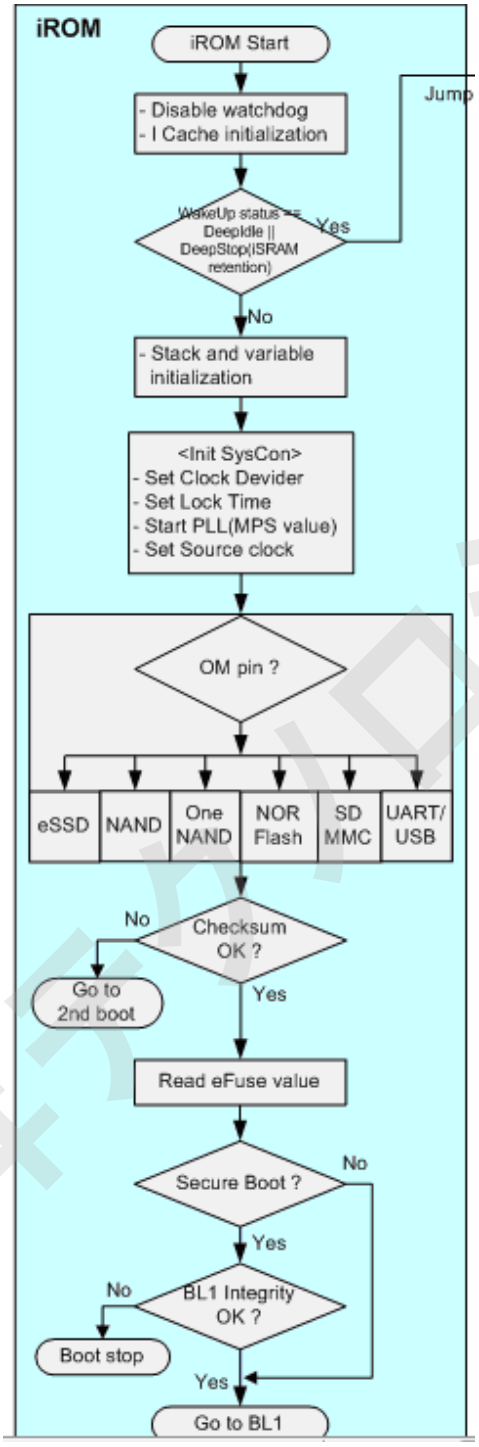
S5PV210 には 64KIROM と 96K IRAM を含みます、システム起動時は IROM と IRAM に依存しています。IROM と IRAM のストレージスペースは以下を参照してください。

Address		Size	Description	Note
0x0000_0000	0x1FFF_FFFF	512MB	Boot area	Mirrored region depending on the boot mode.
0x2000_0000	0x3FFF_FFFF	512MB	DRAM 0	
0x4000_0000	0x7FFF_FFFF	1024MB	DRAM 1	
0x8000_0000	0x87FF_FFFF	128MB	SROM Bank 0	
0x8800_0000	0x8FFF_FFFF	128MB	SROM Bank 1	
0x9000_0000	0x97FF_FFFF	128MB	SROM Bank 2	
0x9800_0000	0x9FFF_FFFF	128MB	SROM Bank 3	
0xA000_0000	0xA7FF_FFFF	128MB	SROM Bank 4	
0xA800_0000	0xAFFF_FFFF	128MB	SROM Bank 5	
0xB000_0000	0xBFFF_FFFF	256MB	OneNAND/NAND Controller and SFR	
0xC000_0000	0xCFFF_FFFF	256MB	MP3_SRAM output buffer	
0xD000_0000	0xD000_FFFF	64KB	IROM	
0xD001_0000	0xD001_FFFF	64KB	Reserved	
0xD002_0000	0xD003_7FFF	96KB	IRAM	
0xD800_0000	0xDFFF_FFFF	128MB	DMZ ROM	
0xE000_0000	0xFFFF_FFFF	512MB	SFR region	

第二節　完全スタートアップシーケンス

システムの起動時には、IROM 中のコードを実行し、通常初期化します。具体的な手順下記の通りです：

ステップ 1 ウォッチドッグを閉じる
ステップ 2 icache を初期化；
ステップ 3 スタックを初期化；
ステップ 4 クロックを設定；
ステップ 5 （NAND / SD /ワンナンド）機器の起動設備を判断し、チェックサムをチェックします。その後ブートデバイスからの初起 16K までのコードを IRAM 0xD0020000 にコピーします；
ステップ 6 セキュリティモードがオンになっている場合、整合性チェックを実行します；
ステップ 7 IRAM 0xD0020010 アドレスにジャンプして、実行し続けます；



IROM 起動シーケンス

第 I 章 LED を点灯するの解説中で、プログラム 210.bin は最終 IRAM 0xD0020000 の開始アドレスにコピーされます。ビンには 16 バイトのヘッドが含めていますので、コードの真の開始アドレスが 0xD0020010 です、そのため上記のステップ 7 は IRAM 0xD0020010 アドレスにジャンプしてシーケンスを起動します。一方、IROM コードは、既にウォッチドッグ閉じましたので、これは最初の章で手動でウォッチドッグ·プログラムを終了しなくても始動シーケンスが正しく実行できるの理由です。次の章では、我々は手動でウォッチドッ

グを停止し、C の関数を呼び出すことにより IROM のコードをテストし、スタックを設定し状況を確認します。

第 IV 章　ウォッチドッグを閉じると C 関数の呼び出し

第一節　ウォッチドッグ· 背景知識

ウォッチドッグの役割は、CPU の動作を監視することで、ノイズやシステムエラーなどの障害が干渉した場合にできるだけ早く通常の仕事を再開することにサポートすることです。PWM、ウォッチドッグが共にタイマー機能があります、両者の違いは、ウォッチドッグのタイマタイミングをリセットシグナルが発送できます、PWM タイマがその機能がありません。後の章では詳しい解説があります、ここでは閉じるのみです。

第二節　プログラム説明

完全なコードは、ディレクトリ 2.led_s_wtd ご参照ください。

1. start.S

1.led_s_wtd コードと比較して、コード 2.led_s_wtd では次の二点が違います：

1) 手動でシャットダウンウォッチを閉じて、レジスタ WTCON に 0 を書き込みのみ；

2) C 関数を使用して遅延機能を実現し、そして、IROM 硬化コードにスタックはに設定するかどうかをテストします；

普通の C 言語遅延関数が一つ含みます。コードは次のとおりです：

```
void delay(int r0)
{
    volatile int count = r0;
    while (count--);
}
```

コンパイルが C 関数を呼び出す時、引数の数が 4 つを超えない場合、r0～r3 の 4 つのレジスタを使用して引数を発送します；4 つ以上な場合ではの残りの他の引数はスタックを介して発送します、delay()は一つの引数だけで、r0d で発送します。また、volatile はコンパイラが自動最適化でこのコードを消去し遅延機能を実現で着ないを防ぐために、追加するものです。 Makefile と write2sd は前のプログラムに差別がないので、以後、ディレクトリとコードは変更がない場合、特に説明は致しません。

第三節　コードコンパイルとプログラミングの実行

SD カードを PC にアクセスして、Fedora 端末で下記のコマンドを実行します：

```
# cd 2.led_s_wtd
# make
```

```
# chmod 777 write2sd
# ./write2sd
```

makeを実行後、210.binファイルは生成します、./write2sd実行し、.binはsdカードセクタ1にプログラミングします。

第四節　実験現象

SD カードを Mini210S に挿入して､SD カードを起動･通電します｡次の現象を確認できます: LED が通常点滅します、、手動でウォッチドッグが成功シャットダウンした、同時に IROM のコードがスタックを設定していることを証明します。ここでの要点は : なぜプログラムが C 言語の関数が正常に呼び出すことから、IROM コードにスタックを設定したと証明できますか？それはコンパイルが C 関数を呼び出す時、引数の受け渡し、現在状態保存と正常戻り、一時的な変数の保存などにスタックが必要で、delay()関数で count 一時的な変数で、プログラムが正常に実行するとスタックの設定を証明できます。

第Ⅴ章　スタック設定と C 言語で LED を点灯

第一節　C 関数を呼び出す前、先にスタックを設定する理由：

1. スタックの整体的役割

1) 保存；

2) 引数受け渡し：アセンブリコードからC関数を呼び出すには、引数必要がある；

3) 一時変数を保存 : 非静的ローカル変数やコンパイラが自動的生成した一時変数の保存機能を含みます；

2. 具体的に説明

1) シーン保存

現場は、事件が発生する現場で、痕跡を記録する必要があります。でないと、一旦破壊されたら、現場を回復することはできません｡ここの現場では､ＣＰＵが作動時にレジスタ使っいました､例えばレジスタR0、R1など、そしてこれらのレジスタの値を保存せず、直接サブルーチンにジャンプし実行するい場合は、関数の実行もレジスタが使用するためで、前のデータが破壊されます。したがって、関数呼び出しの前に、これらのレジスタなどのシーン)等を一時保存して（スタックpush)、関数実行した戻り（スタックpop)、シーンを復元をします。これでCPUが正常に次の指令が実行できます。

レジスタの値を保存するには、通常は push指令を使用します､対応するレジスタの値を一つずつでスタック上保存します、いわゆるスタックpushです。そして呼び出し子関数が実行終了したら、ＰＯＰ用を呼び出し、スタックから対応する値を取り出し、対応するレジスタに与えます、いわゆるスタックpopです。

保存したレジスタの中に、ＬＲの値も含まれでます（ＢＬコマンドでジャンプすると、ＰＣの値はＬＲに保存される為です)、そしてサブルーチンが終了すると、スタックからＬＲの値を取り出して、ＰＣに与え

ます。これでサブ関数の正常戻りが実現できます。

2）引数受け渡し

多くの場合、C言語が関数を呼び出す時に、関数に引数を与えます、そしてコンパイラでアセンブリ言語に変換する時には、引数一時保存する必要があります、一方アクセス権も与え、引数伝送機能を実現します。保存については、二つの場合があります。一つ、引数自体の数が4つを超えない場合で、レジスタR0～R3で伝送します。前のシーン保存で既にレジスタの値を保存しましたので、現在はアイドル· ステートで、引数を置くことができます。

もう一つ、引数の数が4を超える場合、レジスタが足りませんので、スタックを使用する必要があります。

3）一時的な変数はスタックに保存

関数の非静的ローカル変数やコンパイラが自動的生成の一時変数を含めます。

第二節　説明する手順

完全なコードは、ディレクトリ 3.led_c_spを参照してください。

1. start.S

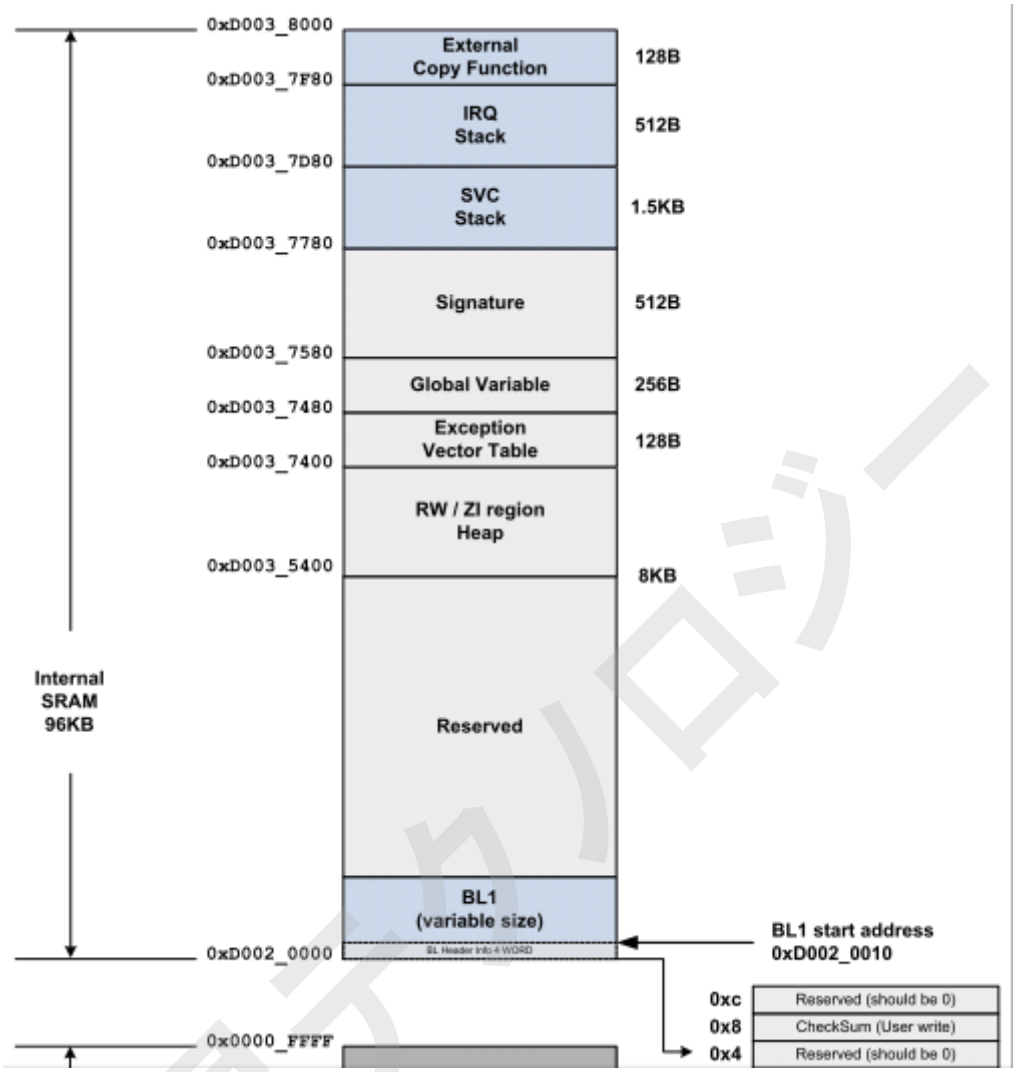
本章では、C関数で照明と遅延機能を実現します。コード3.led_c_spにあります。、start.Sの役割は次のとおりです。

ステップ　1 ウォッチドッグを閉じる；

ステップ　2　スタックを設定する；

ステップ　3 C関数led_blink（）を呼び出し、LEDが点滅を実現します。；

スタックを設定し、すなわち、spレジスタを設定することである、可用なメモリを指向するで、ここで0xD003_7D80に指向しています、理由は下記の通りです：



IROM内部空間の割り当て図

IROM にコード設定のSPは0xD003_7D80で、サムスンのを従ってまたは自己設定し、コードを上書きしないように設定すればOKです。

2. led.c

led.cは、2つのC関数があります、led_blink（）はLEDの点滅、delay（）は遅延機能を実現します、コードは次の通りです：：

```
void delay(int r0)              // 遅延
{
    volatile int count = r0;
    while (count--)
    ;
}
```

```
void led_blink()              // LED  点滅
{
    GPJ2CON = 0x00001111;                  //  ピン・コンフィギュレーション
    while(1)
    {
        GPJ2DAT = 0;              // LED on
    delay(0x100000);
        GPJ2DAT = 0xf;            // LED off
delay(0x100000);
     }
   }
```

コード内のコメントは明白です。アセンブリと比較して、C言語でLEDを動作させるは効率的です、以後のコードは出来るだけ低C言語でプログラミングします。

第三節　コードコンパイルとプログラミングの実行

SDカードをPCにアクセスして、Fedora端末で下記のコマンドを実行します：

```
# cd 3.led_c_sp
# make
# chmod 777 write2sd
# ./write2sd
```

makeを実行後、210.binファイルは生成します、./write2sd実行し、.binはsdカードセクタ1にプログラミングします。

第四節　実験現象

SDカードをMini210Sに挿入して、SDカードを起動・通電します。次の現象を確認できます：

LEDは正常に点滅し、スタックを設定した後、C関数は呼び出せますので、プログラミングの速度は大幅にあがります。

次の章では、icacheの起動、終了をご紹介します。

第 VI 章　icache 制御

第一節　cache は何

プログラムのアクセス権限により、メインメモリとCPU汎用レジスタに高速のクラス、小さいメモリ容量が設定され、実行中の一部のコマンドやデータをメインメモリからレジスタに保存して、一時間内でCPUを直接使用でき、プログラムの計算速度が大幅に上がります。このメインメモリとCPUの間に高速小容量メモ

リは高速 cacheと呼ばれる。

よく使うcache (キャッシュ)は icacheとdcache二種類があります。ICacheの使用はより簡単です、システム通電後、ICacheの内容は無効で、機能はオフになって、｀CP15コプロセッサ｀の｀レジスタ1｀の｀ビット[1]｀に１を書き込むとICacheを起動できます、0を書き込むと終了できます。icache がオフ状態では、CPUは毎回メインメモリをフェッチします、性能は非常に低いです。Icacheがいつでも起動できるため、早いほどICacheを起動する方は効率的です。

ICacheに類似したで、通電時ではdcacheの内容は無効でdcache機能がオフになって、｀CP15コプロセッサ｀の｀レジスタ1｀の｀ビット[2]｀に１を書き込むとdcacheを起動できます、0を書き込むと終了できます。dcacheは、MMUが起動後実行できるしかないので、初期化設備にとっては実行する必要がありませんので、ここではMMUとdcacheは実行しません。

第二節 プログラム説明

完全なコードは、ディレクトリ4.led_c_icacheご参照ください。3.led_c_spと比べったら、start.S 中にicacheの制御コードを追加しただけです。

コードは次の通りです：

```
#ifdef CONFIG_SYS_ICACHE_OFF
bic r0、 r0、 #0x00001000                    @ clear bit 12 (I) I-cache
#else
orr r0、 r0、 #0x00001000                    @ set bit 12 (I) I-cache
#endif
mcr p15、 0、 r0、 c1、 c0、 0
```

ICache CONFIG_SYS_ICACHE_OFFは定義されていない場合は、 icacheはONにします、でないとOFFにします。。コプロセッサのコマンドについてはs3c2410チープマニュアルをご参照ください。

第三節 コードコンパイルとプログラミングの実行

SDカードをPCにアクセスして、Fedora端末で下記のコマンドを実行します：

```
# cd 4.led_c_icache
# make
# chmod 777 write2sd
# ./write2sd
```

makeを実行後、210.binファイルは生成します、./write2sd実行し、.binはsdカードセクタ1にプログラミングします 。

第四節 実験現象

SDカードをMini210Sに挿入して、SDカードを起動・通電します。次の現象を確認できます：

LEDが正常に点滅するが、点滅周期は非常に長い、それはコード中の遅延時間は10倍に設定されるためです。IROM硬化コードは既に icacheを起動しましたので、 icache機能をテストしたい場合は、マクロCONFIG_SYS_ICACHE_OFFを定義することによってicacheをシャットダウンする。試験によって、 icache OFF状態ではLEDが一回点滅するには20秒が必要で、 icache ON状態ではLEDが一回点滅するには10秒です。

第 VII 章　IRAM+0x4000 にコードをリロケーション

第一節　リロケーション

プログラムについて、まず二つの概念を理解しておく必要があります、一つプログラムの現在のアドレス、すなわち実行時にそのプログラムのアドレス；二是程序的链接地址、即程序运行时应该位于的运行地址。プログラムのコンパイル時に、プログラムのリンクアドレスは指定することができます。

S5PV210にとっては、起動時はNAND Flash/sdブートデバイスから最初の16Kコードを IRAMへコピーするだけで、もしプログラムが16Kを超えた場合はどうなりましょう？プログラム全体のコードを DRAM などの広いスペースへ全てコピーして、そして DRAM にジャンプし、コードを実行します、この転移の過程はリロケーションと呼びます。この章では、主にリロケーション方法を説明します。DRAM単にIRAMの 0xD0020010 から0xD0024000までにコードをコピーして、0xD0024000でコードを実行するプロセスです。

第二節　プログラム説明

完全なコードは、ディレクトリ5.link_0x4000、ご参照ください。前章と比べったら、start.S 中にリンカスクリプトlink.ldsを追加致します。

1. link.lds

リンカスクリプトとはリンクスクリプトは、その主な機能は、どのように入力ファイル内のセグメント（SECTION）を出力ファイルへのマッピング、および出力ファイル·のストレージ·レイアウト・ドキュメントを制御することです。リンカスクリプトの基本コマンドはSECTIONSで、一つのSECTIONSには一つや多数のSECTIONを含み、セクション（SECTION）はリンカスクリプトの基本単位で、これは入力ファイルのセグメントの配置方を示します。

リンクスクリプトの標準形式は次のとおりです：

```
SECTIONS
{
    sections-command
    sections-command
}
```

次は link.lds と合わせって詳しい説明致します：

```
SECTIONS
{
```

```
. = 0xD0024000;
.text : {
start.o
* (.text)
}
.data : {
* (.data)
}
bss_start = .;
.bss : {
* (.bss)
}
bss_end = .;
}
```

1) リンカスクリプト中、独自の(.)は現在アドレスを示して、. = 0xD0024000はプログラムのリンクアドレスを示します；

2) link.lds中、.text 、 .data 、 .bss は各自textセクション、dataセクション、bssセクションのSECTION名を示します（セクションの名は固定するではありません、状況によって任意名を与えます。）。.textセクションの内容は start.o と残りのコードのtextセクションを含めます；.dataセクションはコード中すべてのdataセクション；.bss 段はコード中すべてのbssセクションを含めます；

3) bss_start と bss_end はbssクションの開始アドレスと終了アドレス、start.Sでは使用されます。

次は data、text、bssセクションを説明します：

1) dataセクション：データセグメント（datasegment）通常、プログラム中初期化されたグローバル変数を保存するメモリー領域です。データセグメントは、静的メモリ割り当てであります。

2) textセクション：コードセグメント通常のプログラムの実行コードを保存するメモリー領域です。。プログラムが実行される前に、スペースの容量は決めされます、読み取り専用メモリ領域に属しており、部分アーキテクチャはコード· セグメントを書き込みは可能、すなわちプログラムを変更できます。コード· セグメントは、読み取り専用の定数変数を含める場合もあります、例えば文字列定数など：

3) bssセクション：プログラム内で初期化されていないグローバル変数を保存するメモリ領域のことBSS（BlockStarted by Symbol）です。プログラムにグローバル変数がある場合、それはBSSセグメントに保存されます、そして、グローバル変数のデフォルトの初期値は0であるため、手動でbssセグメントをクリアする必要があります。

2. start.S

start.S 中、タイマーを初期化後、また三つの手順を追加します：

ステップ 1 リロケーション、コードは次のとおりです：

```
// _start 現在のアドレス
adr r0、 _start
// _start のリンクアドレス
ldr r1、 =_start
ldr r2、 =bss_start
cmp r0、 r1
beq clean_bss
copy_loop:
ldr r3、 [r0]、 #4
str r3、 [r1]、 #4
cmp r1、 r2
bne copy_loop
```

ADR命令はコードの現在アドレス値を取得するに、ldr命令はのコードのリンクアドレス値のがをフェッチするとことです；コードでは、まず_startラベルは現在のアドレス（つまり0xD0020010）、そして続きリンクアドレス（0xD0024000）があります、binファイルにbssセグメントを保存する必要がないため、符号長はbss_startまたは-_startの実行アドレスをcopy_loopでコピーします。

ステップ 2 bss をクリア、コードは次のとおりです：

```
ldr r0、 =bss_start
ldr r1、 =bss_end
cmp r0、 r1
beq run_on_dram
mov r2、 #0
clear_loop:
str r2、 [r0]、 #4
cmp r0、 r1
bne clear_loop
```

まずbssセグメントの開始アドレス（bss_start）、bssセグメント終了アドレス（bss_end）を取得します；最後にclear_loopコマンドで bssセグメントのメモリをクリアし、bss_startとbss_end位置をlink.ldsにロケーションします。

ステップ 3 ジャンプ、コードは次のとおりです：

```
run_on_dram:
ldr pc、 =main
```

LDR命令はリンクmain関数のアドレスを取得する機能で、ldr pc、 =main を実行すると、プログラムは

0xD002200 +main関数のoffsetアドレスにジャンプします。

第三節　コードコンパイルとプログラミングの実行

SDカードをPCにアクセスして、Fedora端末で下記のコマンドを実行します：

```
# cd 5.link_0x4000
# make
# chmod 777 write2sd
# ./write2sd
```

makeを実行後、210.binファイルは生成します、./write2sd実行し、.binはsdカードセクタ1にプログラミングします。

第四節　実験現象

SDカードをMini210Sに挿入して、SDカードを起動・通電します。次の現象を確認できます：

LEDが正常に点滅します、実験現象は前の章と同じですが、実行中のプログラムには大きな違いがあります。この章ではリロケーションを習得した、次の章でコードをDRAMにリロケーションを説明します。

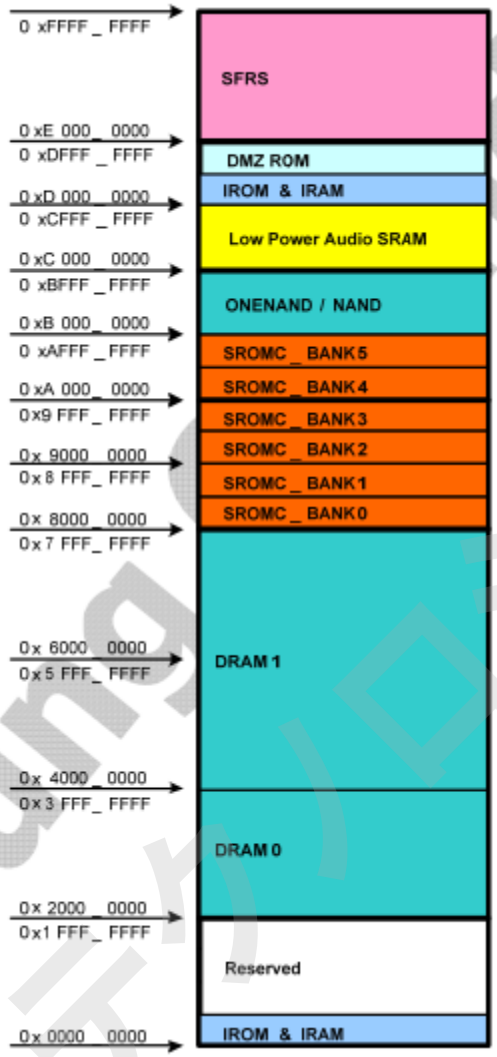
第 VIII 章　DRAM にコードをリロケーション

第一節　DRAM について

前の章では、コードのリロケーションを学びましたが、 スペースは96KのみのIRAMに移転するのは効果が良くないです。正しいアプローチは、DRAM、大容量のメインメモリへリロケーションすることです。

