Mass Lock用
デジタルシステム仕様書

中川　憲保

平成19年4月15日

概 要

これはMass Lock用デジタルシステムについてまとめたものである。TAMA300でMass Lockのデジタル制御を実現させるために今回Digital Signal Processor(DSP)を用いた。DSPを用いた主な理由は、現在の方法では早い制御が必要（Unity Gain Frequency(UGF)で1kHz近く）でありアナログフィルタと同等の働きをするためには高速処理出来るものが必要であったことである。他にも、このシステムがスタンドアローンで動くのでノイズが比較的小さいと思われたこと、Txas Instruments(TI)社のDSPシステム構築環境が初心者にも使いやすかったこと、導入コストが安かったことなどがある。

これからこのDSPシステムについて説明をしていく。

目 次

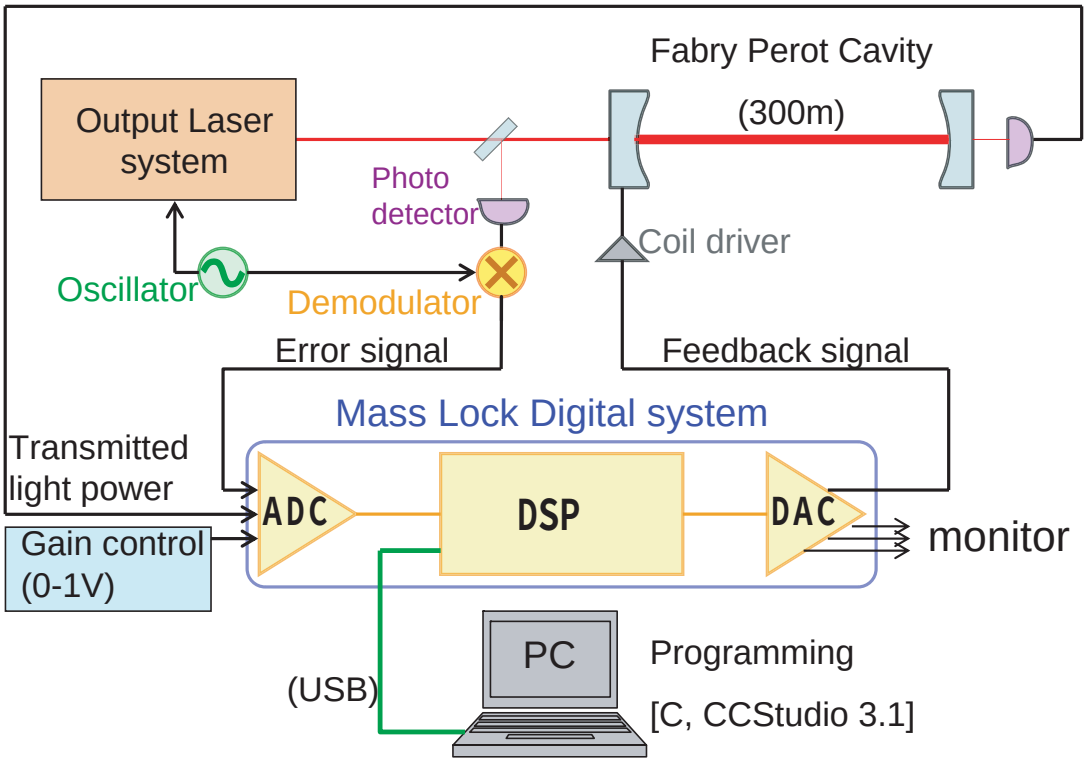

図 1: Mass Lock 用デジタルシステム

1　全体像

今回作成したシステムの主な部分は Analog to digital converter(ADC) から DSP、そして Digital to Analog Converter(DAC) までの部分である。

ADC への入力信号は３つある。１つ目は干渉計からの出射光を Photo Detector(PD) で受け Demodulator で復調された電気的信号。２つ目は Fabry-Perot(FP) 共振器の透過光量を電気的信号に変えたもの。３つ目は feedback のゲインを調整するための 0-1V の電圧信号。

これらの入力信号を DSP によって演算して DAC に出力する。DSP で行っている演算については後で説明するが、ここで実行するプログラムはＰＣ上で C 言語により書かれたプログラムを Code Composer Studio によりコンパイルしたものを USB ケーブルを介して DSP に送り込む。一度プログラムを送って実行させると、USB ケーブルを切断してもそのプログラムを実行し続ける。つまり、PC を用いずに DSP 単体で稼働させる事が出来る。

DAC からの出力信号はメインが１つ、モニター用として３つある。メインの信号はそのまま Coil Driver へと送られて FP 共振器を制御する信号として使われる。モニター用に関して初期の設定では１つ目がほぼメインと同等の信号。２つ目は透過光量によるトリガーと共振器ロック後のフィルタ変更信号。３つ目は透過光量信号。これらモニター用信号はプログラム次第では自由に変更出来る。