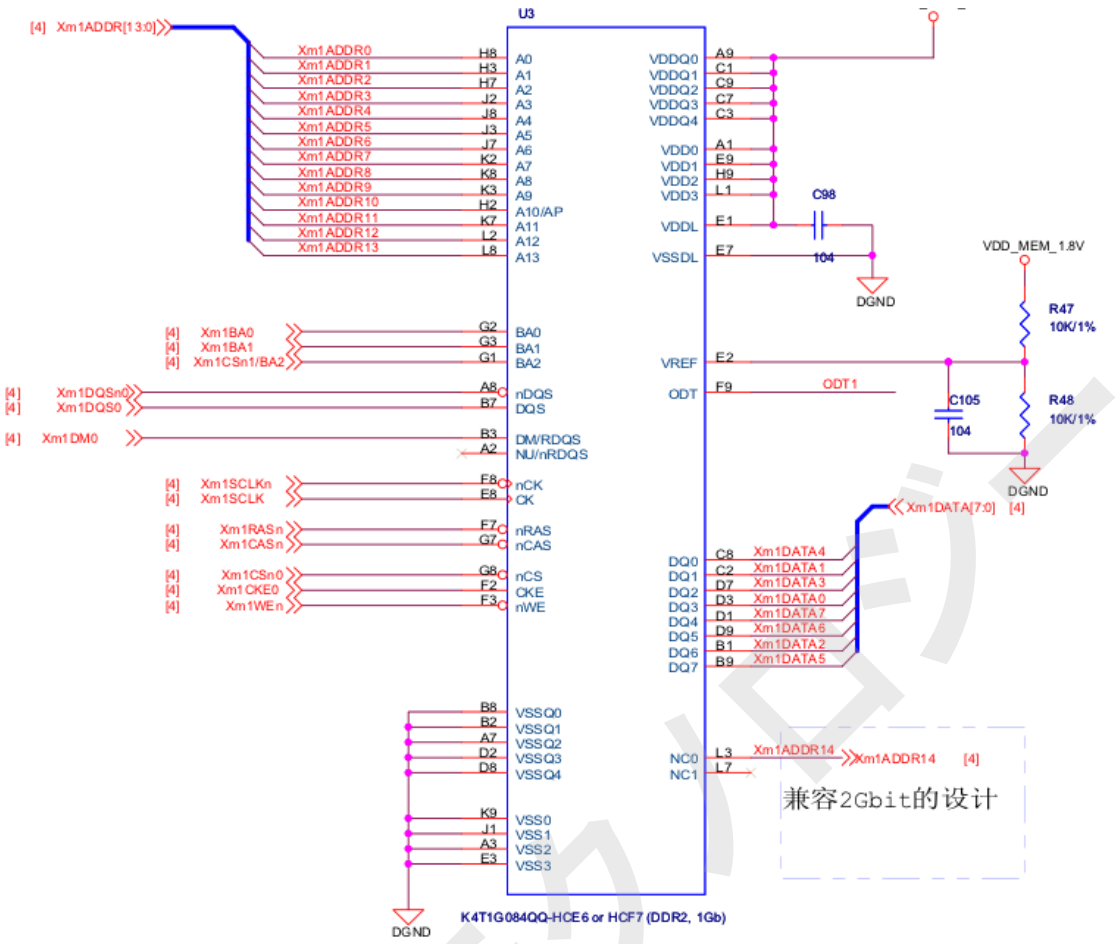
S5PV210には2つの独自作動のDRAMコントローラがあります、DMC0とDMC1です。DMC0は最大512M DRAM、DMC1サポート最大1GのDRAMをサポートしています。両方もDDR/DDR2の128MB、256MB、512MB、1GB、2GB、4GBメモリデバイスのをサポートし、16/32bitビット幅をサポートします。

次の図をご参照ください：



アドレスマップ

DRAM0の対応アドレスは0x2000_0000~0x3FFF_FFF 合計512M、DRAM1対応するアドレスが0x4000_000～0x7FFF_FFFF合計1Gです。Mini210S回路図をご参照ください：



Mini210S DRAMの回路図

DRAM Mini210Sの512Mは四つの128M DRAMチップで組み合わせます、（上記図は1つの場合の回路図）、チップセレクト端子を見れば、4つのDRAMチップは全部DMC0に接続されると確認できます。

DRAMのピン接続図

DRAMを使用するには？Mini210Sにとっては、現在はDMC0を使用するだけで、DMC0とDDR2 DRAMチップをを初期化すれば満足できます。ここで注意して欲しいのは、ここのDRAMの初期化は実験的なもので、SuperbootでDRAMの初期化とは違います。

第二節　プログラム説明

完全なコードは、ディレクトリ6.sdramをご参照ください。本章ではコードが大幅に変更があります。まず、プロジェクト全体はBL1とBL2 、２つのディレクトリに分割されます、ディレクトリBL1下のコードはBL1.binという名前のファイルにコンパイル・リンクされ、ディレクトリBL2下のコードはBL2.binという名前のファイルにコンパイル・リンクされます。BL1.binファイルのリンクアドレスは0である（位置独立コード

で、プログラムが任意の可用メモリで実行できます）、BL2.binファイルのリンクアドレス0x23E00000（位置に依存するコードで、すべてのプログラムは当アドレスで実行します）BL1.binは、SDカードのセクタ1にプログラミングする必要があり、BL2.binはSDカードのセクタ49にプログラミングします。理由は後ほど説明します。プログラミング・プロセスは本章の第四節をご参照ください。**プログラミング・プロセス：**システム通電後、SDカードのセクタ1のBL1.bin を IRAM の 0xD0020000までにコピーし、実行します；本コードは先にDRAMを初期化し、続いてSDカードのセクタ49のBL2.binをDRAM の 0x23E00000までにコピーし、最後にプログラムが当アドレスにジャンプし、実行します。

1. BL1/start.S

前章と比べったら、本章の start.S に２つの手順を追加しました：

ステップ1　mem_init関数を呼び出し、メモリを初期化ます、機能の実現はmemory.Sにおけます、このファイルは、ubootから調達されます；

ステップ2 copy_code_to_dram（）を呼び出し、SDカードからBL2.binファイルをDRAM 0x23E00000にコピーします。copy_code_to_dram（）はmmc_relocate.cファイルにおけます；

2. BL1/memory.S

S5PV210は既にDDR2 DRAM初期化の仕方を解説しました、主に3つの主要手順にあります：PHY DLL初始化、DMC初始化、DDR2 DRAM初期化です。更に詳しいなら27のステップがあります、多くのレジスタを使用しますので、説明には時間が掛かり過ぎますので、本文ではレジスタ·セットには要点のみを説明します。DDR2　DRAM完全初期化手順下記図をご参照ください：

1.2.1.3 DDR2

Initialization sequence for DDR2 memory type:

1. To provide stable power for controller and memory device, the controller must assert and hold CKE to a logic low level. Then apply stable clock. **Note**: XDDR2SEL should be High level to hold CKE to low.
2. Set the **PhyControl0.ctrl_start_point** and **PhyControl0.ctrl_inc** bit-fields to correct value according to clock frequency. Set the **PhyControl0.ctrl_dll_on** bit-field to '1' to turn on the PHY DLL.
3. DQS Cleaning: Set the **PhyControl1.ctrl_shiftc** and **PhyControl1.ctrl_offsetc** bit-fields to correct value according to clock frequency and memory tAC parameters.
4. Set the **PhyControl0.ctrl_start** bit-field to '1'.
5. Set the **ConControl**. At this moment, an auto refresh counter should be off.
6. Set the **MemControl**. At this moment, all power down modes should be off.
7. Set the **MemConfig0** register. If there are two external memory chips, set the MemConfig1 register.
8. Set the **PrechConfig** and **PwrdnConfig** registers.
9. Set the **TimingAref**, **TimingRow**, **TimingData** and **TimingPower** registers according to memory AC parameters.
10. If QoS scheme is required, set the **QosControl0~15** and **QosConfig0~15** registers.
11. Wait for the **PhyStatus0.ctrl_locked** bit-fields to change to '1'. Check whether PHY DLL is locked.
12. PHY DLL compensates the changes of delay amount caused by Process, Voltage and Temperature (PVT) variation during memory operation. Therefore, PHY DLL should not be off for reliable operation. It can be off except runs at low frequency. If off mode is used, set the **PhyControl0.ctrl_force** bit-field to correct value according to the **PhyStatus0.ctrl_lock_value[9:2]** bit-field to fix delay amount. Clear the **PhyControl0.ctrl_dll_on** bit-field to turn off PHY DLL.
13. Confirm whether stable clock is issued minimum 200us after power on
14. Issue a **NOP** command using the **DirectCmd** register to assert and to hold CKE to a logic high level.

15. Wait for minimum 400ns.
16. Issue a **PALL** command using the **DirectCmd** register.
17. Issue an **EMRS2** command using the **DirectCmd** register to program the operating parameters.
18. Issue an **EMRS3** command using the **DirectCmd** register to program the operating parameters.
19. Issue an **EMRS** command using the **DirectCmd** register to enable the memory DLLs.
20. Issue a **MRS** command using the **DirectCmd** register to reset the memory DLL.
21. Issue a **PALL** command using the **DirectCmd** register.
22. Issue two **Auto Refresh** commands using the **DirectCmd** register.
23. Issue a **MRS** command using the **DirectCmd** register to program the operating parameters without resetting the memory DLL.
24. Wait for minimum 200 clock cycles.
25. Issue an **EMRS** command using the **DirectCmd** register to program the operating parameters. If OCD calibration is not used, issue an **EMRS** command to set OCD Calibration Default. After that, issue an **EMRS** command to exit OCD Calibration Mode and to program the operating parameters.
26. If there are two external memory chips, perform steps 14~25 for chip1 memory device.
27. Set the **ConControl** to turn on an auto refresh counter. 28. If power down modes is required, set the **MemControl** registers.

DRAMの完全初期化手順

memory.Sは上記の知識を参照して、メモリを初期化しました。主要手順は次の4つのステップです。

ステップ1　DRAMのドライバ強度（メモリアクセス信号強度）を設定

DRAM Driver Strength の値は大きほど、メモリアクセス信号強度が大きい。メモリは動作周波数には敏感で、動作周波数はメモリの公称周波数より高い場合、当オプションの値を大きくすると、コンピュータがオーバークロック状態での安定性を向上させることができます。ここではデフォルト値を使用します。

ステップ2 DDR型DRAMは、DLL（Delay Locked Loop 遅延ロックループがデータ･ストローブ信号を提供する）技術を使用します、データがオンな場合、メモリコントローラはこのデータ･ストローブ信号で、正確にデータを検索できます。ここでは、あまりにも深く掘る必要はありません、く27ステップの2～4の手順に従い、PHYのDLLを初期化するだけで十分です。関連するコードは、既に解説を持って、ここにコードを貼り付けていません。

ステップ3　DMC0を初期化

27ステップの 5~9と対応し、初期化します、詳しいレジスタ・セットはコードに参照ください。

ステップ4　DDR2 DRAMを初期化

DRAMの初期化はレジスタに DIRECTCMDコマンドを入れるだけで、7ステップの 16~23をご参照ください。

3. BL1/mmc_relocate.c

mem_init関数でDRAMを初期化後、DRAMへのコードをコピーし、実行できます。この機能はBL1ディレクトリ下のmmc_relocate.cで実現します。mmc_relocate.cコードは、下記のとおりです。

```
void copy_code_to_dram(void)
{
unsigned long ch;
void (*BL2)(void);
ch = *(volatile unsigned int *)(0xD0037488);
copy_sd_sd_to_mem copy_bl2 = (copy_sd_sd_to_mem) (*(unsigned int *) (0xD0037F98));
unsigned int ret;
//  通路  0
if (ch == 0xEB000000)
{
        // 0:channel 0
        // 49:ソース、コードはセクタ49にあり、1 sector = 512 bytes
        // 32:長さ、こぴー  32 sector、  16K
        // 0x23E00000:目的地、リンクアドレス  0x23E00000
        ret = copy_bl2(0、  49、  32、(unsigned int *)0x23E00000、  0);
}
//   通路   2
else if (ch == 0xEB200000)
{
        ret = copy_bl2(2、  49、  32、(unsigned int *)0x23E00000、  0);
}
else
        return;
// DRAM  にジャンプ
BL2 = (void *)0x23E00000;
(*BL2)();
}
```

まず、関数ポインタcopy_blを定義して、値を0xD0037F98に与えます。IROM内部硬化のコードは既に、クラスのコピー機能があります、 SDカードからデータをDRAMコピーするのも含めます。関数のアドレス配置は下記図をご参照ください。

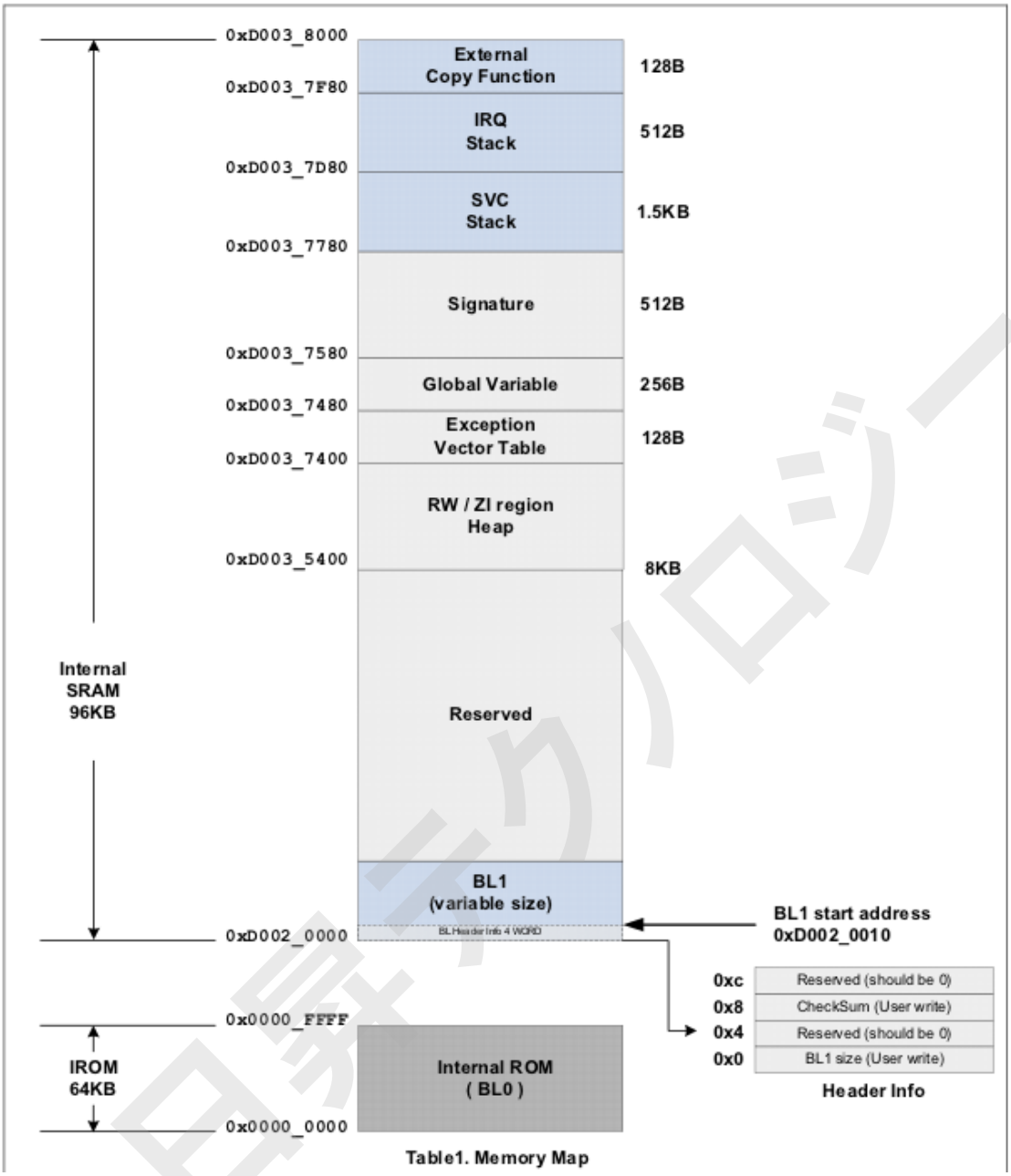


Table1. Memory Map

上記図により、External Copy Function の位置は0xD0037F80~0xD0038000です、其SDカードからDRAMへデータをコピする機能の関数はアドレス0xD0037F98におけます、コードのプロトタイプは下記の通りです：：

- **SD/MMC Copy Function Address**

External source clock parameter is used to fit EPLL source clock at 20MHz.

```
/**
 * This Function copy MMC(MoviNAND/iNand) Card Data to memory.
 * Always use EPLL source clock.
 * This function works at 20Mhz.
 * @param u32 StartBlkAddress : Source card(MoviNAND/iNand MMC)) Address.(It must block address.)
 * @param u16 blockSize : Number of blocks to copy.
 * @param u32* memoryPtr : Buffer to copy from.
 * @param bool with_init : determined card initialization.
 * @return bool(u8) - Success or failure.
 */
#define CopySDMMCtoMem(z,a,b,c,e)(((bool(*)(int, unsigned int, unsigned short, unsigned int*, bool))(*((unsigned
int *)0xD0037F98)))(z,a,b,c,e))
```

StartBlkAddress:コピー開始のセクタナンバー、１セクタの単位は512バイト

blockSize：セクタのコピー数
memoryPtr：DRAMのどのアドレスにコピーする
with_init：ＳＤカードの初期化必要の判別

上記の知識があれば、copy_code_to_dram機能を理解することは簡単です。アドレス0xD0037488値を読み出しことによってチャネル0または1を使用するかどうかを判断できます、チップマニュアルで“sd/MMC/eMMC boot - MMC Channel0 is used for first boot. And Channel 2 is used for Second boot` 、BL1.bin はfirst boot、チャネル0を使用します、 CopysdMMCtoMem 関数を呼び出し、 BL2.bin は sd カードのセクタ49からDRAMの0x23E00000アドレスにコピします、コピの長さは16K。最後に、BL2関数ポインタに0x23E0000の値を与え、BL2関数を呼び出しで、関数は0x23E0000にジャンプして、BL2.binのコードを実行できます。

4. BL2/start.S

BL1.binは 0x23E00000にジャンプした後、実行するのは start.S のコードです、BL2.binリンカスクリプトsdram.ldsがコードセグメントの初期でをstart.oを置くことを指定された。BL2/start.Sで実行するのは、位置指令を使用することです：ldr PC、 =main は main 関数を呼び出し、main関数の機能は前章のコードと同じ、LEDを点滅することです。

第三節　コードコンパイルとプログラミングの実行

SD カードを PC にアクセスして、Fedora 端末で下記のコマンドを実行します：

cd 6.sdram

make

chmod 777 write2sd

./write2sd

makeを実行後、210.binファイルは生成します、./write2sd実行し、.binはsdカードセクタ1にプログラミングします。

write2sd の内容は下記の通りです：

#!/bin/sh

sudo dd iflag=dsync oflag=dsync if=./BL1/BL1.bin of=/dev/sdb seek=1

sudo dd iflag=dsync oflag=dsync if=./BL2/BL2.bin of=/dev/sdb seek=49

./write2sdを実行後、BL1.binは、SDカードのセクタ1にプログラムされ、BL2.binはSDカードのセクター49にプログラムされます。

注：開発ボードにプログラム方法を同じです、以後の章では特に説明致しません。

第四節　実験現象

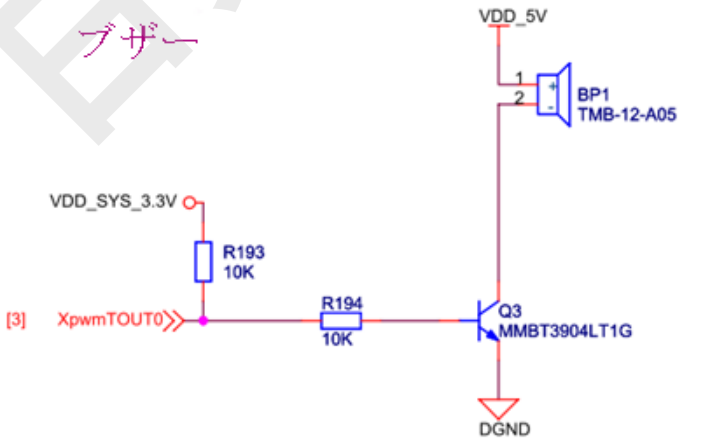
SDカードをMini210Sに挿入して、SDカードを起動・通電します。次の現象を確認できます：

LEDが正常に点滅します、実験現象は前の章と同じですが、実行中のプログラムには大きな違いがあります。ここまで、私達はリロケーションの基本知識を習得します。次からはより順調開発するために、便利なUSBダウンロード工具をご紹介します。

第 VIX 章　ブザー制御

第一節　回路図

Mini210S にはブザーが付きます、この章では、ブザーを制御する方法を学ぶようになります。回路図は下記の通り：



関連ピン：

第二節　プログラム説明

完全なコードは、ディレクトリ 8.buzzer ご参照ください。

ブザーを制御するのは簡単です、その原理は LED と同じ、GPDO_O ピンの制御によりブザーを制御することができます。

1. start.S

start.S は次の三つのプロセスがあります：

ステップ 1 ウォッチドッグを閉じる；

ステップ 2 スタックを設定する、Superboot は DRAM を初期化するために、スタックを DRAM の最後 0x40000000 に設定します；

ステップ 3 main 関数を呼び出し；

2. buzzer.c

完全コード：

```
#define GPD0CON                 (*(volatile unsigned long *)0xE02000A0)
#define GPD0DAT                 (*(volatile unsigned long *)0xE02000A4)
void buzzer_init(void)
{
    GPD0CON |= 1<<0;
}
 void buzzer_on(void)
{
    GPD0DAT |= 1<<0;
}
void buzzer_off(void)
{
    GPD0DAT &= ~(1<<0);
}
```

関数 Buzzer_init（）は GPIO ピンを設定、GPD0_0 を入力機能に設定します；

関数 buzzer_on()をピン GPD0_0 に 0 を出力して、プザーが鳴きます；

関数 buzzer_off()ピン　GPD0_0 に 1 を出力して、プザーが鳴きません；

3. main.c

次に main.c では、buzzer_init（）を呼び出し、ブザーを初期化します、続いて while ループでブザーの鳴き/停止を制御します。

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

cd 8.buzzer

make

8.buzzer のディレクトリ下に buzzer.bin を生成し、それを開発ボードにプログラムします。

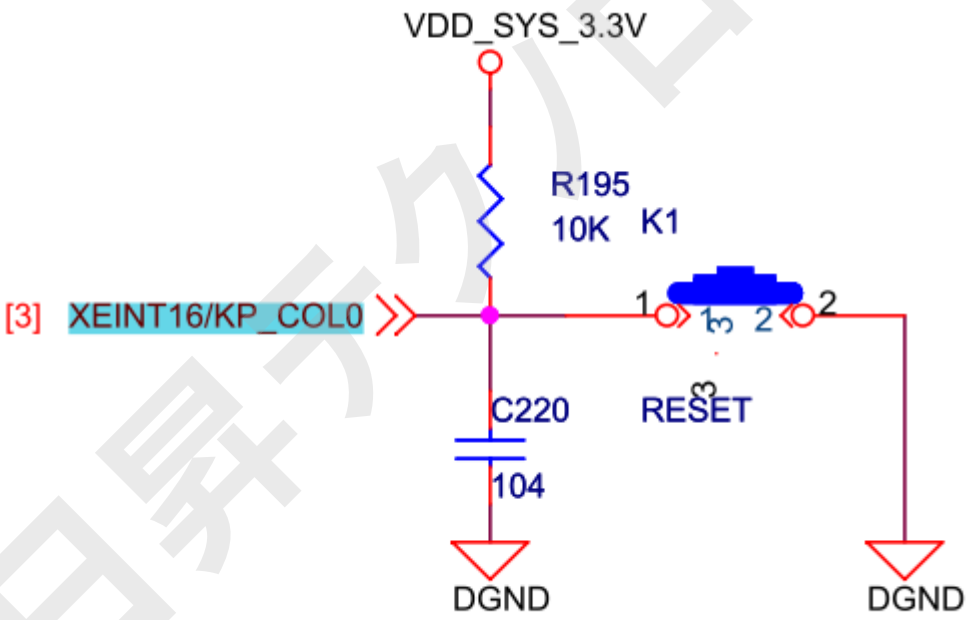
第四節 実験現象

実験現象は簡単で、開発ボードのブザーが鳴き始めます。

第 X 章　クエリモード・検出キー

第一節　回路図

Mini210S には 4 つのユーザーキーがあります、下記は key1 回路図、残りの 3 つのボタンの回路図は KEY1 を類似します：



キー回路図

“XEINT16/KP_COL0” を検索、

キー・ピン図

第二節　プログラム説明

完全なコードは、ディレクトリ 9.key_led ご参照ください。main.c 中のコードを抜きました、他のコードは 8.buzzer とほぼ同じです。

1. main.c

コアコードは：

```
// ポーリングモードでキーイベントを検索
while(1)
{
    dat = GPH2DAT;
    if(dat & (1<<0))            // KEY1 押され、 LED1 点灯、or LED1 消し
        GPJ2DAT |= 1<<0;        // OFF
    else
        GPJ2DAT &= ~(1<<0);             // ON
    if(dat & (1<<1))            // KEY2 押され、 LED2 点灯、or LED2 消し
        GPJ2DAT |= 1<<1;
    else
        GPJ2DAT &= ~(1<<1);
    if(dat & (1<<2))            // KEY3 押され、LED3 点灯、or LED3 消し
        GPJ2DAT |= (1<<2);
    else
        GPJ2DAT &= ~(1<<2);
    if(dat & (1<<3))            // KEY4 押され、LED4 点灯、or LED4 消し
        GPJ2DAT |= 1<<3;
    else
        GPJ2DAT &= ~(1<<3);
}
```

プログラムは簡単で、まず GPJ2_0/1/2/3 ピンを出力機能・GPH2_0/1/2/3 ピンを入力機能と配置し、そしてポーリング検出方式で GPH2_0/1/2/3 ピンの値を読み取って、そしてキーが押されたと検出された時、対応のピンをローレベルと変えて、対応する LED が点灯します、そうではいと、LED が消灯状態に留まります。

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 9.key_led
# make
```

9.key_led のディレクトリ下に key_led.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

ボタンが押されていない、すべての LED が消灯状態を保てます。キーkey1 を押すと、LED1 が点灯し、キーkey1 を離すと、LED1 が消灯します。残りの 3 つのボタンも同じようになります。ポーリング検出方式では CPU 使用率が掛かり過ぎます、ＣＰＵが他の作業をできませんので、後の章では割り込み方式でキーの押し状態を検査します。

第 XI 章　システムクロック初期化

第一節　S5PV210 クロックシステム

S5PV210 は、三種類のクロック：メイン·システムクロック domain(MSYS)、表示クロック domain（DSYS）、ペリフェラル·クロック·domain（PSYS）を含みます。

1) MSYS：Cortex A8 プロセッサ、DRAM コントローラ、3D、IRAM、IROM、割り込みコントローラなどにクロックを提供します；

2) DSYS：相関の表示メンバ単位にクロックを提供します、 FIMC、 FIMD、 JPEG、 and multimedia IPs など；

3) PSYS:ペリフェラルにクロックを提供する、 i2s、 spi、i2c、uart 等

Mini210S 接続の外部水晶発振器の発振周波数（Fin）は 24MHz、クロックでロジック PLL を制御することにより、システムクロックを向上させることができます。S5PV210 には合計 4 つの周波数逓倍器（ダブラー）があります。APLL（MSYS 用）、MPLL（DSYS 使用）、EPLL（PSYS 用）VPLL（video 関連クロック用）を含む、合わせればすなわちＰＬＬです。三種類のクロック domain 中、異なる分周器を使用し、各部品にクロックを出力できます。各クロックの関係図は下記の通りです：

Clocks have the following relationship:

- MSYS clock domain
  - freq(ARMCLK) = freq(MOUT_MSYS) / n, where n = 1 ~ 8
  - freq(HCLK_MSYS) = freq(ARMCLK) / n, where n = 1 ~ 8
  - freq(PCLK_MSYS) = freq(HCLK_MSYS) / n, where n = 1 ~ 8
  - freq(HCLK_IMEM) = freq(HCLK_MSYS) / 2
- DSYS clock domain
  - freq(HCLK_DSYS) = freq(MOUT_DSYS) / n, where n = 1 ~ 16
  - freq(PCLK_DSYS) = freq(HCLK_DSYS) / n, where n = 1 ~ 8
- PSYS clock domain
  - freq(HCLK_PSYS) = freq(MOUT_PSYS) / n, where n = 1 ~ 16
  - freq(PCLK_PSYS) = freq(HCLK_PSYS) / n, where n = 1 ~ 8
  - freq(SCLK_ONENAND) = freq(HCLK_PSYS) / n, where n = 1 ~ 8

S5PV210 クロック分類図

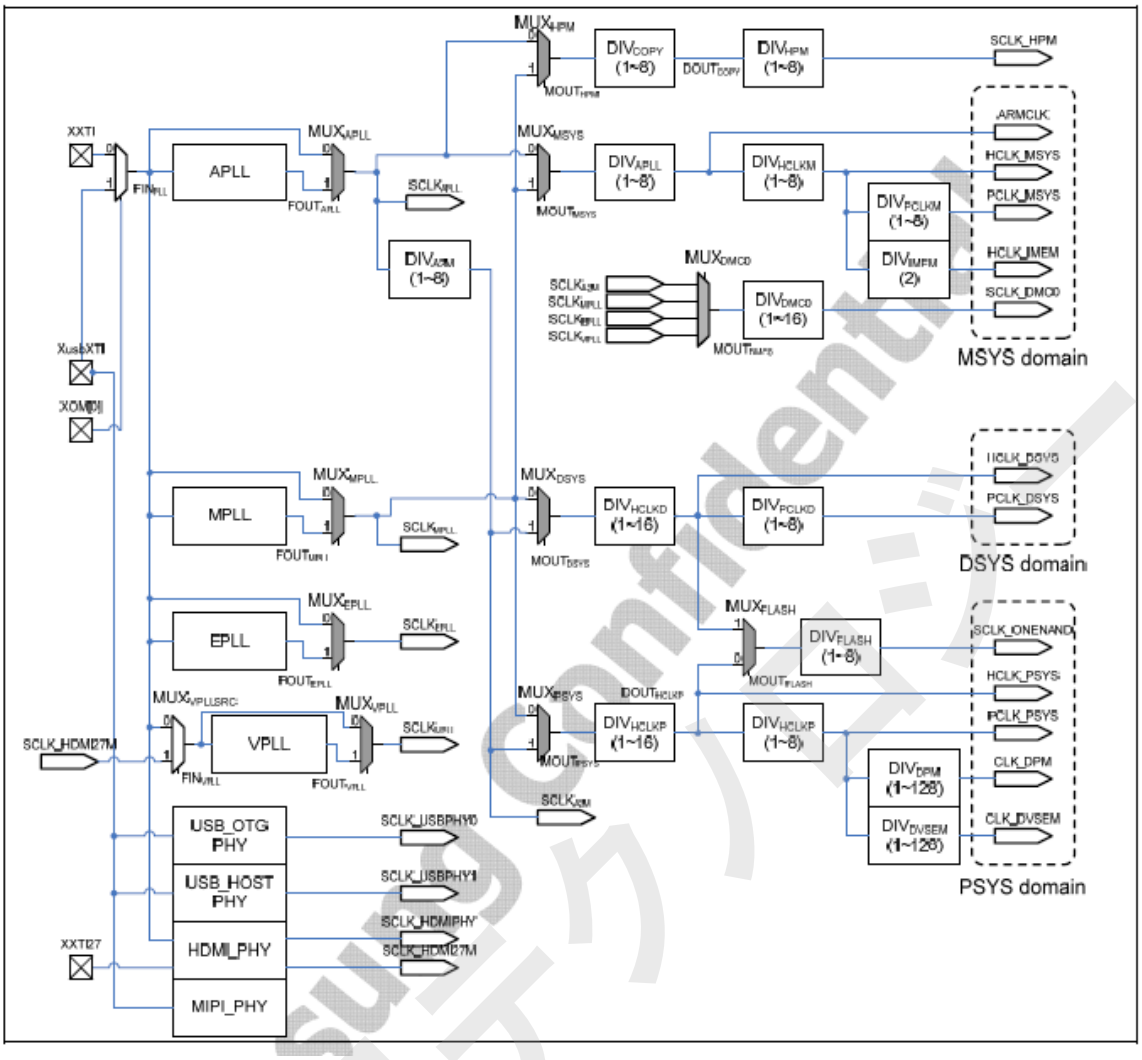
以下に示すように、各種類クロック、チップの値の確定はマニュアル基準値をご参照ください。

Values for the high-performance operation:

- freq(ARMCLK) = 1000 MHz
- freq(HCLK_MSYS) = 200 MHz
- freq(HCLK_IMEM) = 100 MHz
- freq(PCLK_MSYS) = 100 MHz
- freq(HCLK_DSYS) = 166 MHz
- freq(PCLK_DSYS) = 83 MHz
- freq(HCLK_PSYS) = 133 MHz
- freq(PCLK_PSYS) = 66 MHz
- freq(SCLK_ONENAND) = 133 MHz, 166 MHz

S5PV210 クロック基準値設定図

各クロックの設定は、下記図（チップマニュアル P361）を参照してください：



S5PV210 クロック設定図

上記図は大切で、これにを参照して、ハードウェア･コンポーネントが必要なクロックを全部設定できます。（状況により、必要なハードウェアコンポーネントの作動クロックを設定すれば十分です）、第二節で、相関のレジスタを設定し、クロックを初期化します。

第二節　プログラム説明

完全なコードは、ディレクトリ 10.clock_s 、 11.clock_c d ご参照ください。

この章に関わるコードは二組あります。10.clock_s（アセンブリでクロック初期化）と 11.clock_c（C 言語でクロック初期化）、両者の本質は同じで、実行効果も同じです、アセンブリを強化したいなら、10.clock_s 中のコードをご参照ください。C 言語のコードはより明確なアイデア、理解しやすいため、次は 11.clock_c でご説明します。

1. start.S

main 関数を呼び出し前に、クロック初期化関数 clock_init で相関設定を行います。

2. clock.c

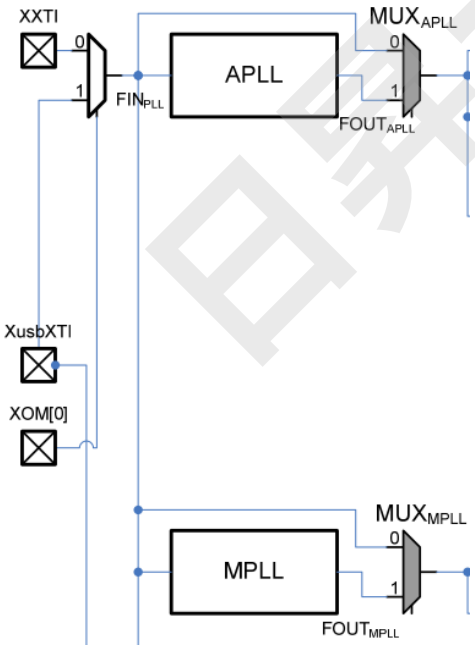
clock_init()は clock.c で定義します、コード：

```
void clock_init()
{
// 1 各クロック･スイッチを設定、PLL は使用しません
CLK_SRC0 = 0x0;
// 2 ロック時間を設定、デフォルト値を使用します
APLL_LOCK = 0x0000FFFF;
MPLL_LOCK = 0x0000FFFF;
// 3 分周を設定
CLK_DIV0 = 0x14131440;
// 4 PLL を設定
APLL_CON0 = APLL_VAL;
MPLL_CON = MPLL_VAL;
// 5 各クロック･スイッチを設定、PLL は使用します
CLK_SRC0 = 0x10001111;
}
```

上記コードは 5 つのステップがあります。次は分解説明します。:

ステップ 1 各クロック･スイッチを設定、PLL は使用しません

第一節の設定図（チップマニュアル P361）を参照します、下記は拡大図：



まず外部 24MHz 水晶発振器を使用します、上記図により、APLL と MPLL のクロック･ソースは"FINPLL`

で決めさせます、チップマニュアルで"FINPLL゛を検索し、対応レジスタは CLK_SRC0 とわかります。

3.7.3.1 Clock Source Control Registers (CLK_SRC0, R/W, Address = 0xE010_0200)

CLK_SRC0	Bit	Description	Initial State
Reserved	[31:29]	Reserved	0x0
ONENAND_SEL	[28]	Control MUXFLASH (0:HCLK_PSYS, 1:HCLK_DSYS)	0
Reserved	[27:25]	Reserved	0x0
MUX_PSYS_SEL	[24]	Control MUX_PSYS (0:SCLKMPLL, 1:SCLKA2M)	0
Reserved	[23:21]	Reserved	0x0
MUX_DSYS_SEL	[20]	Control MUX_DSYS (0:SCLKMPLL, 1:SCLKA2M)	0
Reserved	[19:17]	Reserved	0x0
MUX_MSYS_SEL	[16]	Control MUX_MSYS (0:SCLKAPLL, 1:SCLKMPLL)	0
Reserved	[15:13]	Reserved	0x0
VPLL_SEL	[12]	Control MUXVPLL (0: FINVPLL, 1: FOUTVPLL)	0
Reserved	[11:9]	Reserved	0x0
EPLL_SEL	[8]	Control MUXEPLL (0:FINPLL, 1:FOUTEPLL)	0
Reserved	[7:5]	Reserved	0x0
MPLL_SEL	[4]	Control MUXMPLL (0:FINPLL, 1:FOUTMPLL)	0
Reserved	[3:1]	Reserved	0x0
APLL_SEL	[0]	Control MUXAPLL (0:FINPLL, 1:FOUTAPLL)	0

PLL およびサブ周波数係数を設定する前に PLL を使用できません。安全のために、先に、低周波数の外部 24MHz 水晶分周器を使用します。PLL およびサブ周波数係数を設定完了後、クロック・スイッチの再設定を行います。

ステップ 2 ロック時間を設定

PLL 設定後、クロックは Fin から目標周波数に引き上げるには、時間が必要です。すなわちロック時間です。

ステップ 3 分周を設定

分周相関のレジスタは CLK_DIV0、下記図をご参照ください：

3.7.4.1 Clock Divider Control Register (CLK_DIV0, R/W, Address = 0xE010_0300)

CLK_DIV0	Bit	Description	Initial State
Reserved	[31]	Reserved	0
PCLK_PSYS_RATIO	[30:28]	DIVPCLKP clock divider ratio, PCLK_PSYS = HCLK_PSYS / (PCLK_PSYS_RATIO + 1)	0x0
HCLK_PSYS_RATIO	[27:24]	DIVHCLKP clock divider ratio, HCLK_PSYS = MOUT_PSYS / (HCLK_PSYS_RATIO + 1)	0x0
Reserved	[23]	Reserved	0
PCLK_DSYS_RATIO	[22:20]	DIVPCLKD clock divider ratio, PCLK_DSYS = HCLK_DSYS / (PCLK_DSYS_RATIO + 1)	0x0
HCLK_DSYS_RATIO	[19:16]	DIVHCLKD clock divider ratio, HCLK_DSYS = MOUT_DSYS / (HCLK_DSYS_RATIO + 1)	0x0
Reserved	[15]	Reserved	0
PCLK_MSYS_RATIO	[14:12]	DIVPCLKM clock divider ratio, PCLK_MSYS = HCLK_MSYS / (PCLK_MSYS_RATIO + 1)	0x0
Reserved	[11]	Reserved	0
HCLK_MSYS_RATIO	[10:8]	DIVHCLKM clock divider ratio, HCLK_MSYS = ARMCLK / (HCLK_MSYS_RATIO + 1)	0x0
Reserved	[7]	Reserved	0
A2M_RATIO	[6:4]	DIVA2M clock divider ratio, SCLKA2M = SCLKAPLL / (A2M_RATIO + 1)	0x0
Reserved	[3]	Reserved	0
APLL_RATIO	[2:0]	DIVAPLL clock divider ratio, ARMCLK = MOUT_MSYS / (APLL_RATIO + 1)	0x0

本章第一節のクロック設定でレジスタを設定します。

Values for the high-performance operation:

- freq(ARMCLK) = 1000 MHz
- freq(HCLK_MSYS) = 200 MHz
- freq(HCLK_IMEM) = 100 MHz
- freq(PCLK_MSYS) = 100 MHz
- freq(HCLK_DSYS) = 166 MHz
- freq(PCLK_DSYS) = 83 MHz
- freq(HCLK_PSYS) = 133 MHz
- freq(PCLK_PSYS) = 66 MHz
- freq(SCLK_ONENAND) = 133 MHz, 166 MHz

- ARMCLK = 1000MHz = MOUT_MSYS / (APLL_RATIO + 1)、 ステップ 4~5 のあと、MOUT_MSYS は 1000MHz と設定します、APLL_RATIO=0
- SCLKA2M=200MHz=SCLKAPLL / (A2M_RATIO + 1)、 SCLKAPLL=1000MHz、 A2M_RATIO=4。
- HCLK_MSYS=200MHz=ARMCLK / (HCLK_MSYS_RATIO + 1)、HCLK_MSYS_RATIO=4
- PCLK_MSYS=100MHz=HCLK_MSYS / (PCLK_MSYS_RATIO + 1)、 PCLK_MSYS_RATIO=1
- HCLK_DSYS=166MHz=MOUT_DSYS / (HCLK_DSYS_RATIO + 1)、 ステップ 4~5 のあと、

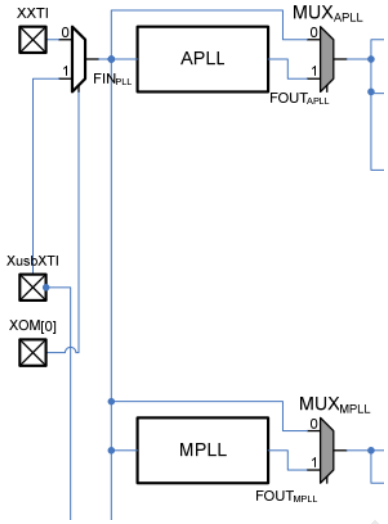
MOUT_DSYS =667MHz、 HCLK_DSYS_RATIO=3

- PCLK_DSYS=83MHz=HCLK_DSYS / (PCLK_DSYS_RATIO + 1)、PCLK_DSYS_RATIO=1
- HCLK_PSYS=133Mhz=MOUT_PSYS / (HCLK_PSYS_RATIO + 1)、ステップ 4~5 のあと、MOUT_PSYS =667MHz 、 HCLK_PSYS_RATIO=4
- PCLK_PSYS=66Mhz=HCLK_PSYS / (PCLK_PSYS_RATIO + 1)、 HCLK_PSYS_RATIO=1、

CLK_DIV0 = 0x14131440;

ステップ 4 PLL を設定

PLL、すなわち周波数逓倍器、動作周波数を拡大するために使用されます。分周器を設定し、PLL を設定する必要があります。 APLL / MPLL は APLL_CON0/MPLL_CON レジスタにより起動します、まずレジスタの設定を行います。



APLL_CON0

APLL_CON0	Bit	Description	Initial State
ENABLE	[31]	PLL enable control (0: disable, 1: enable)	0
Reserved	[30]	Reserved	0
LOCKED	[29]	PLL locking indication 0 = Unlocked 1 = Locked Read Only	0
Reserved	[28:26]	Reserved	0x0
MDIV	[25:16]	PLL M divide value	0xC8
Reserved	[15:14]	Reserved	0
PDIV	[13:8]	PLL P divide value	0x3
Reserved	[7:3]	Reserved	0
SDIV	[2:0]	PLL S divide value	0x1

ALPP_CON0 は APLL、FINPLL=24MHz に設定します、 APLL を通じ、 クロック分周は FOUT=1000Mhz を出力します、FOUT の計算公式は下記の通りです：

FOUT=MDIV*FIN/(PDIV*2^(SDIV-1)) = 1000 MHz

FIN=24MHz、FOUT=1000MHz、その値は： MDIV= 0x7d、PDIV= 0x3、SDIV=1。 三つの値は固定値ではありません、 FOUT=1000Mhz をサポートするなら、任意値を使用できます。

MPLL_CON

3.7.2.2 PLL Control Registers (MPLL_CON, R/W, Address = 0xE010_0108)

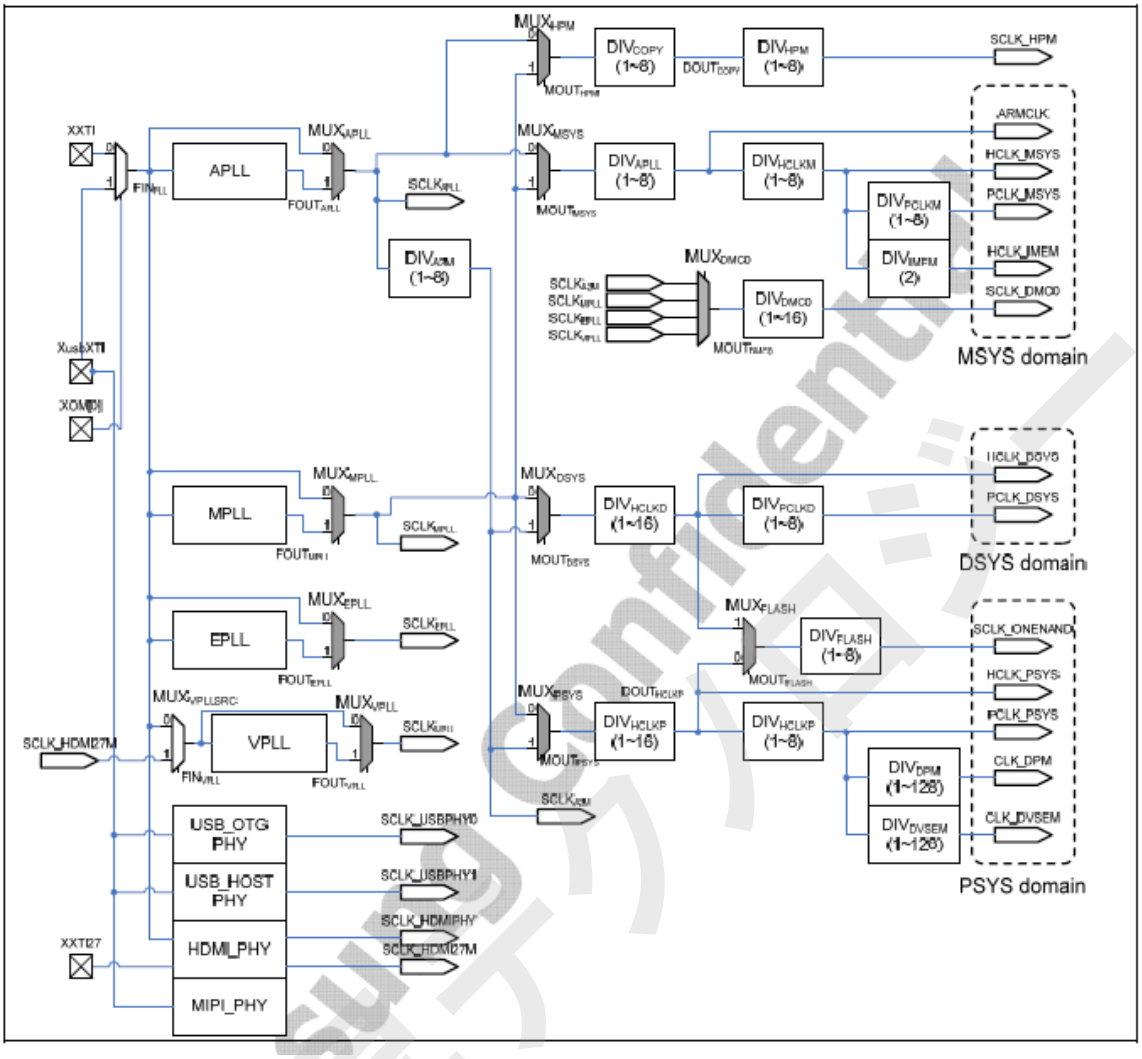
MPLL_CON	Bit	Description	Initial State
ENABLE	[31]	PLL enable control (0: disable, 1: enable)	0
Reserved	[30]	Reserved	0
LOCKED	[29]	PLL locking indication 0 = Unlocked 1 = Locked Read Only	0
Reserved	[28]	Reserved	0
VSEL	[27]	VCO frequency range selection	0x0
Reserved	[26]	Reserved	0
MDIV	[25:16]	PLL M divide value	0x14D
Reserved	[15:14]	Reserved	0
PDIV	[13:8]	PLL P divide value	0x3
Reserved	[7:3]	Reserved	0
SDIV	[2:0]	PLL S divide value	0x1

MPLL_CON レジスタは MPLL を設定します、 MPLL を通じ、クロック分周は FOUT=667Mhz を出力します、FOUT の計算公式は下記の通りです：

FOUT=MDIV*FIN/(PDIV*2^SDIV) = 667 MHz

FIN=24MHz、FOUT=667MHz、その値は： MDIV=0x29B、PDIV= 0xC、SDIV=1。三つの値は固定値ではありません、 FOUT=1000Mhz をサポートするなら、任意値を使用できます。

ステップ 5　各クロック･スイッチを設定



S5PV210 时钟设置参考图

上記図のように、全ての MUX はクロックを選択すると使用されます、関連レジスタは CLK_SRC0、下記図をご参照ください：

3.7.3.1 Clock Source Control Registers (CLK_SRC0, R/W, Address = 0xE010_0200)

CLK_SRC0	Bit	Description	Initial State
Reserved	[31:29]	Reserved	0x0
ONENAND_SEL	[28]	Control MUXFLASH (0:HCLK_PSYS, 1:HCLK_DSYS)	0
Reserved	[27:25]	Reserved	0x0
MUX_PSYS_SEL	[24]	Control MUX_PSYS (0:SCLKMPLL, 1:SCLKA2M)	0
Reserved	[23:21]	Reserved	0x0
MUX_DSYS_SEL	[20]	Control MUX_DSYS (0:SCLKMPLL, 1:SCLKA2M)	0
Reserved	[19:17]	Reserved	0x0
MUX_MSYS_SEL	[16]	Control MUX_MSYS (0:SCLKAPLL, 1:SCLKMPLL)	0
Reserved	[15:13]	Reserved	0x0
VPLL_SEL	[12]	Control MUXVPLL (0: FINVPLL, 1: FOUTVPLL)	0
Reserved	[11:9]	Reserved	0x0
EPLL_SEL	[8]	Control MUXEPLL (0:FINPLL, 1:FOUTEPLL)	0
Reserved	[7:5]	Reserved	0x0
MPLL_SEL	[4]	Control MUXMPLL (0:FINPLL, 1:FOUTMPLL)	0
Reserved	[3:1]	Reserved	0x0
APLL_SEL	[0]	Control MUXAPLL (0:FINPLL, 1:FOUTAPLL)	0

S5PV210 を参照して、各クロック・スイッチを設定します：

APLL_SEL=1、FOUTAPLL を使用します

MPLL_SEL=1、FOUTAPLL を使用します

EPLL_SEL=1、FOUTAPLL を使用します

VPLL_SEL=1、FOUTAPLL を使用します

MUX_MSYS_SEL=0、SCLKAPLL を使用します

MUX_DSYS_SEL=0、SCLKMPLL を使用します

MUX_PSYS_SEL=0、SCLKMPLL を使用します

ONENAND_SEL=1、HCLK_DSYS を使用します

CLK_SRC0=0x10001111;

3. main.c

main 関数で LED 点滅を実現します、コードは前のとほぼ同じです。

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

cd 11.clock_c

make

11.clock_c のディレクトリ下に clock.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

同じく、LED の点滅を確認できます、点滅周期は余り変わりませんが、それは Superboot がクロックを初期化したためです。クロックを初期化せず開発ボードの速度を実験したいなら、clock.c 内でマクロ PLL_OFF を定義し PLL 機能を OFF にします、こうすれば LED の点滅周期は大幅に下がります。本章のクロック設定は APLL と MPLL で行います、そしてハードウェアがほとんどが正常に動作できます。次の章では、シリアルポートを初期化することにより、端末で文字を入力・出力する機能を実現します。本章の知識も必要となります。

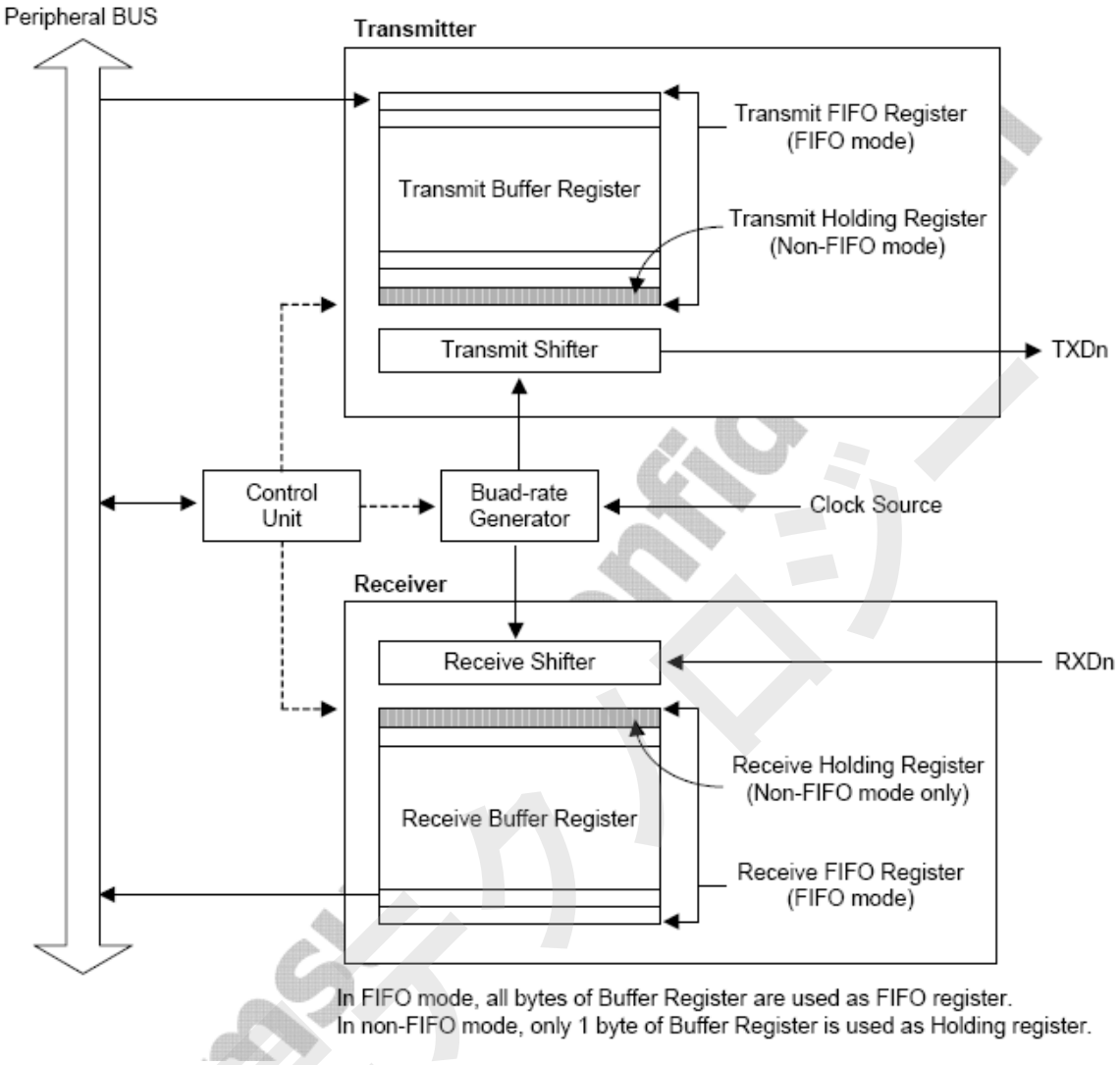
第 XII 章　シリアルポート設定-文字の入力および出力

第一節　S5PV210 UART 説明

汎用非同期レシーバトランスミッタは UART と略称します、すなわち UNIVERSAL ASYNCHRONOUS RECEIVER AND TRANSMITTER、シリアルデータを送信するために使用されます。データを送信する場合は、CPU がパラレルデータ UART に書き込まれ、UART は特定のフォーマットに従ってワイヤーでシリアル発行します；データを受信する場合は、UART は、もう一本のワイヤの信号を検出し、シリアルをバッファ内に収集します、そして CPU が UART のデータを読み取ることができます。

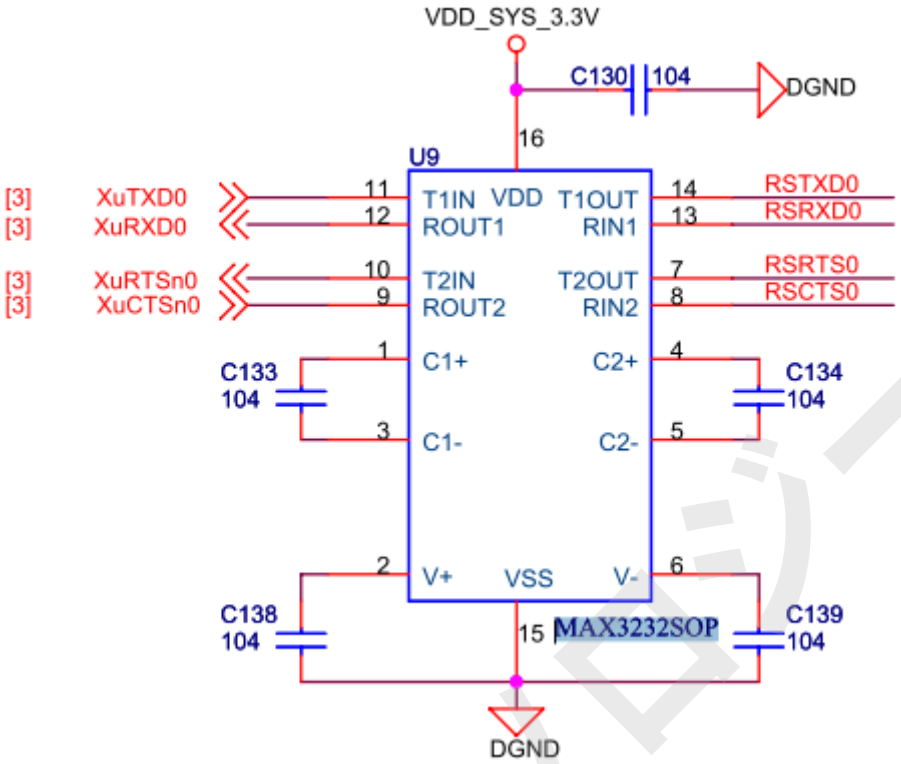
S5PV210 では、UART は 4 つの独立した非同期シリアル I / O ポートを提供し、4 つの独立したチャンネルがあり、各チャネルは DMA モードまたは割り込みモードで動作できます。その中、チャネル 0 は 256byte の受信 FIFO と送信 FIFO があります、チャネル 1 は 64byte の受信 FIFO と送信 FIFO があります、チャネル 2 と 3 は 16byte の受信 FIFO と送信 FIFO があります。

S5PV210 の UART 構造図は次のとおりです：

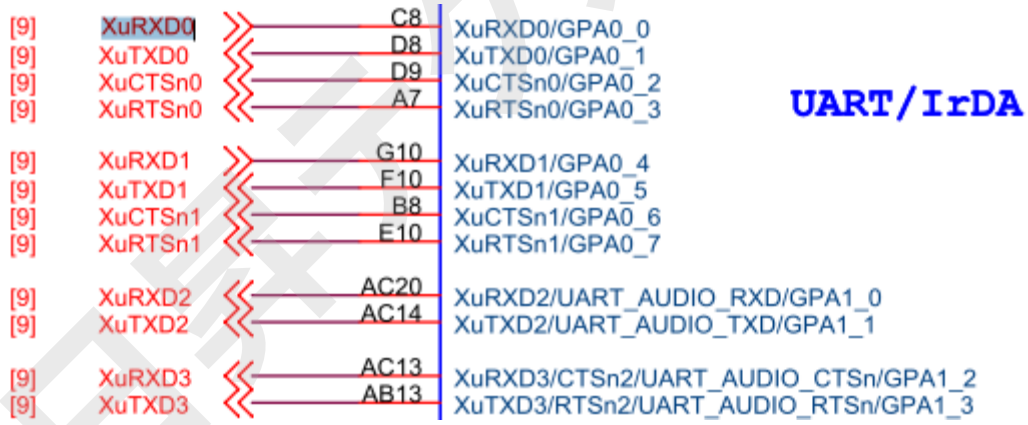


S5PV210 UART の構造図

UART は標準 TTL / CMCOS ロジック･レベルを使用してデータを表現します、そしてデータ抗干渉能力を強化し、伝送距離を向上させるために、TTL / CMOS ロジックレベルを RS-232 ロジック･レベルに変換します、概略図で見れば、Mini210S は MAX3232SOP チップを使用し、TX0 と DX0 を使用します：



"XuTXD0"を検索し、結果は次の通り示します：



UART ピン接続図

UART の関連レジスタを設定することにより、UART を駆動することができます、文字を送受信できます。

第二節　プログラム説明

完全なコードは、ディレクトリ 12.uart_putchar ご参照ください。前の章の 11.clock_c と比べ、main.c に uart.c ファイルを追加したことです。

1. main.c

完全なコードは次のとおりです：