このデジタルシステムと干渉計の全体像は図１のようになっている。

2　ハード

デジタル部分を構成する主なハードとして、主要機器のパートで信号の演算に関わる部分を話して、外装のパートでケースや配線について話す。

2.1　主要機器

主要機器として DSP を搭載する TI 社の TMS320C6713DSK と、アナログとデジタルの相互変換を行うために平塚エンジニアリング社の DSK6000IFA を用いた。

2.1.1　TMS320C6713DSK

TMS320C6713 DSK はテキサスインストゥルメンツ社製 C67x^{TM} シリーズ浮動小数点 DSP を搭載した DSP スターター・キットである。この DSK には、225 MH z で動作する TMS320C6713 DSP が搭載されており、McASP および IIC の評価を容易に行なうことが可能である。ボード上には DSP 以外に、SDRAM、フラッシュ・メモリ、ステレオ・コーデック等、アプリケーションを実装、評価する為の環境が一通り用意されている。また拡張コネクタを搭載しているで、ドータ・ボードによる拡張を容易に行なうことが可能である。USB 経由 JTAG コントロールによるエミュレーションを採用されており、PC との接続が容易で、ノート PC 等でも簡単に開発、評価を行なうことがでる。実物は図 2 のようになる。

TMS320C6713 DSK の主な特徴をまとめておくと次のようになる。

- C6713 225MHz 搭載
- オンボード USB JTAG コントローラによるエミュレーション

- 8MB SDRAM、512KB フラッシュ・メモリ搭載
- ドータ・ボード拡張用コネクタ
- JTAG ヘッダ
- +5V 動作

TMS320C6713 のチップの特徴をまとめておくと次のようになる。

- 225MHz の周波数動作で 1350MFLOPS
- IEEE 準拠の 32bit 単精度浮動小数点演算と 64bit 倍精度浮動小数点演算
- 性能を落とさずに実行できるキャッシュ・ベース
  - 4KByte の一次データ・キャッシュ(L1D)(2WaySetAssociative)
  - 4KByte の一次プログラム・キャッシュ(L1P)(DirectMapped)
  - 192KB の L2 SRAM と 64KByte の L2 キャッシュ/SRAM(4 バンク)(計 256KB)
- 2 つのマルチチャネル・オーディオ・シリアル・ポート（McASP）
- 16 ステレオ・チャネルのインター・IC サウンド（IIS）
- S/PDIF 通信プロトコルとの互換
- 2 つのインター IC コントロール（IIC）ポート
- T1/E1、MVIP、IOM-2 などと直結可能な 2 チャネルのマルチチャネル・バッファード・シリアルポート (McBSP)
- SBSRAM、SDRAM を直結
- 非同期メモリ (SRAM や FIFO)、ブート用の 8/16bitROM を接続可能
- 16 チャネルのエンハンスド DMA

2.1.2 DSK6000IFA

ＤＳＰ多チャンネルアナログ入出力拡張ボード DSK6000IFA は、ＴＩ製ＤＳＫスタータキット（ＴＩ製ＤＳＫ）に接続して、高精度な多チャンネルディジタル信号処理技術の研究、高機能なＤＳＰアプリケーションの開発などに幅広く利用出来る IO ボードとなっている。ADC と DAC が搭載されており、サンプリング周波数の選択もこのボードから行う事が出来る。実物は図 3 のようになる。