```
int main()
```

```
{
    char c;
    uart_init();            // シリアルポートを初期化

    while (1)
    {
    c = getc ();            // 文字 c を受信
    putc(c+1);              // 文字 c1 を送信
    }
    return 0;
}
```

main 関数では、先に uart_init（）を呼び出しで UART を初期化します、そして、 getc で PC 側のデータ（文字）を受信します、続いて putc()を呼び出し、ＰＣへデータ（文字）＋1 で 返送します。

2. uart.c

uart_init()コード：

```
void uart_init()
{

// 1　RX / TX 用にピン・コンフィギュレーションを行います
GPA0CON = 0x22222222;
GPA1CON = 0x2222;

// 2　データ形式のセット
UFCON0 = 0x1;           // FIFO オンにする
UMCON0 = 0x0;           //フロー制御なし
ULCON0 = 0x3;           //データビット：8、パリティなし、ストップビット：1
UCON0 = 0x5;            //クロック：PCLK；割り込み禁止、UART の受信、送信をオンにする

// 3　ボーレートを設定
UBRDIV0 = UART_UBRDIV_VAL;          // 35
UDIVSLOT0 = UART_UDIVSLOT_VAL;      // 0x1
}
```

上記のコードは三つの手順があります、下記は分解説明します：：

ステップ 1　RX / TX 用にピン・コンフィギュレーションを行います

UART のピン接続図を参照して、GPA0CON と GPA1CON レジスタを設定し、GPA0、GPA1 ピンを UART 機能に使います。

2.2.2.1 Port Group GPA0 Control Register (GPA0CON, R/W, Address = 0xE020_0000)

GPA0CON	Bit	Description	Initial State
GPA0CON[7]	[31:28]	0000 = Input 0001 = Output 0010 = UART_1_RTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[7]	0000
GPA0CON[6]	[27:24]	0000 = Input 0001 = Output 0010 = UART_1_CTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[6]	0000
GPA0CON[5]	[23:20]	0000 = Input 0001 = Output 0010 = UART_1_TXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[5]	0000
GPA0CON[4]	[19:16]	0000 = Input 0001 = Output 0010 = UART_1_RXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[4]	0000
GPA0CON[3]	[15:12]	0000 = Input 0001 = Output 0010 = UART_0_RTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[3]	0000
GPA0CON[2]	[11:8]	0000 = Input 0001 = Output 0010 = UART_0_CTSn 0011 ~ 1110 = Reserved 1111 = GPA0_INT[2]	0000
GPA0CON[1]	[7:4]	0000 = Input 0001 = Output 0010 = UART_0_TXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[1]	0000
GPA0CON[0]	[3:0]	0000 = Input 0001 = Output 0010 = UART_0_RXD 0011 ~ 1110 = Reserved 1111 = GPA0_INT[0]	0000

GPA0CON レジスタ図

2.2.3.1 Port Group GPA1 Control Register (GPA1CON, R/W, Address = 0xE020_0020)

GPA1CON	Bit	Description	Initial State
GPA1CON[3]	[15:12]	0000 = Input 0001 = Output 0010 = UART_3_TXD 0011 = UART_2_RTSn 0100 ~ 1110 = Reserved 1111 = GPA1_INT[3]	0000
GPA1CON[2]	[11:8]	0000 = Input 0001 = Output 0010 = UART_3_RXD 0011 = UART_2_CTSn 0100 ~ 1110 = Reserved 1111 = GPA1_INT[2]	0000
GPA1CON[1]	[7:4]	0000 = Input 0001 = Output 0010 = UART_2_TXD 0011 = Reserved 0100 = UART_AUDIO_TXD 0101 ~ 1110 = Reserved 1111 = GPA1_INT[1]	0000
GPA1CON[0]	[3:0]	0000 = Input 0001 = Output 0010 = UART_2_RXD 0011 = Reserved 0100 = UART_AUDIO_RXD 0101 ~ 1110 = Reserved 1111 = GPA1_INT[0]	0000

GPA1CON レジスタ図

ステップ　2　データ形式のセット

<1> ULCON0　　データ形式を設定する、下記図を参照ください

There are four UART line control registers in the UART block, namely, ULCON0, ULCON1, ULCON2, and ULCON3.

ULCONn	Bit	Description	Initial State
Reserved	[31:7]	Reserved	0
Infrared Mode	[6]	Determines whether to use the Infrared mode. 0 = Normal mode operation 1 = Infrared Tx/Rx mode	0
Parity Mode	[5:3]	Specifies the type of parity generation to be performed and checking during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/ checked as 1 111 = Parity forced/ checked as 0	000
Number of Stop Bit	[2]	Specifies how many stop bits are used to signal end-of-frame signal. 0 = One stop bit per frame 1 = Two stop bit per frame	0
Word Length	[1:0]	Indicates the number of data bits to be transmitted or received per frame. 00 = 5-bit 01 = 6-bit 10 = 7-bit 11 = 8-bit	00

- Word Length = 11、8bit　　データの長さ；
- Number of Stop Bit = 0、1bit　　ストップビット；
- Parity Mode = 000、　　パリティなし；
- Infrared Mode =0、　　ノーマルモードを使用；

よって ULCON0=0x3

<2> 9UCON0 は UART のコンフィギュレーションレジスタ、下記図を参照ください

There are four UART control registers in the UART block, namely, UCON0, UCON1, UCON2 and UCON3.

UCONn	Bit	Description	Initial State
Reserved	[31:21]	Reserved	000
Tx DMA Burst Size	[20]	Tx DMA Burst Size 0 = 1 byte (Single) 1 = 4 bytes	0
Reserved	[19:17]	Reserved	000
Rx DMA Burst Size	[16]	Rx DMA Burst Size 0 = 1 byte (Single) 1 = 4 bytes	0
Reserved	[15:11]	Reserved	0000
Clock Selection	[10]	Selects PCLK or SCLK_UART (from Clock Controller) clock for the UART baud rate. 0 = PCLK: DIV_VAL1) = (PCLK / (bps x 16)) –1 1 = SCLK_UART: DIV_VAL1) = (SCLK_UART / (bps x 16)) –1	00
Tx Interrupt Type	[9]	Interrupt request type.(2) 0 = Pulse (Interrupt is requested when the Tx buffer is empty in the Non-FIFO mode or when it reaches Tx FIFO Trigger Level in the FIFO mode.) 1 = Level (Interrupt is requested when Tx buffer is empty in the Non-FIFO mode or when it reaches Tx FIFO Trigger Level in the FIFO mode.)	0
Rx Interrupt Type	[8]	Interrupt request type. (2) 0 = Pulse (Interrupt is requested when instant Rx buffer receives data in the Non-FIFO mode or when it reaches Rx FIFO Trigger Level in the FIFO mode.) 1 = Level (Interrupt is requested when Rx buffer is receiving data in the Non-FIFO mode or when it reaches Rx FIFO Trigger Level in the FIFO mode.)	0
Rx Time Out Enable	[7]	Enables/ Disables Rx time-out interrupts if UART FIFO is enabled. The interrupt is a receive interrupt. 0 = Disables 1 = Enables	0
Rx Error Status Interrupt Enable	[6]	Enables the UART to generate an interrupt upon an exception, such as a break, frame error, parity error, or overrun error during a receive operation. 0 = Does not generate receive error status interrupt. 1 = Generates receive error status interrupt.	0

UCONn	Bit	Description	Initial State
Loop-back Mode	[5]	Setting loop-back bit to 1 trigger the UART to enter the loop-back mode. This mode is provided for test purposes only. 0 = Normal operation 1 = Loop-back mode	0
Send Break Signal	[4]	Setting this bit trigger the UART to send a break during 1 frame time. This bit is automatically cleared after sending the break signal. 0 = Normal transmit 1 = Sends the break signal	0
Transmit Mode	[3:2]	Determines which function is able to write Tx data to the UART transmit buffer register. 00 = Disables 01 = Interrupt request or polling mode 10 = DMA mode 11 = Reserved	00
Receive Mode	[1:0]	Determines which function is able to read data from UART receive buffer register. 00 = Disables 01 = Interrupt request or polling mode 10 = DMA mode 11 = Reserved	00

NOTE:

- Receive Mode = 01 、割り込みモードまたはポーリングモード；
- Transmit Mode = 01、割り込みモードまたはポーリングモード；
- Send Break Signal = 0、通常送信；
- Loop-back Mode = 0、ループバック· モードを使用しない；
- ここでポーリングモードを使用しますので、 bit[6-9]は全部 0 となります；
- Clock Selection = 0、 PCLK UART の動作クロックとして使います；
- DMA を使用しますため、bit[16]と bit[20] 全部 0 となります；

よって UCON0 = 0x5

<3> UFCON0 和 UMCON0

二つのレジスタ相対簡単で、UFCON0 は FIFO をオンにします、UMCON0 はフロー制御の設定とします。

ステップ 3　ボーレートを設定

ボーレートは毎秒伝送のデータビット数です、2 つのレジスタに関わります：: UBRDIV0 和 UDIVSLOT0

UBRDIV n	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
UBRDIVn	[15:0]	Baud rate division value (When UART clock source is PCLK, UBRDIVn must be more than 0 (UBRDIVn >0))	0x0000

UDIVSLOT n	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
UDIVSLOTn	[15:0]	Select the slot where clock generator divide clock source	0x0000

ボーレート設定式 : UBRDIVn + (num of 1's in UDIVSLOTn)/16 = (PCLK / (bps x 16))−1、 Maximum Operating Frequency for Each Sub-block 図で、UART は PSYS 下で作動します、よって、 PCLK 即 PCLK_PSYS =

66.5MHz、ボーレート bps を 115200 と設定します、よって(66.5MHz/(115200 x 16)) - 1 = 35.08 = UBRDIVn + (num of 1's in UDIVSLOTn)/16、 UBRDIV0=35、UDIVSLOT0=0x1

getc()と putc()コード：

```
// 一文字受信
char getc(void)
{
    while ((UFSTAT0 & 0xff) == 0);          // RX FIFO は空の場合、待ちます
    return URXH0;                           //データを読み取り
}

//一文字送信
void putc(char c)
{
    while (UFSTAT0 & (1<<24));              // RX FIFO は満ちの場合、待ちます
    UTXH0 = c;                                      //データを書き込み
}
```

There are four UART transmit buffer registers in the UART block, namely, UTXH0, UTXH1, UTXH2 and UTXH3. UTXHn contains 8-bit data for transmission data.

UTXHn	Bit	Description	Initial State
Reserved	[31:8]	Reserved	-
UTXHn	[7:0]	Transmit data for UARTn	-

UART データ送信レジスタ

There are four UART receive buffer registers in the UART block, namely, URXH0, URXH1, URXH2 and URXH3. URXHn contains 8-bit data for received data.

URXHn	Bit	Description	Initial State
Reserved	[31:8]	Reserved	0
URXHn	[7:0]	Receive data for UARTn	0x00

UART データ受信レジスタ

UTRSTATn	Bit	Description	Initial State
Reserved	[31:3]	Reserved	0
Transmitter empty	[2]	This bit is automatically set to 1 if the transmit buffer register has no valid data to transmit, and the transmit shift register is empty. 0 = Not empty 1 = Transmitter (which includes transmit buffer and shifter register) empty	1
Transmit buffer empty	[1]	This bit is automatically set to 1 if transmit buffer register is empty. 0 = Buffer register is not empty 1 = Buffer register is empty (In Non-FIFO mode, Interrupt or DMA is requested. In FIFO mode, Interrupt or DMA is requested, if Tx FIFO Trigger Level is set to 00 (Empty)) If UART uses FIFO, check Tx FIFO Count bits and Tx FIFO Full bit in UFSTAT register instead of this bit.	1
Receive buffer data ready	[0]	This bit is automatically set to 1 if receive buffer register contains valid data, received over the RXDn port. 0 = Buffer register is empty 1 = Buffer register has a received data (In Non-FIFO mode, Interrupt or DMA is requested) If UART uses the FIFO, check Rx FIFO Count bits and Rx FIFO Full bit in UFSTAT register instead of this bit.	0

送信/受信ステータスレジスタ

UTRSTAT0 送信/受信ステータスレジスタを読み取り、Receive buffer data ready= 1 の場合はデータを受信で、URXH0 レジスタを読み込まれば、 8bit のデーだを受け取ります；Transmitter empty = 1 の場合はデータを送信でき、そして UTXH0 に 8bit のデータを書き込みます。

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 12.uart_putchar
# make
```

12.uart_putchar のディレクトリ下に uart.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

シリアルケーブルを接続、ポートを開きます、PC で任意文字を入力し、シリアル端末では ASCII コード表の次の文字を表示します、例えば入力文字は｀a｀、シリアル端末では｀b｀が表示します。

本章では UART を初期化して、端末で文字入力は可能になりました、これはコードをデバッグすることに効率が大幅に上がりますが、現在のコードは簡単で、一文字だけ入力/出力できます。次の章では putc()和 getc()の関数をしようして、プログラムに printf()と scanf()機能を加えます。

第 XIII 章　printf や scanf 関数移植

第一節　移植方法

printf や scanf 関数の移植方法は、選択が多い：

1) linux 版の printk 関数移植、バージョンが新しいほど、移植はより困難ですが、機能はより豊富だ；

2）uboot の printf や scanf 機能移植、実際 uboot 関数も linux カーネルから移植するものです；

3）完全に自分を書くこともありますが、機能が比較的弱いです；

全体の OS なしの他のコード部分にとコンパイラは共に問題がないの場合、上記の 3 つの方法ともに実現可能です。

次は linux 版の printk 関数を利用して、OS なしプログラムに移植し、機能を加えます。

第二節 移植手順

ステップ 1　13.uart_stdio ディレクトリに printf.rar をアンパックして、解凍が成功すると include や include ディレクトリが確認できます、include は対応のヘッドファイル、lib では printf や scanf のコードを置きます。

ステップ 2　13.uart_stdio ディレクトリ下の makefile を修正し、lib ディレクトリ下のデータを lib.a にコンパイラリンクした後、lib.a を bin にコンパイルします、ソースコードご参照ください。

ステップ 3　main 関数をコンパイル、テストします。

第三節 プログラム説明

完全なコードは、ディレクトリ 13.uart_stdio ご参照ください。前章と比べったら、BL1 ディレクトリでは変更がありません、BL2 ディレクトリでは include や lib ディレクトリを追加致し、main.c も修正します。

1. /lib/printf.c

<1> printf 定義：

```
int printf(const char *fmt、 ...)
{
    int i;
    int len;
    va_list args;              // va_list 即 char *
    va_start(args、 fmt);
    len = vsprintf(g_PCOutBuf、fmt、args);  // 内部は va_arg()を使用します
    va_end(args);
    for (i = 0; i < strlen(g_PCOutBuf); i++)
    {
    putc(g_PCOutBuf[i]);
```

```
    }
    return len;
  }
```

<2> printf 関数は可変引数の関数であり、〝可変引数の関数〟とは：

可変引数の関数のプロトタイプ宣言は type VAFunction(type arg1、 type arg2、 …); 引数一は二部分に分けます：数が固定の固定引数と数が可変の可選引数。関数に最小一つの固定引数が必要で、固定引数の宣言は普通関数のと同じ；可選引数は数が確定しないため、宣言時は"..."で表示します。固定引数と可選引数は共同で引数リストを構成しています。

<3> printf 関数では三つの重要なマクロに関わります：

マクロ 1: #define va_start(ap、 A) (void) ((ap) = (((char *) &(A)) + (_bnd (A、 _AUPBND))))

マクロ 2: #define va_arg(ap、 T) (*(T *)(((ap) += (_bnd (T、 _AUPBND))) - (_bnd(T、 _ADNBND))))

マクロ 3: #define va_end(ap) (void) 0

これらのマクロ中、va は variable argument(可変引数)です;

ap: 可変個引数リストのポインタ；

A：可変個引数リストの固定引数；

T：可変個引数のタイプ。

va_list も一つのマクロで、、定義は typedef char * va_list、実際は char 型ポインタ。

<4> 三つのマクロの機能：

1) va_start マクロ

- 機能:

V により、変数引数リストのポインタをフェッチし、値を AP に与えます。方法：最後の固定引数Aのアドレス＋ 最初の可変引数とAにとってのオフセット･アドレス、その値を ap に与えて、 ap は変数引数リストの最初のアドレス先頭アドレスになります。。

- 例を挙げます：

可変引数の宣言は void va_test(char a、 char b、char c、 …)の場合、その固定引数は a、b、c、最後の固定引数は c で、すなわち va_start(ap、 c)。

2) va_arg マクロ

- 機能:

現在 ap ポインタの可変引数を取り出して、a p ポインタを可変引数に次の指します。

3) va_end マクロ

- 機能:

可変引数の受け取りを終了します。va_end (list)は空きと定義して、実際対応コードはありません。コード対称性のために、va_start と対応します。

<5> 可変引数の数を得るには三つの方法があります：

1) 関数の第一個の引数が次の引数個数を定義します、例え、 func(int num、...)；

2) 隠し引数によって、引数の個数を判断します、例え printf シリーズ、文字列での%の数で判定します；

3) 特別な状況（パラメータが 0xFFFF より小さいの int）な場合、常にリターンアドレスまで、より低いスタックにアクセスすることができます。

上記の知識で printf()関数は理解できます。まず va_start マクロ（args、 fmt）；は可変引数の先頭アドレスを args に保存され、そして vsprintf(g_PCOutBuf、fmt、args)を呼び出し処理します、続いて vsprintf()で va_arg()を使用し、可変引数を取り出して、解析します。普通の文字な場合は変換せず、直接 g_PCOutBuf 保存します；文字列の場合、変数引数リストで文字列へのポインタを取得し、文字列の内容を g_PCOutBuf にコピーします；数字である場合は、 number 関数で処理します、そして解析結果 g_PCOutBuf に保存します。

最後に、PUTC 関数で g_PCOutBuf 内の文字をプリントアウトします。scanf 関数の原理は printf 類似して、特別説明しません。

2. main.c

完全コード：

```
int main()
{
    int a = 0;
    int b = 0;
    char *str = "hello world";

    uart_init();
    printf("%s¥n"、str);
    while (1)
    {
        printf("please enter two number: ¥r¥n");
        scanf("%d %d"、 &a、 &b);
        printf("¥r¥n");
        printf("the sum is: %d¥r¥n"、 a+b);
    }
    return 0;
}
```

まず “hello world”をプリントアウトして、シリアルポートから２つの数字を受け取り、最後に、それらの和を出力する。

第四節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

cd 13.uart_stdio

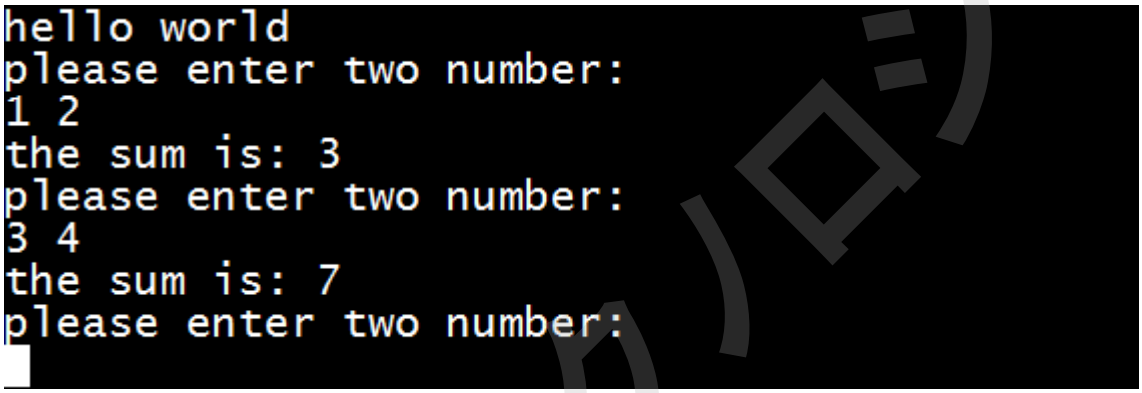
make

13.uart_stdio のディレクトリ下に stdio.bin を生成し、それを開発ボードにプログラムします。

第五節　実験現象

シリアルケーブルを接続、secure CRT やスーパー端末などのポート工具でポートを開きます、そして端末で“hello world”をプリントアウトされます、２つの数字を入力と提示します、入力すると、その和をプリントアウトします。

結果は下記の通りです：



第 XIV 章　NAND Flash の読み取り・書き込み・消去

第一節　NAND Flash について

S5PV210 の NAND Flash コントロールは下記の特徴があります：

1) 512byte、2k、4k、8k ページをサーポート

2) 各種ソフトを通じて、 NAND Flash 読み取り・書き込み・消去機能を実現します

3) 8bit のバス

4) SLC 、 MCL　NAND Flash をサポート

5) 1/4/8/12/16bit の ECC をサポート

6) レジスタバイト/ハーフワード/ワード単位でデータ/ ECC レジスタバイト/ハーフワード/ワードをアクセスや他のレジスタユニットにアクセスすることをサポートします。

注：ここで使用したのは Mini210S の NAND Flash： タイプ SLC、容量 1G、番号 K9K8G08U0A。本章の内容は SLC　NAND Flash に対するもので(256M/512M/1GB など)、 MLC　NANDFlash では適用しません。

第二節　プログラム説明

完全なコードは、ディレクトリ 14.nand ご参照ください。前章と比べったら、nand.c ファイルが追加し、

NAND Flash に対しての操作が加えます。

1. nand.c

<1> NAND Flash 初期化関数 nand_init()、コードは下記の通りです:

```
void nand_init(void)
{
    // 1. NAND Flash コンフィグレーション
    NFCONF
    = (TACLS<<12)|(TWRPH0<<8)|(TWRPH1<<4)|(0<<3)|(0<<2)|(1<<1)|(0<<0);
    NFCONT
    =(0<<18)|(0<<17)|(0<<16)|(0<<10)|(0<<9)|(0<<8)|(0<<7)|(0<<6)|(0x3<<1)|(1<<0);
    // 2. ピン・コンフィグレーション
    MP0_1CON = 0x22333322;
    MP0_2CON = 0x00002222;
    MP0_3CON = 0x22222222;
    // 3. リセット
    nand_reset();
}
```

三つの手順があります：

ステップ 1　NAND Flash コンフィグレーション

NFCONF と NFCONT ２つのレジスタの NAND Flash コンフィグレーション

4.5.2.1 Nand Flash Configuration Register (NFCONF, R/W, Address = 0xB0E0_0000)

NFCONF	Bit	Description	Initial State
Reserved	[31:26]	Reserved	0
MsgLength	[25]	0 = 512 byte Message Length 1 = 24 byte Message Length	0
ECCType0	[24:23]	This bit indicates the kind of ECC to use. 00 = 1-bit ECC 10 = 4-bit ECC 01 = 11 = Disable 1-bit and 4-bit ECC	0
Reserved	[22:16]	Reserved	0000000
TACLS	[15:12]	CLE and ALE duration setting value (0~15) Duration = HCLK x TACLS	0x1
TWRPH0	[11:8]	TWRPH0 duration setting value (0~15) Duration = HCLK x (TWRPH0 + 1) Note: You should add additional cycles about 10ns for page read because of additional signal delay on PCB pattern.	0x0
TWRPH1	[7:4]	TWRPH1 duration setting value (0~15) Duration = HCLK x (TWRPH1 + 1)	0x0
MLCFlash	[3]	This bit indicates the kind of NAND Flash memory to use. 0 = SLC NAND Flash 1 = MLC NAND Flash	0
PageSize	[2]	This bit indicates the page size of NAND Flash Memory, When MLCFlash is 0, the value of PageSize is as follows: 0 = 2048 Bytes/page 1 = 512 Bytes/page When MLCFlash is 1, the value of PageSize is as follows: 0 = 4096 Bytes/page 1 = 2048 Bytes/page	0
AddrCycle	[1]	This bit indicates the number of Address cycle of NAND Flash memory. When Page Size is 512 Bytes, 0 = 3 address cycle 1 = 4 address cycle When page size is 2K or 4K, 0 = 4 address cycle 1 = 5 address cycle	0
Reserved	[0]	Reserved	0

NFCONF レジスタ

- AddrCycle = 1、When page size is 2K or 4K、 1 = 5 address cycle、Mini210S の NAND Flash のページは 2k、アドレス･サイクルは 5 つで；
- PageSize = 0、When MLCFlash is 0、 the value of PageSize is as follows: 0 = 2048 Bytes/page、Mini210S は SLC NAND Flash を使用します、そしてページは 2k；
- MLCFlash = 0、ここでは SLC NAND Flash を使用します；
- TWRPH1/TWRPH0/TACLS はアクセスタイミングの設定、NAND Flash チップマニュアルをご参照ください値は下記のと同じです。 TWRPH1=1、TWRPH0=4、TACLS=1；
- ECCType0/MsgLength、ベアメタル･コードでは ECC を使わないため、ここではグラフを設定しません。

4.5.2.2 Control Register (NFCONT, R/W, Address = 0xB0E0_0004)

NFCONT	Bit	Description	Initial State
Reserved	[31:24]	Reserved	0
Reg_nCE3	[23]	NAND Flash Memory nRCS[3] signal control 0 = Force nRCS[3] to low (Enable chip select) 1 = Force nRCS[3] to High (Disable chip select)	1
Reg_nCE2	[22]	NAND Flash Memory nRCS[2] signal control 0 = Force nRCS[2] to low (Enable chip select) 1 = Force nRCS[2] to High (Disable chip select)	1
Reserved	[21:19]	Reserved	0
MLCEccDirection	[18]	4-bit, ECC encoding / decoding control 0 = Decoding 4-bit ECC, It is used for page read 1 = Encoding 4-bit ECC, It is be used for page program	0
LockTight	[17]	Lock-tight configuration 0 = Disable lock-tight 1 = Enable lock-tight, If this bit is set to 1, you cannot clear this bit. For more information, refer to the *4.3.12 "Lock scheme for data protection"*.	0
LOCK	[16]	Soft Lock configuration 0 = Disable lock 1 = Enable lock Software can modify soft lock area any time. For more information, refer to the *4.3.12 "*.	1
Reserved	[15:14]	Reserved	00
EnbMLCEncInt	[13]	4-bit ECC encoding completion interrupt control 0 = Disable interrupt 1 = Enable interrupt	0
EnbMLCDecInt	[12]	4-bit ECC decoding completion interrupt control 0 = Disable interrupt 1 = Enable interrupt	0
	[11]	Reserved	0
EnbIllegalAccINT	[10]	Illegal access interrupt control 0 = Disable interrupt 1 = Enable interrupt Illegal access interrupt occurs when CPU tries to program or erase locking area (the area setting in NFSBLK (0xB0E0_0020) to NFEBLK (0xB0E0_0024)-1.	0
EnbRnBINT	[9]	RnB status input signal transition interrupt control 0 = Disable RnB interrupt 1 = Enable RnB interrupt	0
RnB_TransMode	[8]	RnB transition detection configuration 0 = Detect rising edge 1 = Detect falling edge	0

NFCONT	Bit	Description	Initial State
MECCLock	[7]	Lock Main area ECC generation 0 = Unlock Main area ECC 1 = Lock Main area ECC Main area ECC status register is NFMECC0/NFMECC1(0xB0E0_0034/0xB0E0_0038),	1
SECCLock	[6]	Lock Spare area ECC generation. 0 = Unlock Spare ECC 1 = Lock Spare ECC Spare area ECC status register is NFSECC(0xB0E0_003C),	1
InitMECC	[5]	1 = Initialize main area ECC decoder/encoder (write-only)	0
InitSECC	[4]	1 = Initialize spare area ECC decoder/encoder (write-only)	0
HW_nCE	[3]	Reserved (HW_nCE)	0
Reg_nCE1	[2]	NAND Flash Memory nRCS[1] signal control	1
Reg_nCE0	[1]	NAND Flash Memory nRCS[0] signal control 0 = Force nRCS[0] to low (Enable chip select) 1 = Force nRCS[0] to High (Disable chip select) Note: The setting all nCE[3:0] zero can not be allowed. Only one nCE can be asserted to enable external NAND flash memory. The lower bit has more priority when user set all nCE[3:0] zeros.	1
MODE	[0]	NAND Flash controller operating mode 0 = Disable NAND Flash Controller 1 = Enable NAND Flash Controller	0

NFCONT レジスタ

- MODE = 1、 NAND Flash コントロールをオンにする；
- Reg_nCE0 = 1、チップセレクトを中止し、 NAND Flash 操作時にまたチップセレクトする
- Reg_nCE1 = 1、チップセレクトを中止し、 NAND Flash 操作時にまたチップセレクトする
- InitMECC/InitSECC/SECCLock/MECCLock、ベアメタル· コードでは ECC を使わないため、ここでは 4 つのグラフを任意設定します；
- RnB_TransMode = 0、Detect rising edge、RnB は NAND Flash ステータス・プローブ・ピンで、立ち上がりエッジでトリガします；
- EnbRnBINT = 0 、RnB 割り込みを禁止；
- EnbIllegalAccINT = 0、Illegal access 割り込みを禁止；
- EnbMLCDecInt/EnbMLCEncInt は MCL 関連、設定する必要はありません；
- LOCK = 0、Soft Lock を使わないため、 Soft Lock を禁止します；
- LockTight = 0、Lock-tight を使わないため、すべての Lock-tight を禁止します；
- MLCEccDirection、MLC 関連、設定する必要はありません；

ステップ 2 ピン・コンフィグレーション

NAND Flash 関連機能；

ステップ 3　リセット

リセット関数 nand_reset 関連コード：

```
static void nand_reset(void)
{
    nand_select_chip();
    nand_send_cmd(NAND_CMD_RES);
    nand_wait_idle();
    nand_deselect_chip();
}
```

NAND Flash のリセット操作は 4 つの手順があります：

1) チップセレクト送信、実際は NFCONT &= ~(1<<1); NFCONT の bit[1]に 0 を書き込みます；

2) リセット・コマンド NAND_CMD_RES (0xff)を送信;実際は NFCMMD = cmd; NFCMMD レジスタにコマンドを書き込みます；

完全な NAND Flash コマンドは下記図ご参照ください：

Table 1. Command Sets

Function	1st Cycle	2nd Cycle	Acceptable Command during Busy
Read	00h	30h	
Read for Copy Back	00h	35h	
Read ID	90h	-	
Reset	FFh	-	O
Page Program	80h	10h	
Two-Plane Page Program(4)	80h---11h	81h---10h	
Copy-Back Program	85h	10h	
Two-Plane Copy-Back Program(4)	85h---11h	81h---10h	
Block Erase	60h	D0h	
Two-Plane Block Erase	60h---60h	D0h	
Random Data Input(1)	85h	-	
Random Data Output(1)	05h	E0h	
Read Status	70h		O
Read EDC Status(2)	7Bh		O
Chip1 Status(3)	F1h		O
Chip2 Status(3)	F2h		O

3) NAND Flash 待ち ;実際は while(!(NFSTAT & (BUSY<<4)))、 NFSTAT の bit[4]を読み取り、NAND Flash の状態を確認します；

4) チップセレクトを中止、実際は NFCONT |= (1<<1); NFCONT の bit[1]に 1 を書き込みます；

<2> NAND Flash で ID 関数 nand_read_id()を呼び出します、コードは：

```
void nand_read_id(void)
{
```

```
    nand_id_info nand_id;
    // 1. チップセレクト送信
    nand_select_chip();
    // 2. ID 読み取り
    nand_send_cmd(NAND_CMD_READ_ID);
    nand_send_addr(0x00);
    nand_wait_idle();
    nand_id.IDm =                nand_read();
    nand_id.IDd =                nand_read();
    nand_id.ID3rd = nand_read();
    nand_id.ID4th = nand_read();
    nand_id.ID5th = nand_read();

    printf("NANDFlash: makercode = %x、devicecode = %x\r\n"、nand_id.IDm、nand_id.IDd);
    nand_deselect_chip();
}
```

Device	Device Code(2nd Cycle)	3rd Cycle	4th Cycle	5th Cycle
K9K8G08U0A	D3h	51h	95h	58h
K9WAG08U1A K9NBG08U5A	Same as K9K8G08U0A in it			

NAND Flash ID 読み取り

上記図のように、NAND Flash の ID 読み取り操作は５つの手順があります：

ステップ 1　チップセレクト送信；

ステップ 2　ID 読み取りコマンド送信、コマンド NAND_CMD_READ_ID(0x90)；

ステップ 3　アドレス 0x00 を送信；関数 nand_send_addr()を呼び出し；

ステップ 4　NAND Flash レディ待つ；

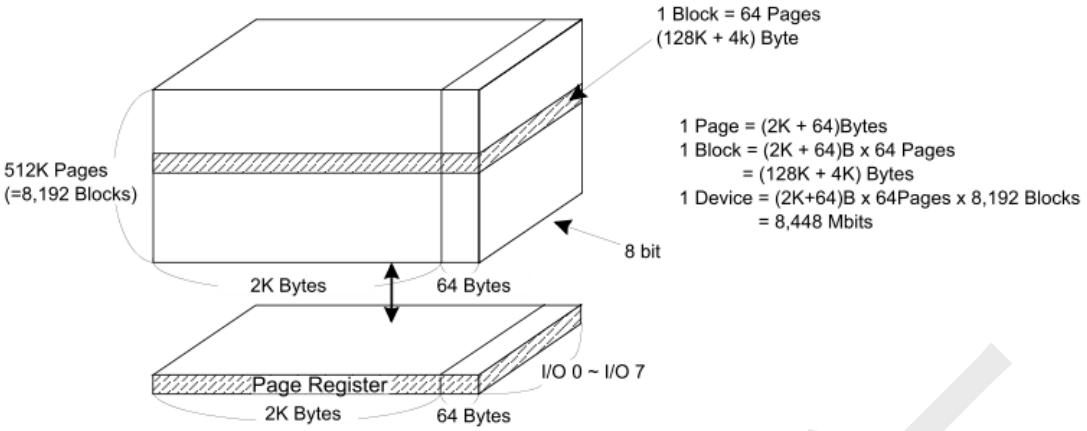
ステップ 5　ID 読み取りには、nand_read()関数を使用します、実際は NFDATA レジスタを読み取ります；

次は関数 nand_send_addr()を説明します、コアコードは：

```
{
    //カラムアドレス、すなわち　ページ内アドレス
    col = addr % NAND_PAGE_SIZE;
    //行アドレス、すなわちページアドレス
    row = addr / NAND_PAGE_SIZE;
```

```
    // Column Address A0~A7
    NFADDR = col & 0xff;
    for(i=0; i<10; i++);

    // Column Address A8~A11

    NFADDR = (col >> 8) & 0x0f;
    for(i=0; i<10; i++);

    // Row Address A12~A19
    NFADDR = row & 0xff;
    for(i=0; i<10; i++);

    // Row Address A20~A27
    NFADDR = (row >> 8) & 0xff;
    for(i=0; i<10; i++);

    // Row Address A28~A30
    NFADDR = (row >> 16) & 0xff;
    for(i=0; i<10; i++);
}
```

まず、ページの大きさに応じてページ· アドレスとページオフセット· アドレスを取得して、そして 5 つのサイクルを介してアドレスを送信します。実際は NFADDR レジスタを書き込み、各サイクルの送信方法は NAND Flash のチップマニュアルをご参照ください。詳細は下記図のように：:



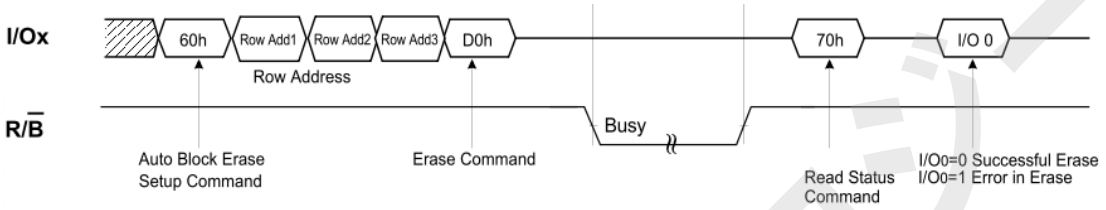
	I/O 0	I/O 1	I/O 2	I/O 3	I/O 4	I/O 5	I/O 6	I/O 7	
1st Cycle	A0	A1	A2	A3	A4	A5	A6	A7	Column Address
2nd Cycle	A8	A9	A10	A11	*L	*L	*L	*L	Column Address
3rd Cycle	A12	A13	A14	A15	A16	A17	A18	A19	Row Address
4th Cycle	A20	A21	A22	A23	A24	A25	A26	A27	Row Address
5th Cycle	A28	A29	A30	*L	*L	*L	*L	*L	Row Address

アドレスが送信された後、ID 5 を読み出すことができます、一番目は MAKDER CODE、2 番目は DEVICE CODE です。

<3> NAND Flash 消去関数 nand_erase()、コアコードは：

```
{
    //  row アドレス、ページのアドレス取得
    unsigned long row = block_num * NAND_BLOCK_SIZE;

    // 1. チップセレクトシグナル送信
    nand_select_chip();
    // 2. 消去：第 1 サイクルコマンド 0x60 を送信、第 2 サイクルは· ブロック· アドレスを送信、第 3 サイクルはコマンド 0xd0nand_send_cmd(NAND_CMD_BLOCK_ERASE_1st)を送信;
    for(i=0; i<10; i++);
    // Row Address A12~A19
    NFADDR = row & 0xff;
    for(i=0; i<10; i++);
    // Row Address A20~A27
    NFADDR = (row >> 8) & 0xff;
    for(i=0; i<10; i++);
    // Row Address A28~A30
    NFADDR = (row >> 16) & 0xff;
```

```
        NFSTAT = (NFSTAT)|(1<<4);
        nand_send_cmd(NAND_CMD_BLOCK_ERASE_2st);
        for(i=0; i<10; i++);
        // 3. 準備完了を待ち
        nand_wait_idle();

        // 4. ステータスを読み取り
        unsigned char status = read_nand_status();
}
```



上記図によって、NAND Flash の消去操作は 6 つの手順があります：

ステップ 1　チップセレクト送信；

ステップ 2　消去コマンド 1 NAND_CMD_BLOCK_ERASE_1(0x60) 送信；

ステップ 3　ページアドレスを送信；

ステップ 4　消去コマンド 2 NAND_CMD_BLOCK_ERASE_2st(0xD0) 送信；

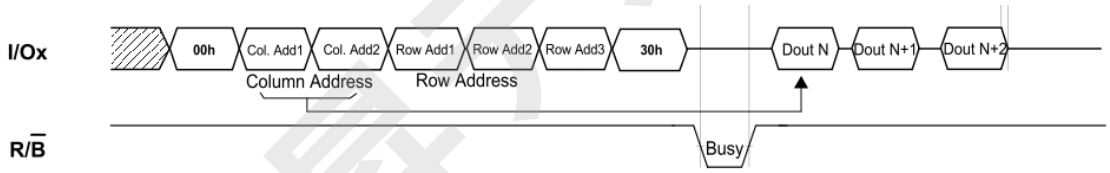
ステップ 5　NAND Flash 準備完了を待ち；

ステップ 6　ステータスを読み取り、消去状態を確認します。消去に失敗した場合、不良ブロックをプリントアウトして、チップセレクトをキャンセルします、それ以外の場合は直接チップセレクトをキャンセルできます。 ステータス読み取りには read_nand_status（）を使用します、実際は nand_send_cmd(NAND_CMD_READ_STATUS)；ch=nand_read();ステータス状態読み取り、NAND_CMD_READ_STATUS、続いて値を読み取ります。

<4> NAND Flash 関数 copy_nand_to_sdram()、 NAND Flash からデータを DRAM に読み取ります、コアコードは：

```
{
        // 1. チップセレクト送信
        nand_select_chip();
        // 2. Nand から sdram にデータを読み取り、第 1 サイクルコマンド 0x00 を送信、第 2 サイクルは･アドレス nand_addr を送信、第 3 サイクルはコマンド 0x30 を送信、1 ページ(2K)のデータを読み取ります
        while(length)
        {
            nand_send_cmd(NAND_CMD_READ_1st);
```

```
        nand_send_addr(nand_addr);
        NFSTAT = (NFSTAT)|(1<<4);
        nand_send_cmd(NAND_CMD_READ_2st);
        nand_wait_idle();
        // カラムアドレス、ページ内アドレス
        unsigned long col = nand_addr % NAND_PAGE_SIZE;
        i = col;
        // 1 ページのデータを読み取り、毎回は 1byte コピーして、データの長さ　length が完了するまで合計 2048 回の（2K）コピーします。
        for(; i<NAND_PAGE_SIZE && length!=0; i++、length--)
        {
            *sdram_addr = nand_read();
            sdram_addr++;
            nand_addr++;
        }
    }

    // 3. ステータス読み取り
    unsigned char status = read_nand_status();
}
```



NAND Flash 読み取り操作

上記図によって、NAND Flash の読み取り操作は 7 つの手順があります：

ステップ 1　チップセレクト送信；

ステップ 2　読み取りコマンド 1 NAND_CMD_READ_1st(0x00)を送信；

ステップ 3　アドレス読み取り関数 nand_send_cmd()を呼び出し、5 つのアドレス・サイクルを送信します；

ステップ 4　読み取りコマンド 2 NAND_CMD_READ_2st(0xD0) を送信；

ステップ 5　NAND Flash 準備完了を待ち；

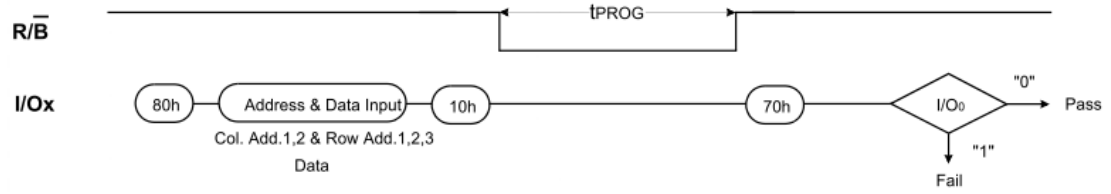
ステップ 6　ページ内のオフセットから読み始め、ページの最後を読んで、毎回 1byte を読み取ります；

ステップ 7 ステータスを読み取り、成功するかどうかを判決します。

<5> NAND Flash 書き込み関数 copy_sdram_to_nand ()、DRAM から NAND Flash にデータを書き込みます、コアコードは下記の通りです：

```
{
    // 1. チップセレクトシグナル送信
    nand_select_chip();
    // 2.  Sdram から  nand にデータを読み取り、第 1 サイクルコマンド 0x80 を送信、第 2 サイクルはアドレス nand_addr を送信、第 3 サイクルは 1 ページ(2K)のデータを書き込み、第 4 サイクルはコマンド 0x10 を送信；
    while(length)
    {
        nand_send_cmd(NAND_CMD_WRITE_PAGE_1st);
        nand_send_addr(nand_addr);
        //カラムアドレス、ページ内アドレス
        unsigned long col = nand_addr % NAND_PAGE_SIZE;
        i = col;
        //  1 ページのデータを書き込み、毎回は 1byte コピーして、データの長さ length が完了するまで合計 2048 回の（2K）コピーします。
        for(; i<NAND_PAGE_SIZE && length!=0; i++、length--)
        {
            nand_write(*sdram_addr);
            sdram_addr++;
            nand_addr++;
        }
        NFSTAT = (NFSTAT)|(1<<4);
        nand_send_cmd(NAND_CMD_WRITE_PAGE_2st);
        nand_wait_idle();
    }
    // 3. ステータス読み取り
    unsigned char status = read_nand_status();
```

Figure 8. Program & Read Status Operation



上記図によって、NAND Flash の書き込み操作は 7 つの手順があります：

ステップ 1　チップセレクト送信；

ステップ 2　書き込みコマンド 1 NAND_CMD_WRITE_PAGE_1st (0x80)を送信；

ステップ 3　アドレスを送信、関数 nand_send_cmd()を呼び出し、 5 つのアドレス・サイクルを送信します；

ステップ 4　読み取りコマンド 2 NAND_CMD_WRITE_PAGE_2st (0x10) を送信；

ステップ 5　NAND Flash 準備完了を待ち；

ステップ 6　 ページ内のオフセットから書き込み、ページの最後まで書き込み、毎回 1byte を書き込みます；

ステップ 7 ステータスを読み取り、成功するかどうかを判決します。

2. main.c

main.c 中、まず関数 nand_init()を呼び出し NAND Flash を初期化します、続いてメニューをプリントアウトして、4 種のオプションがあります：

NAND Flash テスト：

ID 読み取り機能(nand_read_id())；

消去機能(nand_erase())；

読み取り機能(copy_nand_to_sdram())；

書き込み機能(copy_sdram_to_nand())；

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 14.nand
# make
```

14.nand のディレクトリ下に nand.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

先ずは数字 1、2、3、4...、をプリントアウトし続き、 KEY1 を押すと外部割り込み EINT16_31 は発生し、IRQ_handler にジャンプします。最後に irq_handler()と割り込みハンドラ関数 isr_key()を呼び出します。当関数は先ず゜we get company:EINT16_31゜をプリントアウトします、そして割り込みをクリアします。テスト結果は次のとおりです：

```
***************Int test ***************
0
1
2
3
4
5
6
7
we get company:EINT16_31
8
9
10
we get company:EINT16_31
11
12
13
```

第 XV 章　PWM タイマー

第一節　S5PV210 的 PWM タイマー

S5PV210 には合計 5 つの 32 ビット PWM タイマーがあります、タイマー0,1,2,3 は PWM 機能があります、タイマ 4 は出力ピンがありません。PWM タイマは PCLK_PSYS をクロックソースとして使用します、相関知識は第 VII 章クロック初期化を参照ください、相関構造は下記図ご参照ください：

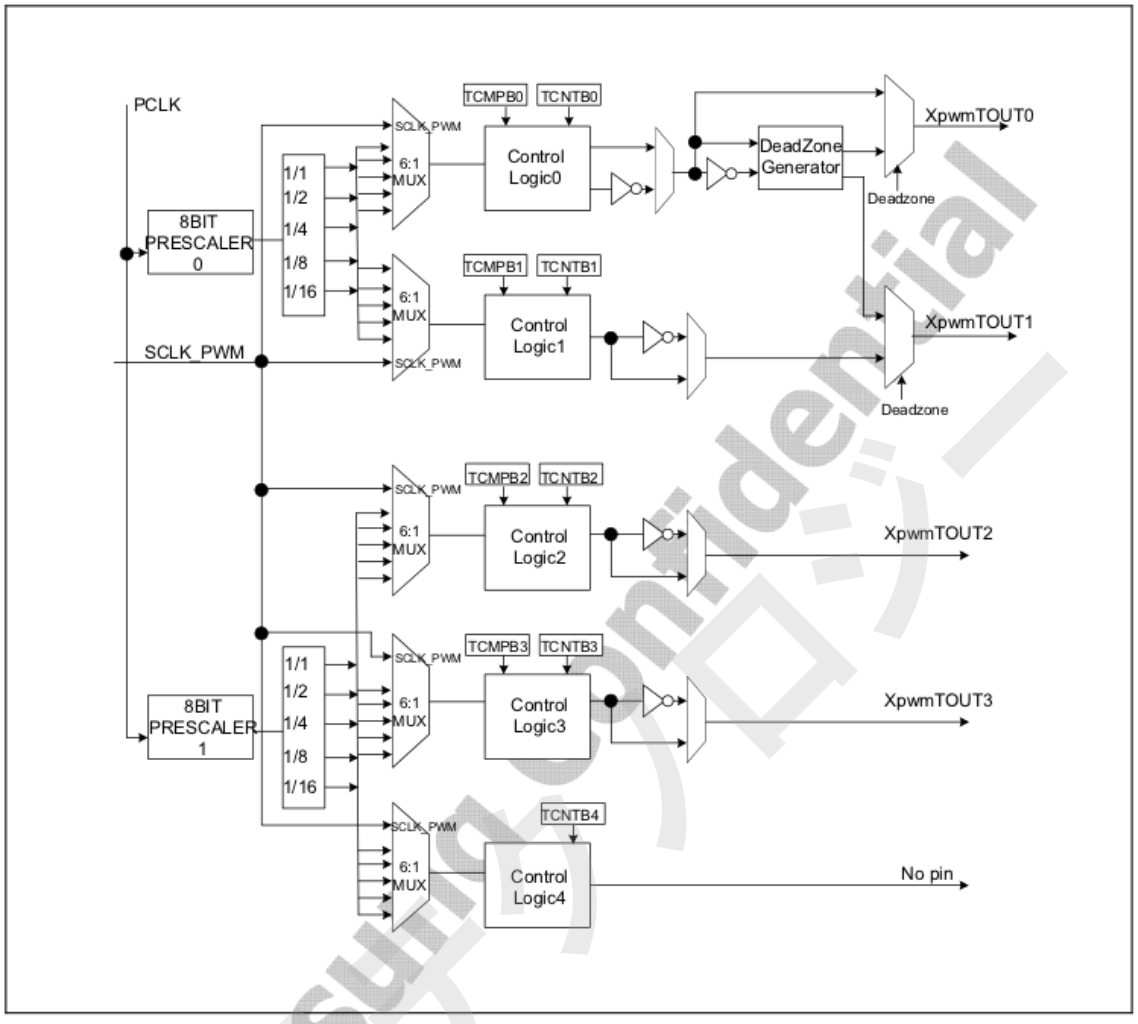


Figure 1-2　PWM TIMER Clock Tree Diagram

第二節　プログラム説明

完全なコードは、ディレクトリ 16.timer ご参照ください。

1. main.c

コアコード：

```
int main(void)
{
    // シリアルポート初期化
    uart_init();
    // 割り込み初期化
    system_initexception();
    // timer 設置
    timer_request();
    while(1);
```

}

手順は 4 つあります、中に ステップ 3、はタイマー機能で

ステップ 1　シリアルポート初期化；

ステップ 2　割り込み初期化；

ステップ 3　timer 設置、関数 timer_request()は timer.c 中に定義されます；

ステップ 4　無限ループタイマー割り込みが発生するまで待ちます。

2. timer.c

```
void timer_request(void)
{
    printf("\r\n##############Timer test#############\r\n");
    // timer を全部禁止する
    pwm_stopall();
    // timer0 割り込みの割り込みハンドラ関数を設置する
    intc_setvectaddr(NUM_TIMER0,irs_timer);
    //   timer0 割り込みを有効にする
    intc_enable(NUM_TIMER0);
    //   timer0 を設置する
    timer_init(0,65,4,62500,0);
}
```

手順は 3 つあります：

ステップ 1 timer を全部禁止する、レジスタ TCON に 0 を書き込み；

ステップ 2 VIC を設置し、先ず timer0 割り込みの割り込みハンドラ関数 irs_timer()を設置し、timer0 割り込みを有効にする；

ステップ 3 timer0 を設置し、関数 timer_init()を呼び出します；

コア機能は：

1) 分周を設置

まず、分周係数を設定する、相関レジスタは TCFG0：

TCFG0	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Dead zone length	[23:16]	Dead zone length	0x00
Prescaler 1	[15:8]	Prescaler 1 value for Timer 2, 3 and 4	0x01
Prescaler 0	[7:0]	Prescaler 0 value for timer 0 and 1	0x01

次に 16 分周に設定する、相関レジスタは TCFG1：

1.5.1.2 Timer Configuration Register (TCFG1, R/W, Address = 0xE250_0004)

TCFG1	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Divider MUX4	[19:16]	Selects Mux input for PWM Timer 4 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX3	[15:12]	Selects Mux input for PWM Timer 3 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX2	[11:8]	Selects Mux input for PWM Timer 2 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX1	[7:4]	Selects Mux input for PWM Timer 1 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX0	[3:0]	Selects Mux input for PWM Timer 0 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00

上記の設定が完了したら、タイマでクロックを入力できます：

Timer Input Clock Frequency = PCLK / ({prescaler value + 1}) / {divider value} =66MHz/(65+1)/16=62500hz

2) カウントを設定

レジスタ TCNTB0=62500 と TCMPB0=0 を設置します、 timer0 起動後、TCNTB0 徐々に-1 で変化し、TCMPB0 になる場合は割り込みを発生します、すなわち 1 秒で timer0 が 1 回割り込みします。

1.5.1.4 Timer0 Counter Register (TCNTB0, R/W, Address = 0xE250_000C)

TCNTB0	Bit	Description	Initial State
Timer 0 Count Buffer	[31:0]	Timer 0 Count Buffer Register	0x0000_0000

1.5.1.5 Timer0 Compare Register (TCMPB0, R/W, Address = 0xE250_0010)

TCMPB0	Bit	Description	Initial State
Timer 0 Compare Buffer	[31:0]	Timer 0 Compare Buffer Register	0x0000_0000

3) timer0 起動

レジスタ TCON を設置、先ず手動設定ビットを更新し、その後手動アップデートビットをクリアし、自動ロードを使用し、最後に timer0 を起動します。

4) timer0 割り込みを有効にする

レジスタ TINT_CSTAT を設置、 timer0 割り込みを有効にする。

1.5.1.18 Interrupt Control and Status Register (TINT_CSTAT, R/W, Address = 0xE250_0044)

TINT_CSTAT	Bit	Description	Initial State
Reserved	[31:10]	Reserved Bits	0x00000
Timer 4 Interrupt Status	[9]	Timer 4 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 3 Interrupt Status	[8]	Timer 3 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 2 Interrupt Status	[7]	Timer 2 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 1 Interrupt Status	[6]	Timer 1 Interrupt Status Bit. Clears by writing `1' on this bit.	0x0
Timer 0 Interrupt Status	[5]	Timer 0 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 4 interrupt Enable	[4]	Enables Timer 4 Interrupt. 1 = Enabled 0 = Disabled	0x0
Timer 3 interrupt Enable	[3]	Enables Timer 3 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 2 interrupt Enable	[2]	Enables Timer 2 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 1 interrupt Enable	[1]	Enables Timer 1 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 0 interrupt Enable	[0]	Enables Timer 0 Interrupt. 1 = Enables 0 = Disabled	0x0

最後に timer0 割り込みハンドラ関数 irs_timer()を分析し、手順は 3 つあります：

ステップ 1 timer0 割り込みステータスレジスタ TINT_CSTAT をクリア；

ステップ 2 timer0 割り込み回数をプリントアウトし、timer0 発生するとプリントアウトします。

ステップ 3 関数 intc_clearvectaddr()を呼び出し、VIC 割り込みをクリア;

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

cd 16.timer

make

16.timer のディレクトリ下に timer.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

端末は数字 1、2、3、4...,をプリントアウトし続き、周波数は 1 回/秒です：