DSK6000IFA の特徴をまとめておくと次のようになる。

- A/D:8 チャンネル同時サンプリング
- D/A:4 チャンネル同時変換
- 最高サンプリング周波数 200kHz、16bit
- 入出力レンジ ±1V

- デジタル外部入力、出力：8bitTTL レベル
- 電源:+5（DSK より供給）

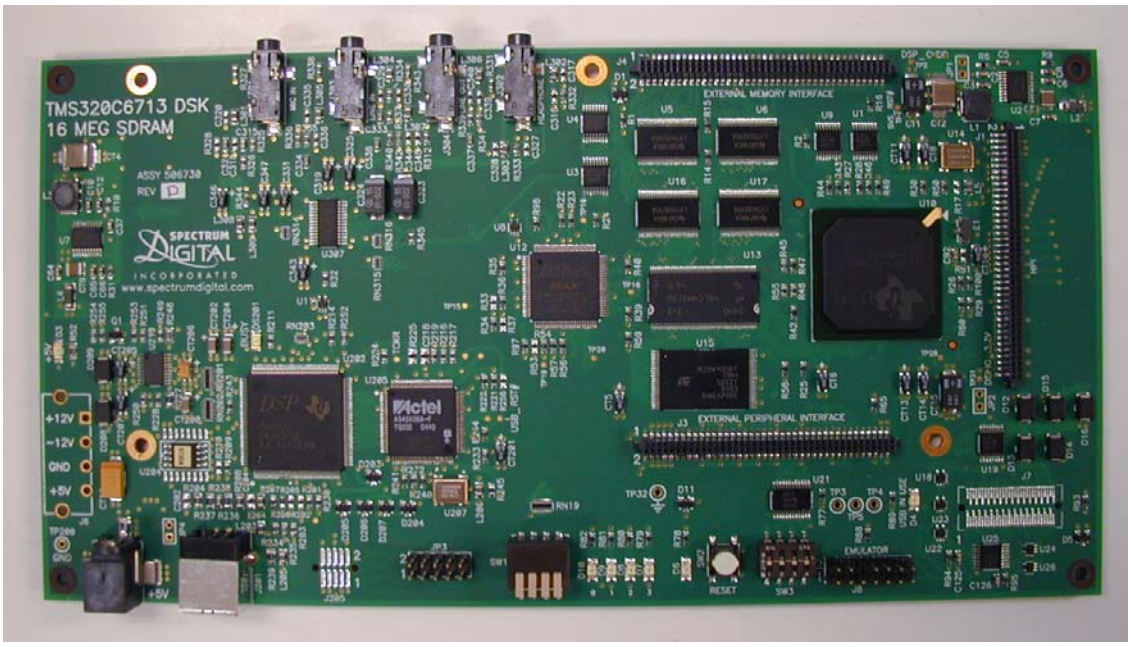

図 2: TMS320C6713DSK

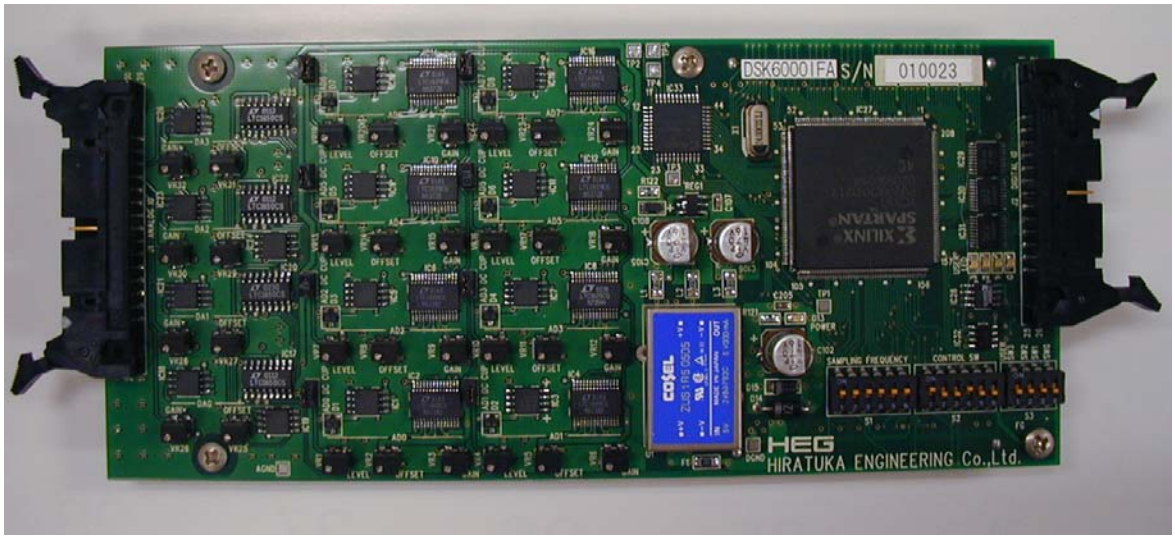

図 3: DSK6000IFA

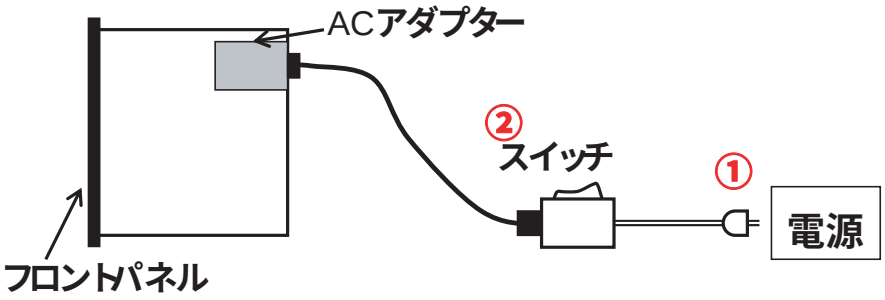

図 4: 電源の入れる手順

2.2 外装

外装についてはフロントパネル部、電源部、内部配線について解説を書いていく。

2.2.1 フロントパネル

フロントパネルにある入力端子は以下の通りである。

入力端子 LEMO 同軸ケーブル×４、入力電圧-1〜+1V

出力端子 LEMO 同軸ケーブル×４、出力電圧-1〜+1V

外部入力 USB2.0(USB1.1 等は使用不可) × 1

注意する点としては、それぞれ入出電力の制限に配慮することと、USB は必ず 2.0 で使用するということである。

2.2.2 電源部（バックパネル）

バックパネルには電源コネクタが一つある。DSP 回路を保護するために直接電源につなぐことを避けるように DSK 側から指示があり、その電源の入れ方に順番があるので次に記しておく。

1. 電源ケーブルを DSP ボックス側とスイッチ BOX につなぐ
2. スイッチ BOX のコンセントを電源側に挿す
3. スイッチを入れる

2.2.3 内部配線

内部配線について、入出力の信号線はツイストケーブルで DSK6000IFA の端子台につながって、そのまま AD、DA ポートに入力される。外部入力の USB については USB コネクタの中継器を経て、TMS320C6713DSK の端子につながっている。電源についてはバックパネルで AC アダプターに電源が供給された後に、そこから 5V 電源として TMS320C6713DSK に入力される。

3　ソフト

ここではコンパイラの節で実際に DSP を動かすための手順、プログラムの節で実際のプログラムの中身について説明する。

3.1　コンパイラ

プログラムは C 言語で書き、TI 社の Code Composer Studio という専用ソフトによりコンパイルする。DSK にも簡易版が付属しておりこれでも十分に稼働する。

今後よりもっと複雑な事も出来るように Code Composer Studio IDE というソフトも手元にあるので必要に応じて使い分けて下さい。

3.1.1　CCS のインストール

o CD-ROM から”CCS DSK v.3.0”と”DSK6713 Drivers and Target content”を画面の指示に従いインストールする。next か yes を押すだけで大丈夫。

o インストールが終われば、画面に出てきた CCStudio3 のショートカットをクリック

o 初回は DSP の選択画面が出てくるので、ここで C6713DSK を選択し、import する。