```
############Timer test############
TimerOIntCounter = 0
TimerOIntCounter = 1
TimerOIntCounter = 2
TimerOIntCounter = 3
TimerOIntCounter = 4
TimerOIntCounter = 5
TimerOIntCounter = 6
TimerOIntCounter = 7
TimerOIntCounter = 8
TimerOIntCounter = 9
TimerOIntCounter = 10
```

第 XVI 章　PWM タイマー

第一節 S5PV210 的 PWM タイマー

S5PV210 には合計 5 つの 32 ビット PWM タイマーがあります、タイマー0、1、2、3 は PWM 機能があります、タイマ 4 は出力ピンがありません。PWM タイマは PCLK_PSYS をクロックソースとして使用します、関連知識は第 VII 章クロック初期化を参照ください、関連構造は下記図ご参照ください：

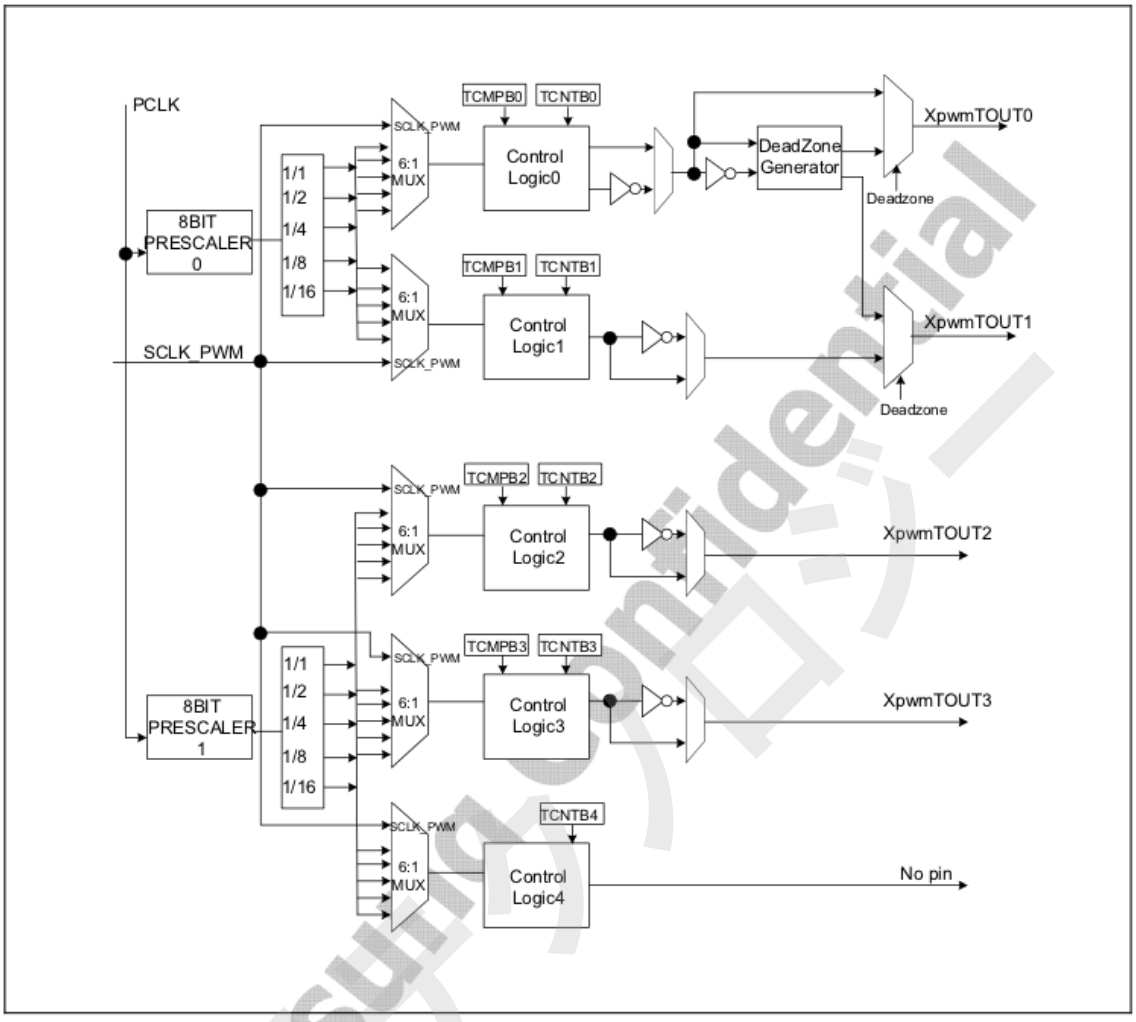


Figure 1-2 PWM TIMER Clock Tree Diagram

第二節 プログラム説明

完全なコードは、ディレクトリ 16.timer ご参照ください。

1. main.c

コアコード：

```
int main(void)
{
    // シリアルポート初期化
    uart_init();
    // 割り込み初期化
    system_initexception();
    // timer 設定
    timer_request();
    while(1);
```

}

手順は 4 つあります、中に ステップ 3、はタイマー機能で

ステップ 1　シリアルポート初期化；

ステップ 2　割り込み初期化；

ステップ 3　timer 設定、関数 timer_request()は timer.c 中に定義されます；

ステップ 4　無限ループタイマー割り込みが発生するまで待ちます。

2. timer.c

```
void timer_request(void)
{
        printf("\r\n##############Timer test############\r\n");
        // timer を全部禁止する
        pwm_stopall();
        // timer0 割り込みの割り込みハンドラ関数を設定する
        intc_setvectaddr(NUM_TIMER0、irs_timer);
        //   timer0 割り込みをオンにする
        intc_enable(NUM_TIMER0);
        //   timer0 を設定する
        timer_init(0、65、4、62500、0);
}
```

手順は 3 つあります：

ステップ 1 timer を全部禁止する、レジスタ TCON に 0 を書き込み；

ステップ 2 VIC を設定し、先ず timer0 割り込みの割り込みハンドラ関数 irs_timer()を設定し、timer0 割り込みをオンにする；

ステップ 3 timer0 を設定し、関数 timer_init()を呼び出します；

コア機能は：

1) 分周を設定

まず、分周係数を設定する、相関レジスタは TCFG0：

TCFG0	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Dead zone length	[23:16]	Dead zone length	0x00
Prescaler 1	[15:8]	Prescaler 1 value for Timer 2, 3 and 4	0x01
Prescaler 0	[7:0]	Prescaler 0 value for timer 0 and 1	0x01

次に 16 分周に設定する、相関レジスタは TCFG1：

1.5.1.2 Timer Configuration Register (TCFG1, R/W, Address = 0xE250_0004)

TCFG1	Bit	Description	Initial State
Reserved	[31:24]	Reserved Bits	0x00
Divider MUX4	[19:16]	Selects Mux input for PWM Timer 4 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX3	[15:12]	Selects Mux input for PWM Timer 3 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX2	[11:8]	Selects Mux input for PWM Timer 2 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX1	[7:4]	Selects Mux input for PWM Timer 1 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00
Divider MUX0	[3:0]	Selects Mux input for PWM Timer 0 0000 = 1/1 0001 = 1/2 0010 = 1/4 0011 = 1/8 0100 = 1/16 0101 = SCLK_PWM	0x00

上記の設定が完了したら、タイマでクロックを入力できます：

Timer Input Clock Frequency = PCLK / ({prescaler value + 1}) / {divider value} =66MHz/(65+1)/16=62500hz

2) カウントを設定

レジスタ TCNTB0=62500 と TCMPB0=0 を設定します、 timer0 起動後、TCNTB0 徐々に-1 で変化し、TCMPB0 になる場合は割り込みを発生します、すなわち 1 秒で timer0 が 1 回割り込みします。

1.5.1.4 Timer0 Counter Register (TCNTB0, R/W, Address = 0xE250_000C)

TCNTB0	Bit	Description	Initial State
Timer 0 Count Buffer	[31:0]	Timer 0 Count Buffer Register	0x0000_0000

1.5.1.5 Timer0 Compare Register (TCMPB0, R/W, Address = 0xE250_0010)

TCMPB0	Bit	Description	Initial State
Timer 0 Compare Buffer	[31:0]	Timer 0 Compare Buffer Register	0x0000_0000

3) timer0 起動

レジスタ TCON を設定、先ず手動設定ビットを更新し、その後手動アップデートビットをクリアし、自動ロードを使用し、最後に timer0 を起動します。

4) timer0 割り込みをオンにする

レジスタ TINT_CSTAT を設定、 timer0 割り込みをオンにする。

1.5.1.18 Interrupt Control and Status Register (TINT_CSTAT, R/W, Address = 0xE250_0044)

TINT_CSTAT	Bit	Description	Initial State
Reserved	[31:10]	Reserved Bits	0x00000
Timer 4 Interrupt Status	[9]	Timer 4 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 3 Interrupt Status	[8]	Timer 3 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 2 Interrupt Status	[7]	Timer 2 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 1 Interrupt Status	[6]	Timer 1 Interrupt Status Bit. Clears by writing `1' on this bit.	0x0
Timer 0 Interrupt Status	[5]	Timer 0 Interrupt Status Bit. Clears by writing '1' on this bit.	0x0
Timer 4 interrupt Enable	[4]	Enables Timer 4 Interrupt. 1 = Enabled 0 = Disabled	0x0
Timer 3 interrupt Enable	[3]	Enables Timer 3 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 2 interrupt Enable	[2]	Enables Timer 2 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 1 interrupt Enable	[1]	Enables Timer 1 Interrupt. 1 = Enables 0 = Disables	0x0
Timer 0 interrupt Enable	[0]	Enables Timer 0 Interrupt. 1 = Enables 0 = Disabled	0x0

最後に timer0 割り込みハンドラ関数 irs_timer()を分析し、手順は 3 つあります：

ステップ 1 timer0 割り込みステータスレジスタ TINT_CSTAT をクリア；

ステップ 2 timer0 割り込み回数をプリントアウトし、timer0 発生するとプリントアウトします。

ステップ 3 関数 intc_clearvectaddr()を呼び出し、VIC 割り込みをクリア;

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

cd 16.timer

make

16.timer のディレクトリ下に timer.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

端末は数字 1、2、3、4...、をプリントアウトし続き、周波数は 1 回/秒です：

```
############Timer test###########
TimerOIntCounter = 0
TimerOIntCounter = 1
TimerOIntCounter = 2
TimerOIntCounter = 3
TimerOIntCounter = 4
TimerOIntCounter = 5
TimerOIntCounter = 6
TimerOIntCounter = 7
TimerOIntCounter = 8
TimerOIntCounter = 9
TimerOIntCounter = 10
```

第 XVII 章　ウォッチドッグタイマとリセット

第一節　S5PV210 ウォッチドッグタイマ

S5PV210 の ウォッチドッグタイマは普通の 16bit タイマと同じで､それは PWM タイマと異なったのはウォッチドッグタイマが reset 発信できます。S5PV210 ウォッチドッグタイマの構造図は下記の通りです：

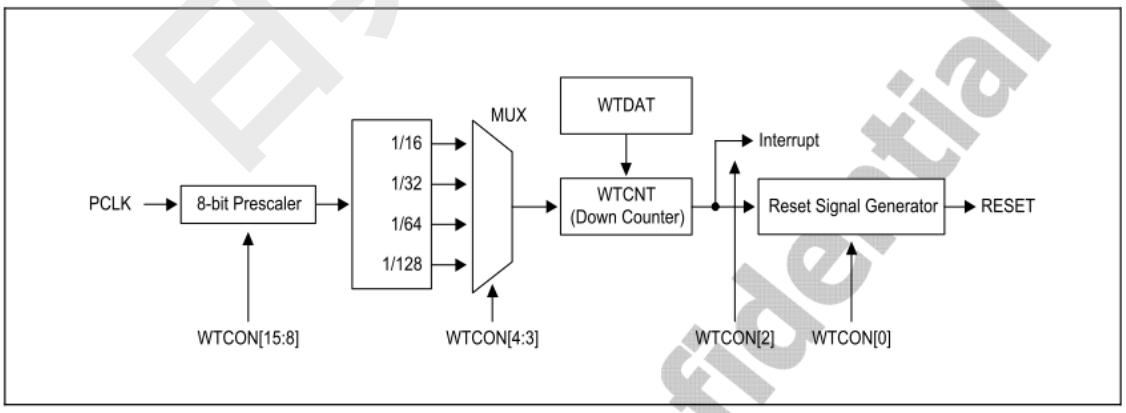


Figure 3-1　Watchdog Timer Block Diagram

第二節 程序相关讲解プログラム説明

完全なコードは、ディレクトリ 17.watchdog ご参照ください。

1. main.c

手順は 4 つあります：

ステップ 1 シリアルポート初期化；

ステップ 2 割り込み初期化；

ステップ 3 ウォッチドッグテスト、関数 wtd_test()を呼び出し、 wtd.c 中に定義されます；

ステップ 4 無限ループ、ウォッチドッグ割り込み発生を待ちます；

2. wtd.c

wtd_test()手順は 4 つあります：

ステップ 1 VIC 割り込み設定、 WTD ハンドラ関数を isr_wtd()と設定し、割り込みをオンにします；

ステップ 2 ウォッチドッグタイマ機能テスト、関数 wtd_operate()を呼び出し、ここではタイマ機能を使用して、 reset 機能は使用しません；

wtd_operate()コード：

```
void wtd_operate(unsigned long uenreset、 unsigned long uenint、 unsigned long uselectclk、
unsigned long uenwtd、 unsigned long uprescaler、unsigned long uwtdat、 unsigned long
uwtcnt)
 {
      WTDAT = uwtdat;
      WTCNT = uwtcnt;

      /*uenreset: reset オン状態判断
      *uenint:  割り込みオン状態判断
      *uselectclk:  分周係数
      *uenwtd:  タイマオン状態判断
      *bit[8:15]:プリスケーラ因子
      */
      WTCON= (uenreset<<0)|(uenint<<2)|(uselectclk<<3)|(uenwtd<<5)|((uprescaler)<<8);
 }
```

先ずカウント相関のレジスタ WTDAT と WTCNT を設定します、レジスタ WTDAT はウォッチドッグタイマのタイムアウト時間を管理します的、ウォッチドッグタイマ起動後、レジスタ WTDAT の値は自動的レジスタ WTCNT に送信します、WTCNT のカウントは 0 になると:割り込みがオンな場合、割り込みします；reset がオンな場合、reset を送信します、WTDAT の値をロードし、リカウントします。

3.4.1.2 Watchdog Timer Data Register (WTDAT, R/W, Address = 0xE270_0004)

The WTDAT register specifies the time-out duration. The content of WTDAT cannot be automatically loaded into the timer counter at initial watchdog timer operation. However, using 0x8000 (initial value) drives the first time-out. In this case, the value of WTDAT is automatically reloaded into WTCNT.

WTDAT	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
Count reload value	[15:0]	Watchdog timer count value for reload.	0x8000

3.4.1.3 Watchdog Timer Count Register (WTCNT, R/W, Address = 0xE270_0008)

The WTCNT register contains the current count values for the watchdog timer during normal operation. Note that the content of the WTDAT register cannot be automatically loaded into the timer count register if the watchdog timer is enabled initially, therefore the WTCNT register must be set to an initial value before enabling it.

WTCNT	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
Count value	[15:0]	The current count value of the watchdog timer	0x8000

レジスタ WTCON を設定し、 reset 機能、割り込み、分周、タイマ機能などを設定します：

WTCON	Bit	Description	Initial State
Reserved	[31:16]	Reserved	0
Prescaler value	[15:8]	Prescaler value. The valid range is from 0 to (2^8-1).	0x80
Reserved	[7:6]	Reserved. These two bits must be 00 in normal operation.	00
Watchdog timer	[5]	Enables or disables Watchdog timer bit. 0 = Disables 1 = Enables	1
Clock select	[4:3]	Determines the clock division factor. 00 = 16 01 = 32 10 = 64 11 = 128	00
Interrupt generation	[2]	Enables or disables interrupt bit. 0 = Disables 1 = Enables	0
Reserved	[1]	Reserved. This bit must be 0 in normal operation.	0
Reset enable/disable	[0]	Enables or disables Watchdog timer output bit for reset signal. 1 = Asserts reset signal of the S5PV210 at watchdog time-out 0 = Disables the reset function of the watchdog timer.	1

ウォッチドッグの割り込みハンドラ関数、コードは：

```
void isr_wtd()
{
    //割り込み回数を記録
    static int wtdcounter=0;
```

```
        printf("%d¥r¥n"、++wtdcounter);

        // ウォッチドッグ割り込みクリア
        WTCLRINT = 1;
        // VIC 割り込みクリア
        intc_clearvectaddr();
        if(wtdcounter==5)
        {
                //  reset  ウォッチドッグ
                printf("waiting system reset¥r¥n");
                wtd_operate(1、1、0、1、100、100000000、100000000);
        }
}
```

手順は 3 つあります：

ステップ 1 割り込み回数をプリントアウト；

ステップ 2 割り込みクリア；

ステップ 3 割り込みが 5 回発生すると、ウォッチドッグの reset 機能を使用、システムリセット；

第三節　コードコンパイルとプログラミングの実行

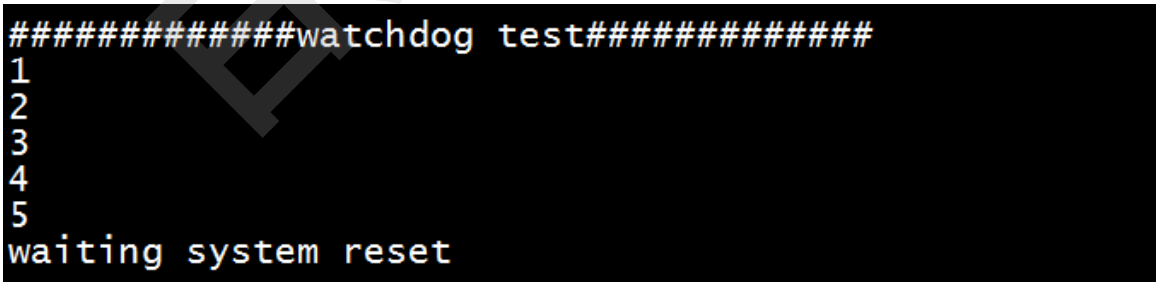
コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 17.watchdog
# make
```

17.watchdog のディレクトリ下に watchdog.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

先ず 1、2、3、4...、をプリントアウトし、5 になると、ウォッチドッグの reset 機能起用、システムが rise、ウォッチドッグのリセット機能が正常に作動します。実際効果は下記図ご参照ください：



第 XVIII 章　RTC 読み取りおよび書き込み時間

第一節　S5PV210 の RTC

RTC リアルタイム· クロック· チップで、システムがパワーオフ状態で、予備リチウムバッテリ時間を記録し続けています。S5PV210 RTC の特性は：

1) BCD 数字サポート；

2) alarm 機能サポート；

3) tick 機能サポート；

4) ミリ秒級 tick time サポート；

S5PV210 RTC 構造図：

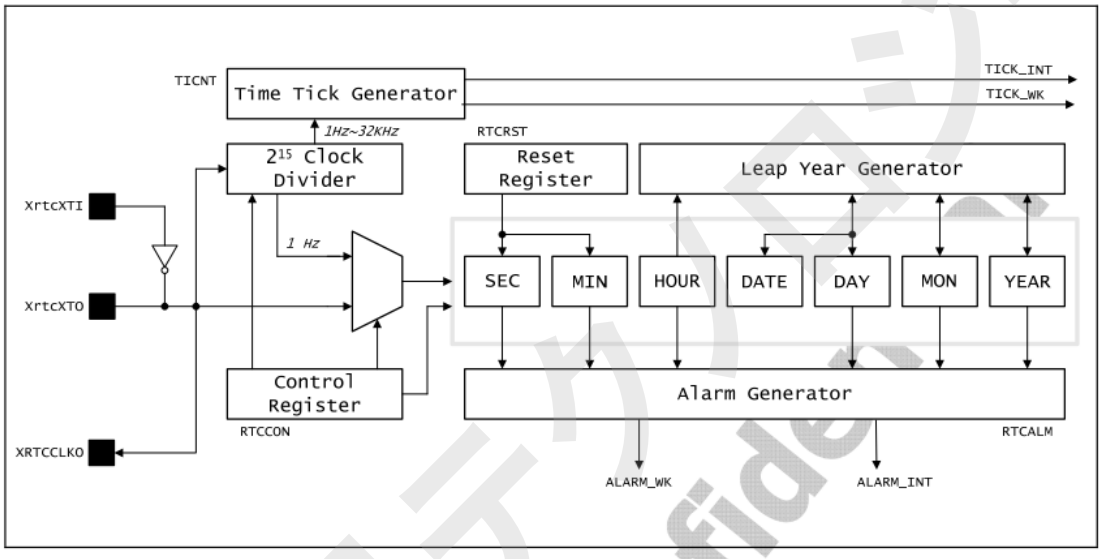


Figure 4-1　Real Time Clock Block Diagram

第二節　プログラム説明

完全なコードは、ディレクトリ 18.rtc ご参照ください。

1. main.c

手順は 3 つあります、中に ステップ 3 は RTC で：

ステップ 1　シリアルポート初期化；

ステップ 2　割り込み初期化；

ステップ 3　メニュをプリントアウトし、２選択があります；

d で関数 rtc_realtime_display()を呼び出し、現在 RTC 時間を表示します；

s で関数 rtc_settime()を呼び出し、 RTC 時間をリセットします；

２つの関数は rtc.c に定義されます；

2. rtc.c

<1> rtc_realtime_display()コード：

```
void rtc_realtime_display(void)
{
    int counter = 0;

    // rtc  コントローラをオンにする
    rtc_enable(true);
    // rtc tick timer をオンにする
    rtc_ticktime_enable(true);

    //  時間を 5 回プリントアウトする
    while( (counter++) < 5)
    {
    rtc_print();
    delay(0x1000000/5);
    }

    // rtc  コントローラをオフにする
    rtc_ticktime_enable(false);
    // rtc tick timer をオフにする
    rtc_enable(false);
}
```

手順は 3 つあります：

ステップ 1　rtc_enable(true)関数で rtc コントローラをオンにします、rtc_ticktime_enable(true)関数で rtc tick timer をオンにします、

2 つの関数はレジスタ RTCCON を設定することです：

RTCCON	Bit	Description	Initial State
Reserved	[31:10]	Reserved	0
CLKOUTEN	[9]	Enables RTC clock output on XRTCCLKO pad. 0 = Disables 1 = Enables	0
TICEN	[8]	Enables Tick timer 0 = Disables 1 = Enables	0
TICCKSEL	[7:4]	Tick timer sub clock selection. 4'b0000 = 32768 Hz 4'b0001 = 16384 Hz 4'b0010 = 8192 Hz 4'b0011 = 4096 Hz 4'b0100 = 2048 Hz 4'b0101 =1024 Hz 4'b0110 =512 Hz 4'b0111 =256 Hz 4'b1000 =128 Hz 4'b1001 =64 Hz 4'b1010 =32 Hz 4'b1011 =16 Hz 4'b1100 =8 Hz 4'b1101 =4 Hz 4'b1110 =2 Hz 4'b1111 =1 Hz	4'b0000
CLKRST	[3]	RTC clock count reset. 0 = RTC counter (2^{15} clock divider) enable 1 = RTC counter reset and disable Note: When RTCEN is enabled, CLKRST affects RTC.	0
CNTSEL	[2]	BCD count select. 0 = Merge BCD counters 1 = Reserved (Separate BCD counters) Note: When RTCEN is enabled, CNTSEL affects RTC.	0
CLKSEL	[1]	BCD clock select. 0 = XTAL 1/2^{15} divided clock 1 = Reserved (XTAL clock only for test) Note: When RTCEN is enabled, CLKSEL affects RTC.	0
RTCEN	[0]	Enables RTC control. 0 = Disables 1 = Enables Note: When RTCEN is enabled, you can change the BCD time count setting, 2^{15} clock divider reset, BCD counter select, and BCD clock select can be performed.	0

ステップ 2　rtc_print()関数は時間をプリントアウト、機能は读レジスタ BCDYEAR、BCDMON 、BCDDATE、BCDHOUR、BCDMIN、BCDSEC、BCDDAY を読み取り、年月日、時、分、秒、日曜をプリントアウトします；

ステップ 3　rtc コントローラと rtc tick timer をオフにします。同じく関数 rtc_enable() と rtc_ticktime_enable()呼び出し、今回はオフ機能。

<2> rtc_settime()コード：

```
void rtc_settime(void)
{
    //初期値をリセット

    unsigned long year = 12;
```

```
    unsigned long month = 5;
    unsigned long date = 1;
    unsigned long hour = 12;
    unsigned long min = 0;
    unsigned long sec = 0;
    unsigned long weekday= 3;

    // 時間を BCD コードに変換
    year = ( ((year/100)<<8) +(((year/10)%10)<<4) + (year%10)
    month = ( ((month/10)<<4)+ (month%10) );
    date = ( ((date/10)<<4) + (date%10) );
    weekday = (weekday%10);
    hour =( ((hour/10)<<4) + (hour%10) );
    min =( ((min/10)<<4) + (min%10) );
    sec =( ((sec/10)<<4) + (sec%10) );

    rtc_enable(true);
    // BCD コードを保存
    BCDSEC = sec;
    BCDMIN = min;
    BCDHOUR = hour;
    BCDDATE = date;
    BCDDAY = weekday;
    BCDMON = month;
    BCDYEAR = year;
    rtc_enable(false);
    printf("reset success¥r¥n");
  }
```

手順は 3 つあります：

ステップ 1　year、month、date の時間変数を初期値にリセットします；

ステップ 2　時間を BCD コードに変換します；

ステップ 3　BCD コードをレジスタ BCDYEAR、BCDMON、BCDDATE に保存します；

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 18.rtc
```

make

18.rtc のディレクトリ下に rtc.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

先ずメニューをプリントアウトし、ｓで時間をリセット、ｄで現在時間を表示します、下記図ご参照ください：

```
###############rtc test##############
[d] Display rtc realtime(hour:min:sec:weekday date/month/year)
[s] Reset rtc realtime(12:0:0:Tuesday 1/1/2012)
Enter your choice:s
reset success

###############rtc test##############
[d] Display rtc realtime(hour:min:sec:weekday date/month/year)
[s] Reset rtc realtime(12:0:0:Tuesday 1/1/2012)
Enter your choice:d
12 :  0 :  1     Tuesday,    5/ 1/2012
12 :  0 :  2     Tuesday,    5/ 1/2012
12 :  0 :  3     Tuesday,    5/ 1/2012
12 :  0 :  4     Tuesday,    5/ 1/2012
12 :  0 :  5     Tuesday,    5/ 1/2012
```

第 XIX 章　点線を描画

第一節　S5PV210 LCD コントローラ

LCD が文字、画像を正常に表示するには、 LCD ドライバの他に、LCD コントローラも必要です。LCD コントローラの機能は コントローラの主要機能は、システムメモリバッファ内の LCD 画像データを外部 LCD ドライバに送信され、必要な制御信号を生成します。例えば VSYNC、HSYSNC、VCLK。S5PV210 内部に LCD コントローラが集積します、構造図は：

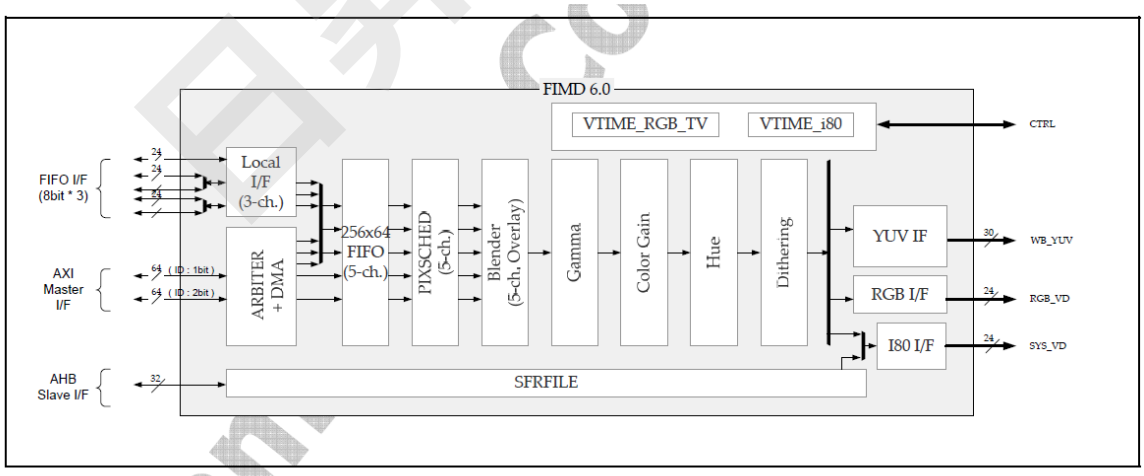


Figure 1-1　Block Diagram of Display Controller

コントローラは VSFR、VDMA、VPRCS、VTIME とビデオ･クロック･ジェネレータなどのいくつかのモジ

ュールで構成されます：

1）VSFR は 121 組のプログラマブルレジスタグループ、1 セットの gamma LUT レジスタ組（64 個レジスタ）、1 セットの i80 コマンドレジスタ組（12 個レジスタ）と 5 つの 256 * 32 パレットメモリのセットで構成されます。主要機能は LCD コントローラのコンフィギュレーションです。

2）VDMA は、LCD 専用の DMA 転送チャネルで、CPU の介入なしに、自動的にシステムバスから取得したビデオ･データを VPRCS に転送します。

3）VPRCS は 受信したデータを特定のフォーマット（例 16bpp/24bpp）に変換し、データ･インタフェースを通じて外部 LCD に送信します；

4）VTIME モジュールはプログラマブル･ロジック･コンポーネントと LCD ドライバ･インタフェース･タイミング制御機能、VTIME モジュールは VSYNC、HSYNC、VCLK 信号を生成します；

S5PV210 LCD コントローラ主要特性：

1）3 つのインタフェースをサポート：RGB/i80/YUV;

2）DMA プログラマブルをサポート

3）5 つの 256 * 32 ビットカラーパレット

4）最大 16MB の仮想画面

5）透明オーバーレイ（overlay）をサポート

6）多仕様/解像度の LCD をサポート

第二節　プログラム説明

完全なコードは、ディレクトリ 19.lcd ご参照ください。そして Mini210S の標準配置は 4.3ichLCD、タイプは H43-Hsd043I9W1。

1. main.c

手順は 3 つあります、中に ステップ 2、3 は LCD 相関で：

ステップ 1　シリアルポート初期化；

ステップ 2　LCD 初期化；

ステップ 3　メニューをプリントアウト、選択は：画面をクリア、クロス描き、水平線描き、垂直線描き

2. lcd.c

(一)　関数 lcd_init()：

S5PV210 lcd コントローラコンフィギュレーションは、下記図ご参照ください：

1.4.1 OVERVIEW OF PROGRAMMER'S MODEL

Use the following registers to configure display controller:

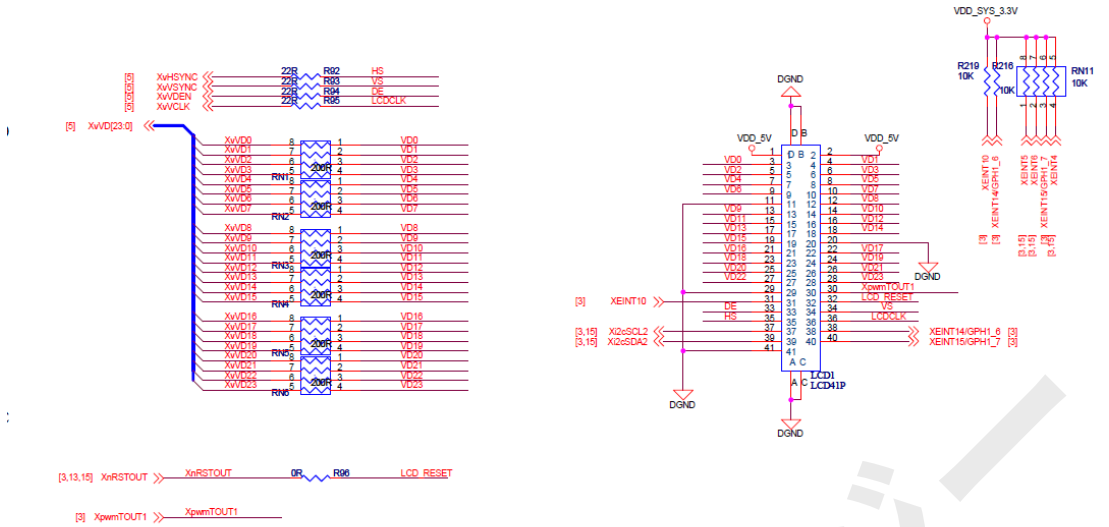
1. VIDCON0: Configures video output format and displays enable/disable.
2. VIDCON1: Specifies RGB I/F control signal.
3. VIDCON2: Specifies output data format control.
4. VIDCON3: Specifies image enhancement control.
5. I80IFCONx: Specifies CPU interface control signal.
6. VIDTCONx: Configures video output timing and determines the size of display.
7. WINCONx: Specifies each window feature setting.
8. VIDOSDxA, VIDOSDxB: Specifies window position setting.
9. VIDOSDxC,D: Specifies OSD size setting.
10. VIDWxALPHA0/1: Specifies alpha value setting.
11. BLENDEQx: Specifies blending equation setting.
12. VIDWxxADDx: Specifies source image address setting.
13. WxKEYCONx: Specifies color key setting register.

上記のシーケンスを参照して、 lcd.c 中関数 lcd_init()は lcd コントローラを初期化します、手順は 9 つあります：

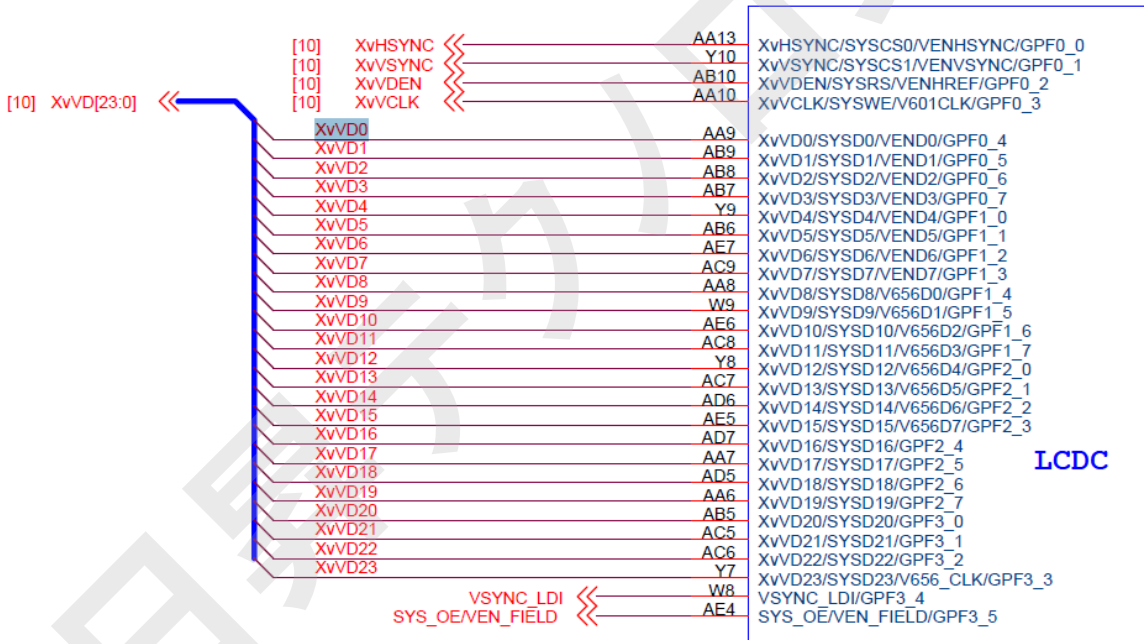
ステップ 1　関連ピンをコンフィグし、LCD に使用します、コード：

```
GPF0CON = 0x22222222;
GPF1CON = 0x22222222;
GPF2CON = 0x22222222;
GPF3CON = 0x22222222;
```

関連ピンは GPIO ピンの GPF0/1/2/3:



`XvVD0`を検索して、ピン構造図が下記の通りです：



ステップ 2　バックライトをオンにする、コード：

```
//バックライトをオンにする
GPD0CON |= (1<<4);
GPD0DAT |= (1<<1);
```

原理図：

ステップ 3　DISPLAY_CONTROL コンフィグ、データ出力パスを設定して、チップマニュアルに従い設定します。

Using the display controller data, you can select one of the above data paths by setting DISPLAY_PATH_SEL[1:0] (0xE010_7008). For more information, refer to Chapter, "Section 02.03. Clock controller".

ステップ 4　VIDCONx コンフィグ、インタフェースタイプ、クロック、極性、LCD コントローラ等；

1)　VIDCON0：

1.5.2.1 Video Main Control 0 Register (VIDCON0, R/W, Address = 0xF800_0000)

VIDCON0	Bit	Description	Initial State
Reserved	[31]	Reserved (should be 0).	0
DSI_EN	[30]	Enables MIPI DSI. 0 = Disables 1 = Enables (i80 24bit data interface, SYS_ADD[1])	0
Reserved	[29]	Reserved (should be 0)	0
VIDOUT	[28:26]	Determines the output format of Video Controller. 000 = RGB interface 001 = Reserved 010 = Indirect I80 interface for LDI0 011 = Indirect I80 interface for LDI1 100 = WB interface and RGB interface 101 = Reserved 110 = WB Interface and i80 interface for LDI0 111 = WB Interface and i80 interface for LDI1	000
L1_DATA16	[25:23]	Selects output data format mode of indirect i80 interface (LDI1). (VIDOUT[1:0] == 2'b11) 000 = 16-bit mode (16 bpp) 001 = 16 + 2-bit mode (18 bpp) 010 = 9 + 9-bit mode (18 bpp) 011 = 16 + 8-bit mode (24 bpp) 100 = 18-bit mode (18bpp) 101 = 8 + 8-bit mode (16bpp)	000
L0_DATA16	[22:20]	Selects output data format mode of indirect i80 interface (LDI0). (VIDOUT[1:0] == 2'b10) 000 = 16-bit mode (16 bpp) 001 = 16 + 2-bit mode (18 bpp) 010 = 9 + 9-bit mode (18 bpp) 011 = 16 + 8-bit mode (24 bpp) 100 = 18-bit mode (18bpp) 101 = 8 + 8-bit mode (16bpp)	000
Reserved	[19]	Reserved (should be 0).	0
RGSPSEL	[18]	Selects display mode (VIDOUT[1:0] == 2'b00). 0 = RGB parallel format 1 = RGB serial format Selects the display mode (VIDOUT[1:0] != 2'b00). 0 = RGB parallel format	0
PNRMODE	[17]	Controls inverting RGB_ORDER (@VIDCON3). 0 = Normal: RGBORDER[2] @VIDCON3 1 = Invert: ~RGBORDER[2] @VIDCON3 Note: This bit is used for the previous version of FIMD. You do not have to use this bit if you use RGB_ORDER@VIDCON3 register.	00
CLKVALUP	[16]	Selects CLKVAL_F update timing control. 0 = Always 1 = Start of a frame (only once per frame)	0

VIDCON0	Bit	Description	Initial State
Reserved	[15:14]	Reserved.	0
CLKVAL_F	[13:6]	Determines the rates of VCLK and CLKVAL[7:0]. VCLK = HCLK / (CLKVAL+1), where CLKVAL >= 1 Notes. 1. The maximum frequency of VCLK is 100Mhz(pad:50pf). 2. CLKSEL_F register selects Video Clock Source.	0
VCLKFREE	[5]	Controls VCLK Free Run (Only valid at RGB IF mode). 0 = Normal mode (controls using ENVID) 1 = Free-run mode	0
CLKDIR	[4]	Selects the clock source as direct or divide using CLKVAL_F register. 0 = Direct clock (frequency of VCLK = frequency of Clock source) 1 = Divided by CLKVAL_F	0x00
Reserved	[3]	Should be 0.	0x0
CLKSEL_F	[2]	Selects the video clock source. 0 = HCLK 1 = SCLK_FIMD HCLK is the bus clock, whereas SCLK_FIMD is the special clock for display controller. For more information, refer to Chapter, "02.03 CLOCK CONTROLLER".	0
ENVID	[1]	Enables/ disables video output and logic immediately. 0 = Disables the video output and display control signal. 1 = Enables the video output and display control signal.	0
ENVID_F	[0]	Enables/ disables video output and logic at current frame end. 0 = Disables the video output and display control signal. 1 = Enables the video output and display control signal. * If this bit is set to "on" and "off", then "H" is read and video controller is enabled until the end of current frame.	0

- ENVID_F=1、 フレームの終了後 LCD コントローラをオンにする;
- ENVID=1、 LCD コントローラをオンにする ;
- CLKSEL_F=1、 クロックソースを HCLK_DSYS=166MHz に設定;
- CLKDIR=1、 必要分周を選択;
- CLKVAL_F=14、周波数係数 15、すなわち VCLK = 166M/(14+1) = 11M;
- RGSPSEL=0、 RGB パラレル ;
- VIDOUT=0、 RGB インタフェースを使用 ;

未設定の bit はデフォルト値を使用します。

2) VIDCON1 :

1.5.2.2 Video Main Control 1 Register (VIDCON1, R/W, Address = 0xF800_0004)

VIDCON1	Bit	Description	Initial State
LINECNT (read only)	[26:16]	Provides the status of the line counter (read only). Up count from 0 to LINEVAL.	0
FSTATUS	[15]	Specifies the Field Status (read only). 0 = ODD Field 1 = EVEN Field	0
VSTATUS	[14:13]	Specifies the Vertical Status (read only). 00 = VSYNC 01 = BACK Porch 10 = ACTIVE 11 = FRONT Porch	0
Reserved	[12:11]	Reserved	0
FIXVCLK	[10:9]	Specifies the VCLK hold scheme at data under-flow. 00 = VCLK hold 01 = VCLK running 11 = VCLK running and VDEN disable	0
Reserved	[8]	Reserved	0
IVCLK	[7]	Controls the polarity of the VCLK active edge. 0 = Video data is fetched at VCLK falling edge 1 = Video data is fetched at VCLK rising edge	0
IHSYNC	[6]	Specifies the HSYNC pulse polarity. 0 = Normal 1 = Inverted	0
IVSYNC	[5]	Specifies the VSYNC pulse polarity. 0 = Normal 1 = Inverted	0
IVDEN	[4]	Specifies the VDEN signal polarity. 0 = Normal 1 = Inverted	0
Reserved	[3:0]	Reserved	0x0

VIDCON1 次の 2 つの bit を設定します：

- IHSYNC=1、極性反転
- IVSYNC=1、極性反転

極性反転の理由は下記の通りです：

H43-Hsd043I9W1.pdf(p13) タイミング図：VSYNC と HSYNC は低パルスです

S5PV210 マニュアル(p1207) タイミング図：VSYNC と HSYNC 高パルスなので、極性反転する必要があります。

ステップ 5 VIDTCONx コンフィグ、タイミングと長さと幅を設定します；

レジスタを設定する前に、LCD のタイミングセットを説明します、下記図ご参照ください：

1.3.11.2 LCD RGB Interface Timing

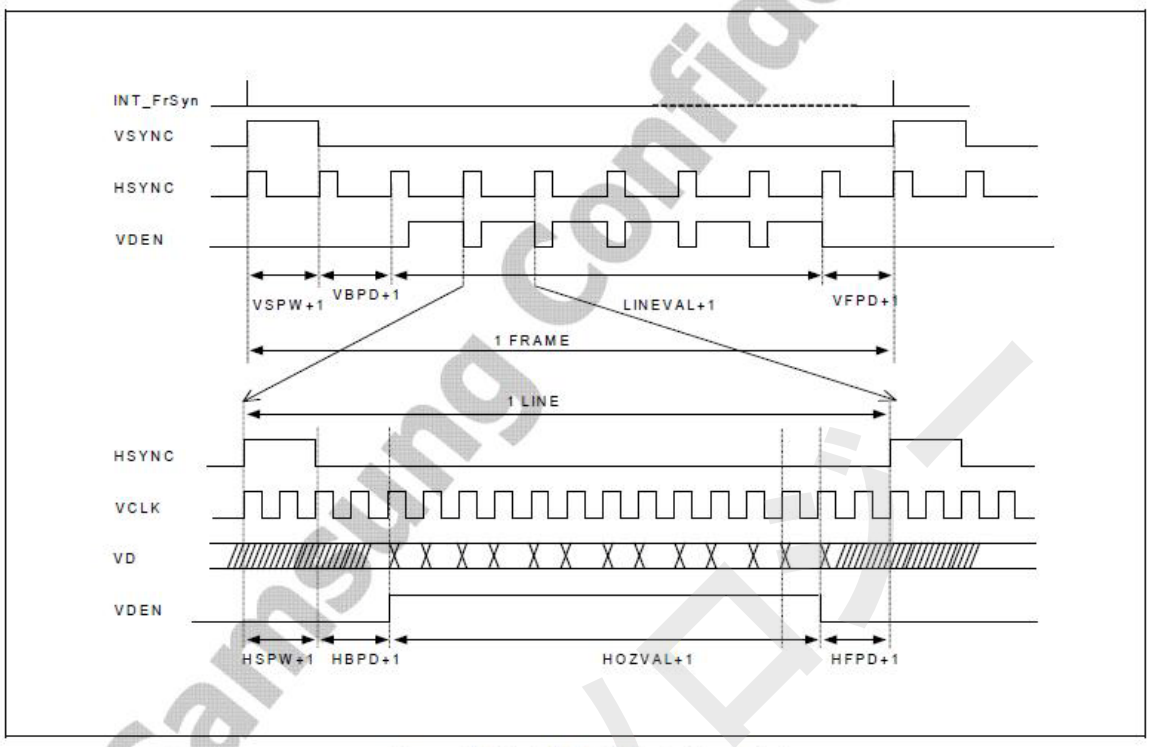


Figure 1-20 LCD RGB Interface Timing

各タイミングの意味は下記図ご参照ください：

- VBPD(vertical back porch)：表示在一帧图像开始时，垂直同步信号以后的无效的行数
- VFBD(vertical front porch)：表示在一帧图像结束后，垂直同步信号以前的无效的行数
- VSPW(vertical sync pulse width)：表示垂直同步脉冲的宽度，用行数计算
- HBPD(horizontal back porch)：表示从水平同步信号开始到一行的有效数据开始之间的 VCLK 的个数
- HFPD(horizontal front porth)：表示一行的有效数据结束到下一个水平同步信号开始之间的 VCLK 的个数
- HSPW(horizontal sync pulse width)：表示水平同步信号的宽度，用 VCLK 计算

<1> 各フレームの送信処理は、次のとおりです。：

1）VSYNC 信号は有効な場合、信号幅は（VSPW＋1）個 HSNC 信号周期、すなわち（VSPW ＋1）の無効ラインです。

2）VSYNC 信号パルスには、前の（VBPD ＋1）の HSYNC 信号サイクルを通過した後、有効なラインデータが得ます。だから、VSYNC 信号が有効後、また（VSPW+1+VBPD+1）の無効ラインを通過します；

3）（LINEVAL＋1）行の有効データを送信します；

4）最後には、（VFPD＋1）行の無効ラインです；

<2> 各行のピクセルの送信処理プロセス：：

1）HSNC 信号が有効な場合、1 行のデータの先頭を表示し、信号幅は（HSPW +1）個の VCLK 信号サイクル、すなわち（HSPW＋1）個の無効ピクセル；

2）HSYNC 信号パルスが通過後、（HBPD+1）の VCLK 信号サイクルを通過後、有効画素データが得ます；

3）（HOZVAL+1）ピクセルの有効データを送信します；

4) 最後には、(HFPD+1)個の無効ピクセルです；

上記の知識があれば、タイミング相関のレジスタ VIDTCON0、VIDTCON1 と VIDTCON2 を設定できます、コード：

```
#define HSPW        0
#define HBPD        (40 - 1)
#define HFPD        (5 - 1)
#define VSPW        0
#define VBPD        (8 - 1)

#define VFPD        (8 - 1)
// タイミング設定
VIDTCON0 = VBPD<<16 | VFPD<<8 | VSPW<<0;
VIDTCON1 = HBPD<<16 | HFPD<<8 | HSPW<<0;
//長と幅設定
VIDTCON2 = (LINEVAL << 11) | (HOZVAL << 0);
```

先ずレジスタ VIDTCON0 について、下記図ご参照ください：

1.5.2.5 Video Time Control 0 Register (VIDTCON0, R/W, Address = 0xF800_0010)

VIDTCON0	Bit	Description	Initial State
VBPDE	[31:24]	Vertical back porch specifies the number of inactive lines at the start of a frame after vertical synchronization period. (Only for even field of YVU interface)	0x00
VBPD	[23:16]	Vertical back porch specifies the number of inactive lines at the start of a frame after vertical synchronization period.	0x00
VFPD	[15:8]	Vertical front porch specifies the number of inactive lines at the end of a frame before vertical synchronization period.	0x00
VSPW	[7:0]	Vertical sync pulse width determines the high-level width of VSYNC pulse by counting the number of inactive lines.	0x00

レジスタ VIDTCON0

6.3 Data Input Format

Parallel 24-bit RGB Input Timing Table

Parameters	Symbol	Min.	Typ.	Max.	Unit	Conditions
DCLK frequency	fclk	5	9	12	MHz	
VSYNC period time	Tv	277	288	400	Th	
VSYNC display area	Tvd		272		Th	
VSYNC back porch	Tvbp	3	8	31	Th	
VSYNC front porch	Tvfp	2	8	93	Th	
HSYNC period time	Th	520	525	800	DCLK	
HSYNC display area	Thd		480		DCLK	
HSYNC back porch	Thbp	36	40	255	DCLK	
HSYNC front porch	Thfp	4	5	65	DCLK	

LCD チップマニュアル・タイミング図

2) VFPD VFPD : Vertical front porch、単位は行。LCD チップマニュアル・タイミング図で、VSYNC front porch は VFPD+1、Th は 1 行の VCLK の数を表示して、VFPD=8-1 行(Th) ;

3) VBPD : Vertical back porch、単位は行。LCD チップマニュアル・タイミング図で、 VSYNC back porch は VBPD+1、Th は 1 行の VCLK の数を表示して、VBPD=8-1 行(Th) ;

4) VSPW : Vertical sync pulse width、単位は行。LCD チップマニュアル・タイミング図で、その値を直接出していませんので、自己計算は必要です。公式は：VSPW= VSYNC period time - VSYNC display area - VSYNC back porch - VSYNC

front porch=288-272-8-8=0、VSPW は 0 と設定します ;

次はレジスタ VIDTCON1、下記図ご参照ください:

1.5.2.6 Video Time Control 1 Register (VIDTCON1, R/W, Address = 0xF800_0014)

VIDTCON1	Bit	Description	Initial State
VFPDE	[31:24]	Vertical front porch specifies the number of inactive lines at the end of a frame before vertical synchronization period. (Only for the even field of YVU interface).	0
HBPD	[23:16]	Horizontal back porch specifies the number of VCLK periods between the falling edge of HSYNC and start of active data.	0x00
HFPD	[15:8]	Horizontal front porch specifies the number of VCLK periods between the end of active data and rising edge of HSYNC.	0x00
HSPW	[7:0]	Horizontal sync pulse width determines the high-level width of HSYNC pulse by counting the number of VCLK.	0x00

VIDTCON1 レジスタ

6.3 Data Input Format

Parallel 24-bit RGB Input Timing Table

Parameters	Symbol	Min.	Typ.	Max.	Unit	Conditions
DCLK frequency	fclk	5	9	12	MHz	
VSYNC period time	Tv	277	288	400	Th	
VSYNC display area	Tvd		272		Th	
VSYNC back porch	Tvbp	3	8	31	Th	
VSYNC front porch	Tvfp	2	8	93	Th	
HSYNC period time	Th	520	525	800	DCLK	
HSYNC display area	Thd		480		DCLK	
HSYNC back porch	Thbp	36	40	255	DCLK	
HSYNC front porch	Thfp	4	5	65	DCLK	

LCD チップマニュアル・タイミング図

- HBPD：Horizontal back porch、単位は VCLK。LCD チップマニュアル・タイミング図で、HSYNC back porch は HBPD +1、 HBPD=40-1(VCLK);
- HFPD：Horizontal front porch、単位は VCLK。LCD チップマニュアル・タイミング図で、HSYNC front porch は HFPD +1、HFPD=5-1(VCLK);
- HSPW：Horizontal sync pulse width、単位は VCLK。LCD 芯チップマニュアル・タイミング図で、その値を直接出していませんので、自己計算は必要です。: HSPW+1 = HSYNC display area - HSYNC display area - HSYNC back porch- HSYNC front porch=525-480-40-5=0、 HSPW は 0 と設定します；

最後はレジスタ VIDTCON2、下記図ご参照ください:

1.5.2.7 Video Time Control 2 Register (VIDTCON2, R/W, Address = 0xF800_0018)

VIDTCON2	Bit	Description	Initial State
LINEVAL	[21:11]	Determines the vertical size of display. In the Interlace mode, (LINEVAL + 1) should be even.	0
HOZVAL	[10:0]	Determines the horizontal size of display.	0

NOTE: HOZVAL = (Horizontal display size) -1 and LINEVAL = (Vertical display size) –1.

- HOZVAL：HOZVAL = (Horizontal display size) -1=480-1=479
- LINEVAL：LINEVAL = (Vertical display size) -1=272-1=271

ステップ 6 WINCON0 コンフィグ、window0 のデータ形式を設定します；

S5PV210 LCD コントローラは overlay 機能があります、5 つの window をサポートします。ここでは 1 つ、 window0 を生成します、コード：

```
WINCON0 |= 1<<0;
WINCON0 &= ~(0xf << 2);
WINCON0 |= 0xB<<2;
```

本章の機能を実現するには、下記の bit を設定すれば十分です：

- ENWIN_F=1、 オン；
- BPPMODE_F= 1011、24bpp；

ステップ 7　VIDOsd0A/B/C コンフィグ、WINDOW0 座標系を設定します；

コード：

```
VIDOsd0A = (LeftTopX<<11) | (LeftTopY << 0);
VIDOsd0B = (RightBotX<<11) | (RightBotY << 0);
VIDOsd0C = (LINEVAL + 1) * (HOZVAL + 1);
```

WINDOW0 の左上と右下隅の座標の長さと幅を設定します。

ステップ 8　VIDW00ADD0B0 と VIDW00ADD1B0 をコンフィグ、フレームバッファのアドレスを設定します；

コード：

```
VIDW00ADD0B0 = FB_ADDR;
VIDW00ADD1B0 = (((HOZVAL + 1)*4 + 0) * (LINEVAL + 1)) & (0xffffff);
```

VIDWxxADD0	Bit	Description	Initial State
VBASEU_F	[31:0]	Specifies A [31:0] of the start address for Video frame buffer.	0

ステップ 9　SHADOWCON を設定し、 dma をチャンネル 0 を使用可能にします；

コード：

```
SHADOWCON = 0x1;
```

Window0 をしようします、 DMA がチャンネル 0 を使用します；

(二)関数 lcd_draw_pixel()：

lcd_init()初期化と LCD コントローラを設定した後、LCD で グラフィックを描けます、コードには描画関係のコードは 関数 lcd_draw_pixel()に基づきます、機能は LCD ポイントを描き、そして各ポイントでグラフィックを構成されます。

LCD でポイント描くの本質は、FrameBuffer に色の値を記入することです。次は関数 lcd_draw_pixel()を分析します、

コード：

```
void lcd_draw_pixel(int row、 int col、 int color)
{
    unsigned long * pixel = (unsigned long *)FB_ADDR;
    *(pixel + row * COL + col) = color;
}
```

FB_ADDR = 0x23000000、すなわち framebuffer のベースアドレス。row と col はオフセットの値、color は色の値、framebuffer で対応アドレスに色の値を記入すれば、LCD でそのポイントを描けます。

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

cd 19.lcd

make

19.lcd のディレクトリ下に lcd.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

先ずシリアルポートで下記のメニューをプリントアウトします：

```
###############LCD Test##############
[1] lcd_clear_screen
[2] lcd_draw_cross
[3] lcd_draw_hline
[4] lcd_draw_vline
[5] lcd_draw_circle
Enter your choice:
```

1：LCD をクリア；

2：LCD 左上隅にクロスを描き；

3：LCD 真ん中に水平線を描き；

4：LCD 真ん中に垂直線を描き；

第 XX 章　ADC 変換試験

第一節　S5PV210　ADC

S5PV210　ADC は 10bit 和 12bit をサポート、　そして 10 チャネル入力をサポートしており、入力のアナログ信号を 10bit/12bit のバイナリデジタル信号に変換します。本章でクロックは 5MHz、最大変換速度 1MSPS。この章では、基本の ADC 変換知識をご紹介します。構造図は下記図をご参照ください。

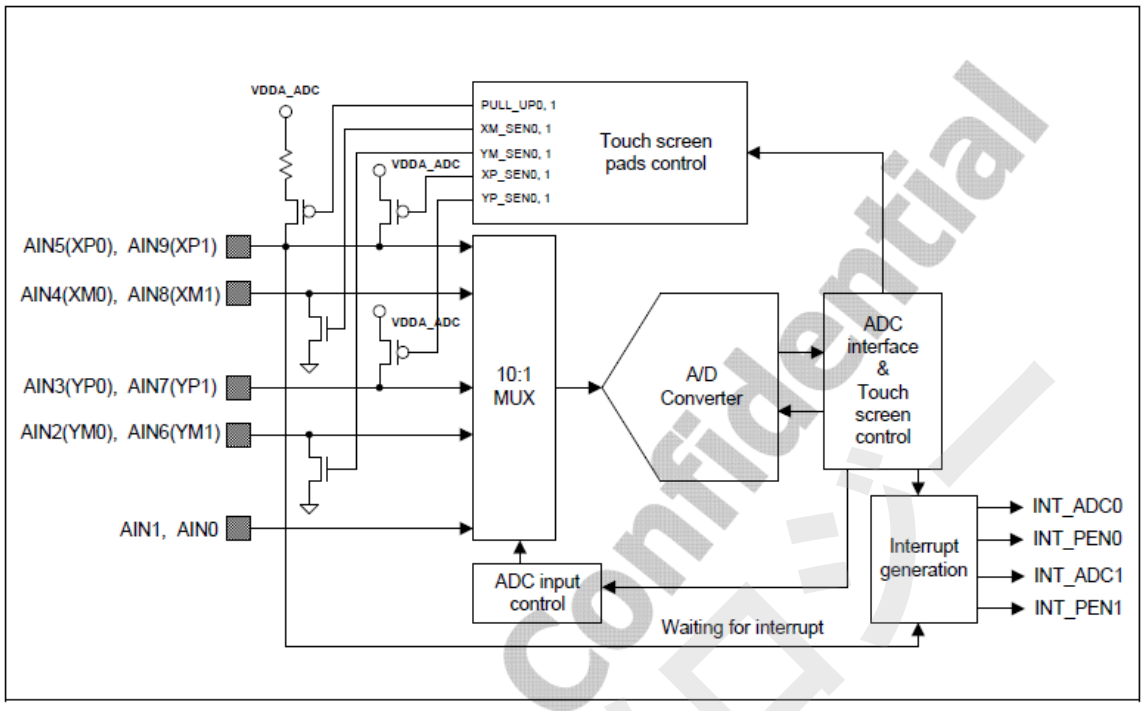
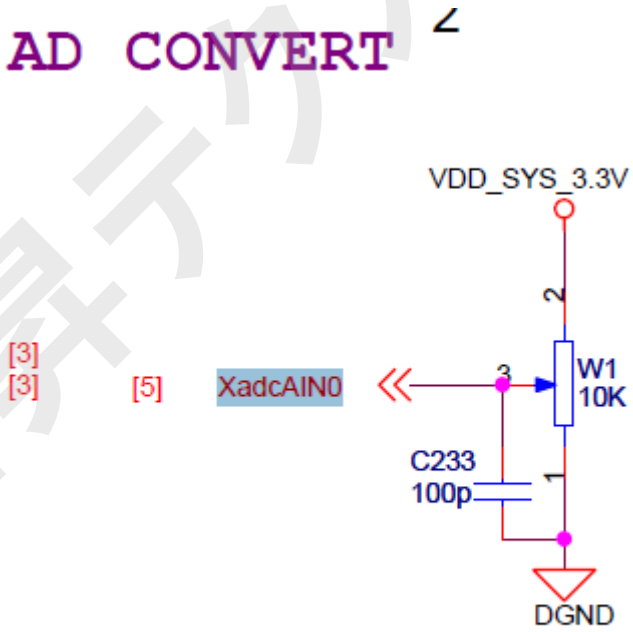


Figure 7-1 ADC and Touch Screen Interface Functional Block Diagram

Mini210S adc 回路図：



チャンネル 0 の入力は可変抵抗器に接続されており、可変抵抗を調整することで、、ADC は異なる値に変換することができます。

第二節　プログラム説明

完全なコードは、ディレクトリ 20.adc ご参照ください。

1. main.c

main()関数は簡単で、adc_test()関数を呼び出しで adc をテストします、adc_test()はディレクトリ adc.c に定義されます。

2. adc.c

関数 adc_test()コード：

```
void adc_test(void)
{
    printf("¥r¥n####################adc test#################¥r¥n");
    while(1)
    {
        printf("adc = %d¥r¥n"、read_adc(0));
        delay(0x100000);
    }
}
```

1 つの while ループを通じてチャンネル 0 の adc から変換値を読み取りし続きます、コア関数は read_adc()、手順は 5 つあります：

ステップ 1　クロック設定。コード：

```
TSADCCON0 = (1<<16)|(1 << 14) | (65 << 6);
```

先ず 12bit adc を使用し、分周を有効にして、分周係数を 66 と設定します

TSADCCONn	Bit	Description	Initial State
TSSEL	[17]	Touch screen selection 0 = Touch screen 0 (AIN2~AIN5) 1 = Touch screen 1 (AIN6~AIN9) This bit exists only in TSADCCON0. Note: An access to TSADCCON1 bits is prohibited when TSSEL bit is 0, and an access to TSADCCON0 bits except TSSEL is prohibited when TSSEL bit is 1. An access to TSSEL bit is always permitted.	0
RES	[16]	ADC output resolution selection 0 = 10bit A/D conversion 1 = 12bit A/D conversion	0
ECFLG	[15]	End of conversion flag(Read only) 0 = A/D conversion in process 1 = End of A/D conversion	0
PRSCEN	[14]	A/D converter prescaler enable 0 = Disable 1 = Enable	0
PRSCVL	[13:6]	A/D converter prescaler value Data value: 5 ~ 255 The division factor is (N+1) when the prescaler value is N. For example, ADC frequency is 3.3MHz if PCLK is 66MHz and the prescaler value is 19. Note: This A/D converter is designed to operate at maximum 5MHz clock, so the prescaler value should be set such that the resulting clock does not exceed 5MHz.	0xFF
Reserved	[5:3]	Reserved	0
STANDBY	[2]	Standby mode select 0 = Normal operation mode 1 = Standby mode Note: In standby mode, prescaler should be disabled to reduce more leakage power consumption.	1
READ_ START	[1]	A/D conversion start by read 0 = Disables start by read operation 1 = Enables start by read operation	0
ENABLE_START	[0]	A/D conversion starts by enable. If READ_START is enabled, this value is not valid. 0 = No operation 1 = A/D conversion starts and this bit is automatically cleared after the start-up.	0

ステップ 2　チャンネル選択：

```
ADCMUX = 0;
```

レジスタ ADCMUX を設定、チャンネル 0 を選択。

ステップ 3　変換起動。コード：

```
TSADCCON0 |= (1 << 0);
while (TSADCCON0 & (1 << 0));
```

先ずレジスタ TSADCCON0 の bit[0]を設定し、 A/D 変換を起動し、bit[0]を読み取り、変換機能を確認します。

ステップ 4　変換プロセスを確認します。コード：

```
while (!(TSADCCON0 & (1 << 15)) );
```

レジスタ TsdACCON0 の bit[15]読み取り、 1 になると転換が終了します。

ステップ 5　データを読み取り、コード：

return (TsdATX0 & 0xfff);

ここでは 12bit モードを使用しますため、レジスタ TsdATX0 の初頭の 12bit のみ読み取ります。

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

cd 20.adc

make

在 20.adc のディレクトリ下に adc.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

シリアルポート端末で数字をプリします、12bit ADC を使用するため、範囲は 0~4095 です、可変抵抗器を調整することによって、ACD 値を変更することができます、効果は次のとおりです。:

```
##################adc test################
adc = 1853
adc = 1868
adc = 1863
adc = 1862
adc = 1860
adc = 1860
adc = 1867
adc = 1862
adc = 1858
adc = 1869
adc = 1847
adc = 1877
adc = 1858
```

第 XXI 章　コマンド機能追加

第一節　コマンド機能について

ここでのコマンド機能は linux の shell と類似で、コマンドを入力して、プログラムが解析し、実行します。ここでは簡単な例を挙げます：：

1) help：ヘルプ情報

2) md：　　memory display メモリ表示

3) mw：　　memory write メモリ書き込み

4) loadb：　　シリアルポートで bin ファイルをメモリにダウンロード

5) go：メモリ内の bin ファイルを実行

第二節　プログラム説明

完全なコードは、ディレクトリ 21.shell ご参照ください。

1. main.c

main 関数コード：

```
while (1)
{
    printf("Mini210S: ");
    gets(buf);                          // コマンド入力待ち
    argc = shell_parse(buf、 argv); // 解析コマンド
    command_do(argc、 argv);            // 実行コマンド
}
```

注釈でわかります。関数 shell_parse()は/BL2/shell.c に定義されます、command_do()は/BL2/command.c に定義されます。

2. shell.c

shell_parse コアコード：

```
while (*buf)                            // 逐語読み取り
    {
        if (*buf != ' ' && state == 0)      // 文字' 'を読み取り
        {
            argv[argc++] = buf;
            state = 1;
        }
        if (*buf == ' ' && state == 1)              //スキップスペース
        {
            *buf = '¥0';
            state = 0;
        }
        buf++;
}
```

buf 内の文字を逐語読み取り、解析します。結果は argc と argv に保存しまう。例：コマンド go 0x21000000 を入力して、最終解析結果は argc = 2、argv[0] = “go”、argv[1] = “0x21000000”。

3. command.c

command_do()を説明します、コード：

```
// コマンドに基づきコードを実行します
```

```
int command_do(int argc、 char * argv[])
{
    if (argc == 0)
    return -1;

    if (strcmp(argv[0]、 "help") == 0)                    // help 実行
    help(argc、 argv);
    …                                                     // コード省略

    if 2(strcmp(argv[0]、 "loadb") == 0)       // loadb コマンド実行
    loadb(argc、 argv);
    …                                                     // コード省略
}
```

異なるコマンドにより、異なる実行関数を呼び出します。例えば help は関数 help()を呼び出します。

各コマンドの実行関数を説明します：

int help(int argc、 char * argv[])　：ヘルプ情報をプリントアウト。

int md(int argc、 char * argv[])　：メモリを読み取り、メモリ読み取り/書き込みは全部ポインタ操作。

int mw(int argc、 char * argv[])　：メモリ書き込み

int loadb(int argc、 char * argv[])　：シリアルポートで bin ファイルをメモリにダウンロードし、ファイルサイズとダウンロード先を取得します。最後に、getc()関数で bin ファイルを逐語受け取ります。

int go(int argc、 char * argv[])　：メモリ内の bin ファイルを実行、 ポインタを定義し、値を与え、呼び出します。

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 21.shell
# make
```

21.shell のディレクトリ下に shell.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

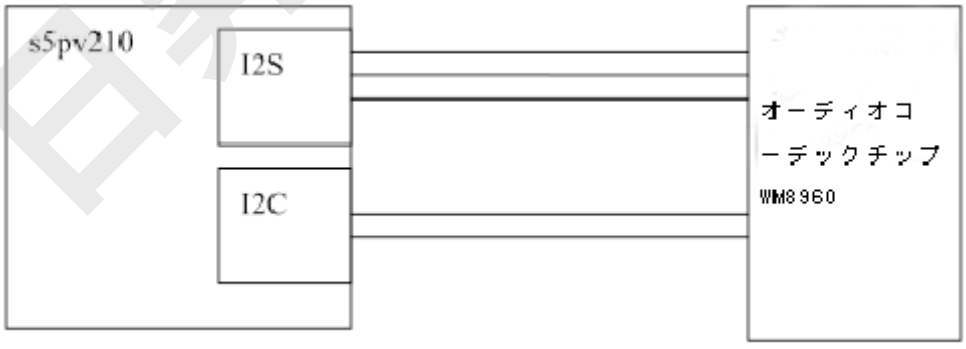
シリアルポートで｀Mini210S：｀をプリントアウト、そして help を入力し、ヘルプ情報をプリントアウトします：

```
mini210s: help
help usage:
md - memory dispaly
mw - memory write
loadb - loadb filesize addr
go - go addr
mini210s: mw 0x21000000 0x12345678
mini210s: md 0x21000000
21000000: 12345678 ffffffff ffffffff ffffffff
21000010: ffffffff ffffffff ffffffff ffffffff
21000020: ffffffff ffffffff ffffffff ffffffff
21000030: ffffffff ffffffff ffffffff ffffffff
21000040: ffffffff ffffffff ffffffff ffffffff
21000050: ffffffff ffffffff ffffffff ffffffff
21000060: ffffffff ffffffff ffffffff ffffffff
21000070: ffffffff ffffffff ffffffff ffffffff
21000080: ffffffff ffffffff ffffffff ffffffff
21000090: ffffffff ffffffff ffffffff ffffffff
210000a0: ffffffff ffffffff ffffffff ffffffff
210000b0: ffffffff ffffffff ffffffff ffffffff
210000c0: ffffffff ffffffff ffffffff ffffffff
210000d0: ffffffff ffffffff ffffffff ffffffff
210000e0: ffffffff ffffffff ffffffff ffffffff
210000f0: ffffffff ffffffff ffffffff ffffffff
mini210s:
```

本章で loadb と go コマンド機能を実現できます、次の章ではコマンドでオーディオチップ wm8960 のベアメタル·プログラムをダウンロード・実行します。。

第 XXII 章　WM8960 オーディオ再生

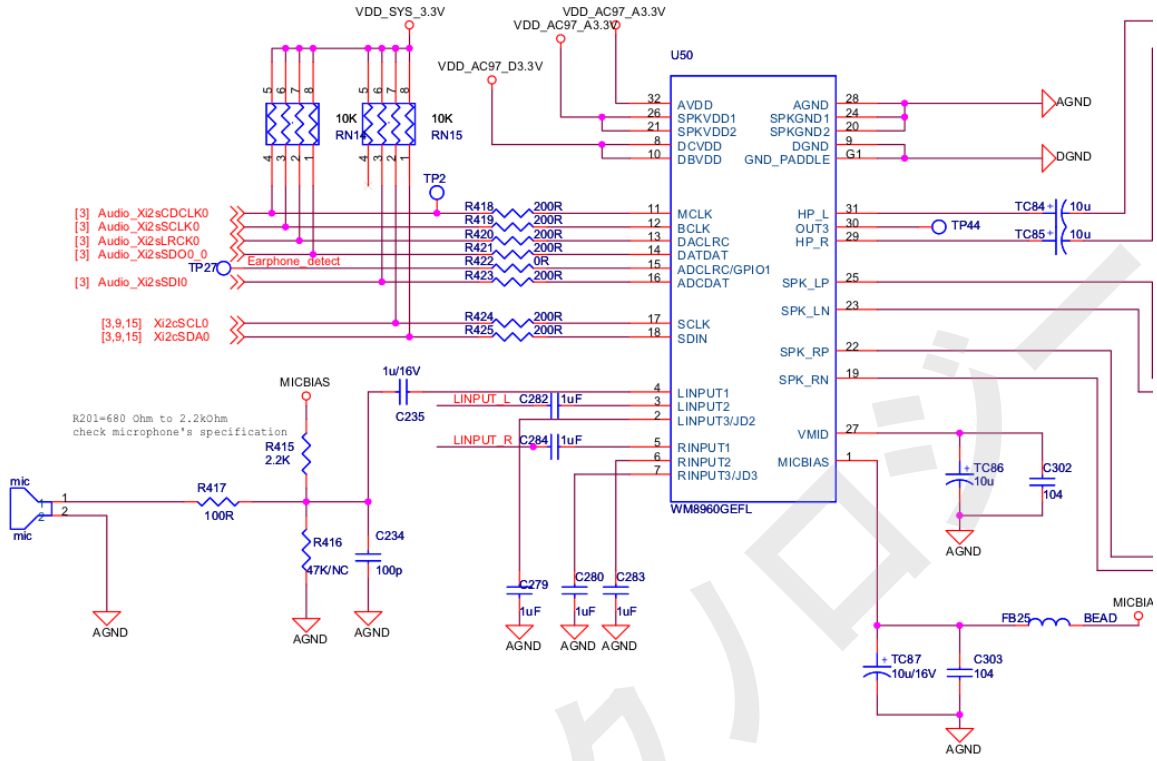
第一節　音频播放原理オーディオ再生の原理



S5PV210 は I2S/i2c を通じてオーディオコーデックチップ wm8960 と交信します。I2S 機能は音声データの送信；I2C の機能は制御情報を伝送します（例えば、ボリュームコントロール、ミュートなど）、wm8960 機能はエンコードとデコード。 wm8960 を駆動されるには、手順は 3 つあります：：(1) i2s 初期化、 (2) i2c

初期化、　(3) wm8960 初期化。

Mini210S 回路図：



第二節　プログラム説明

完全なコードは、ディレクトリ 22.audio ご参照ください。

1. Makefile

makefile でプログラムのリンクアドレスを 0x21000000 と設定します、すなわちプログラムは 0x21000000 でしか作動できません、よって、プログラムは最初にメモリ 0x21000000 にダウンロードする必要があります。

2. main.c

コード：

```
void main(void)
{
printf("Audio Test¥r¥n");
int offset = 0x2E;                    // オーディオデータ開始アドレス
short * p = (short *)0x22000000;      // オーディオファイルの開始位置
iic_init();                                  // i2c 初期化
```

```
wm8960_init();                          // wm8960 初期化
iis_init();                             // iis 初期化
//オーディオファイルのループ再生
while (1)
{
// polling Primary Tx FIFO0 full status indication.
while((IISCON & (1<<8)) == (1<<8));
IISTXD = *(p+offset);                   //  毎回  2byte データを送信
offset++;
if (offset > (882046-0x2e) /2)          // 2byte の数  = (ファイルサイズ  -  オフセット)/2
offset = 0x2E;
}
}
```

main 関数に手順は 4 つあります：

ステップ 1　iic_init()関数を呼び出し、i2c を初期化；

ステップ 2　wm8960_init()関数を呼び出し、wm8960 を初期化；

ステップ 3　iis_init()関数を呼び出し、i2s を初期化；

ステップ 4　i2s でオーディオデータを送信、オーディオファイルのループ再生します；

3. audio.c

audio.c のコア関数を説明します。

関数一 iic_init()：

```
void iic_init(void)
{
GPD1CON |= 0x22;                        //ピン· コンフィギュレーション
GPD1PUD |= 0x5;
I2CCON0 = (1<<7) | (0<<6) | (1<<5) | (0xe);
I2CSTAT0 = 0x10;
}
```

i2c 初期化は 3 つの手順があります：

ステップ 1　ピンをコンフィグし、i2c 機能を有効にします；

ステップ 2　i2c コントローラをコンフィグし、クロック設定、ack を有効にするなど；

ステップ 3　受信/送信機能を有効にします；

関数二 iic_write ()：

```
void iic_write(int slave_addr、 int addr、 int data)
{
// アドレス
I2CDS0 = slave_addr;
// s シングルとアドレス送信
I2CSTAT0 = 0xf0;

// 待ち
while ((I2CCON0 & 0x10) == 0);          // データ送信待ち
while ((I2CSTAT0 & 0x1));               //スレーブから ACK の返信待ち

// 7bit のアドレスと 9bit のデータ
I2CDS0 = addr<<1 | ((data>>8) & 0x0001);
I2CCON0 &= ~(1<<4);                     // 割り込みクリア
while ((I2CCON0 & 0x10) == 0);          // データ送信待ち
while ((I2CSTAT0 & 0x1));               // スレーブから ACK の返信待ち
I2CDS0 = (data & 0x00FF);
I2CCON0 &= ~(1<<4);                     // 割り込みクリア
while ((I2CCON0 & 0x10) == 0);          // データ送信待ち
while ((I2CSTAT0 & 0x1));               // スレーブから ACK の返信待ち

// p 情報送信
I2CSTAT0 = 0xd0;
I2CCON0 &= ~(1<<4);                     //割り込みクリア

//待ち遅延
int i=0;
for(i=0; i<50; i++);
return;
}
```

i2c 書き込みは 3 つの手順があります：

ステップ 1　s シングルとアドレスを送信します、 slave_addr の bit[0:6]は 7bit のデバイスアドレス、bit[7]=0 は送信；送信終了後、ACK 応答を待ちます；

ステップ 2　16bit のデータを送信的数据、先頭 7bit のデータはレジスタのオフセットアドレス、次の 9bit はレジスタの値です；

ステップ 3　p シングル送信、終了を表示します；
ステップ 4　遅延、p シングル送信完了を待ちます；

関数三 wm8960_init ()：

```
void wm8960_init(void)
{
#define WM8960_DEVICE_ADDR
0x34
// リセット
iic_write(WM8960_DEVICE_ADDR、 0xf、 0x0);
//  電源設定
iic_write(WM8960_DEVICE_ADDR、 0x19、 1<<8 | 1<<7 | 1<<6);
iic_write(WM8960_DEVICE_ADDR、 0x1a、 1<<8 | 1<<7 | 1<<6 | 1<<5 | 1<<4 | 1<<3);
iic_write(WM8960_DEVICE_ADDR、 0x2F、 1<<3 | 1<<2);
// クロック設定
iic_write(WM8960_DEVICE_ADDR、 0x4、 0x0);
// ADC-DAC 設定
iic_write(WM8960_DEVICE_ADDR、 0x5、 0x0);
…
}
```

wm8960_init()は iic_write()を呼び出して wm8960 チップを初期化します、手順は wm8960 チップマニュアルご参照ください：

ステップ 1　wm8960 のデバイスアドレスを確認します、wm8960 チップマニュアルで検索して、そのデバイスアドレスは 0x1a です。左に 1 bit を移動し、低位（最後尾）に 0（送信後）を変更します。
WM8960_DEVICE_ADDR = 0x34、　reset；
ステップ 2　power1 2 3 を設定；
ステップ 3　クロックをを設定；
ステップ 4　ADC-DAC を設定、（非静音）；
ステップ 5　audio interface を設定；
ステップ 6　volume を設定；
ステップ 7　mixer を設定；

関数四 iis_init ()：

```
void iis_init(void)
{
int N;
```

```
//  ピン・コンフィギュレーション、 i2s 機能を有効にする；
GPICON = 0x22222222;
//  i2s 関連クロックを設定；
EPLL_CON0 = 0xa8430303;                    // MPLL_FOUT = 67.7Mhz
EPLL_CON1 = 0xbcee;                        // from linux kernel setting
CLK_SRC0 = 0x10001111;
CLK_CON = 0x1;                             // 1 = FOUT_EPLL MUXI2S_A 00 = Main CLK
//  i2s コントローラを設定；
N = 5;
IISPSR = 1<<15 | N<<8;
IISCON |= 1<<0 | (unsigned)1<<31;
IISMOD = 1<<9 | 0<<8 | 1<<10;
}
}
```

i2s 初期化は 3 つの手順があります：

ステップ 1 ピンをコンフィグし、 i2s 機能を有効にします；

ステップ 2 i2s 関連クロックを設定します： EPLL_CON0 設定、EPLL に 67.7Mhz を出力され、クロックスイッチ CLK_SRC0 を設定します；

ステップ 3 i2s コントローラを設定します：分周設定、クロック選択、受送信モード設定；

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 22.audio
# make
```

22.audio のディレクトリ下に audio.bin を生成し、それを開発ボードにプログラムします。

プログラム shell.bin を使用し、audio.bin とオーディオファイルを DRAM にプログラミングします：

```
mini210s: loadb 11056 0x21000000
load bin file to address 0x21000000
load finished!
mini210s: loadb 882046 0x22000000
load bin file to address 0x22000000
load finished!
mini210s: go
go to address 0x21000000
Audio Test
```

loadb について：

loadb ファイルサイズ メモリアドレス、 loadb を実行後、シリアルポートでファイルを発送できます。

loadb 11056 0x21000000：　　audio.bin をメモリ 0x21000000 までダウンロードします。

loadb 882046 0x22000000：　　WindowsXP.wav をメモリ 0x22000000 までダウンロードします。

最後に go コマンドを実行し、0x2100000 にジャンプし audio.bin のコードを実行します。

第四節　実験現象

開発ボードにヘッドフォンをプラグインし、Windows XP の起動音がループで聞こえます。

第 XXIII 章　LCD 文字や画像表示

第一節　LCD 文字や画像表示

前の章では LCD で点線を描画する方法を説明しました、本章は進展関数、文字や画像描画方法をご紹介します。関数 lcd_draw_pixel()でポイントを拡張して、文字（キャラクター）を構成;画像も同じ原理で構成されます。

第二節 プログラム説明

完全なコードは、ディレクトリ 23.lcd_picture ご参照ください。

1. main.c

手順は 4 つあります：

ステップ 1： シリアルポート初期化；

ステップ 2： 関数 lcd_init()を呼び出し、LCD 初期化；

ステップ 3： 関数 lcd_draw_bmp()を呼び出し、LCD 上画像を描画します。{0>图片的数据保存在一个数组中、我们只需要把数组中的值一个个的读出来并写到 FrameBuffer 中即可<}0{>画像データがひと組の配列データに保存されます。ここで、データを読み出し、FrameBuffer に書き込みます<0}；

ステップ 4： 文字を描画、関数 printf()を呼び出して “FriendlyARM”をプリントアウトします。 printf()では putc()と lcd_draw_char()を呼び出しため、シリアルポートと LCD 共にプリント情報を表示します。

2. lcd.c

関数 lcd_draw_char()は文字を描画します：

ステップ 1　フォント取得。受信のパラメータを標的で、マトリックスアレイ fontdata8×16 から対応するフォント行列を取り出します。配列 fontdata8×16 は font8x16.c 位置に定義されます。このファイルは、Linux カーネルから引き出されます。
ステップ 2　改行の条件を確認します。｀¥n｀を検索した時時改行します、｀¥r｀を検索した時時キャリッジリターンラインします；
ステップ 3　8x16 ピクセル内で一文字を描きます。font_8x16.c 内に定義された文字は全部 8x16bit ピクセル内で構成されます、1bit は 1 ピクセル対応します。例えば当 bit 値は 1 になると関数 lcd_draw_pixel()を呼び出してブルーを描画し、0 は描画しません；
ステップ 4　カーソルを 8×16 ピクセルの位置移動します；

関数 lcd_draw_bmp()は画像を描きます：
ステップ 1： 配列から、ピクセルの色の値を取得します；
ステップ 2： 関数 lcd_draw_pixel()で配列の値を LCD 上で描画し、最後に画像を構成されます；

第三節　コードコンパイルとプログラミングの実行

コードをコンパイルし、Fedora 端末で下記のコマンドを実行します：

```
# cd 23.lcd_picture
# make
```

23.lcd_picture のディレクトリ下に lcd.bin を生成し、それを開発ボードにプログラムします。

第四節　実験現象

LCD 上に文字が表示します。