o セーブして CCS をスタートする。以上、ここまででインストール作業は終了。

3.1.2　プログラムの実行

o まず、Cavity Mass Lock 用プログラムはここに置いておきます。この Mass lock フォルダを My project にコピーする。

o 不足するライブラリのコピー

・DSK6x11 を `C:\CCStudio\C6000` のフォルダへコピー

`(@ http://www.icrr.u-tokyo.ac.jp/~nakagawa/DSK6X11.zip)`

・rts6701.lib を `C:\CCStudio\C6000\cgtools\lib` のフォルダへコピー

`(@ http://www.icrr.u-tokyo.ac.jp/~nakagawa/rts6701.zip)`

o CCS を起動後、Debug-connect で DSP に接続。

o Project-open で Mass lock フォルダ内の coplete.pjt を開く。

o パスの指定があるので、それぞれ `C:\CCStudio` 内の各ファイルを指定してあげる。

o Rebuild All で compile する。

o File-Load Program で thru.put を開く。

o 左にある人が走ってるアイコンでラン。これでプログラムが動く。

o Debug-disconnect で DSP と接続遮断。プログラムは動き続ける。

以上で、稼働までの操作説明終わり。

3.1.3　変えるべき変数

初期状態は、ある日のベストの値を入力しているので、干渉計の環境が変わり次第逐次変更して下さい。

・透過光量オフセット
・制御をかけるかどうかのトリガーレベル
・ロック判断 counter シグナルの閾値、および時間間隔
・lock 前 filter gain
・lock 後 filter gain
以上になります。

3.2 mass lock 用プログラム

ここではプログラムの内容について書いていく。各腕のプログラムは、06 年 08 月末時点のものがそれぞれ下の場所に置いてある。

NS 腕 `http://www.icrr.u-tokyo.ac.jp/~nakagawa/main/NSmass_1.0.0_060831.c`

WE 腕 `http://www.icrr.u-tokyo.ac.jp/~nakagawa/main/WEmass_1.0.0_060831.c`

プログラムの中身の詳細については次以降の節で説明していく。

4　諸注意

```
line 1-26 （諸設定、省略）
line 27-39 変数設定(↓)

#define ORDER 12  フィルタの次数定義

/************ Constant Value ********************************************/
double Troffset  = 0.008;  /*Transmitted light offset*/  透過光量のオフセット調整（単位[V]）
double Trtrigger = 0.023;  /*Trigger threshold by Tr*/  透過光量によるトリガー値（単位[V]）
double Trcounter = 0.030;  /*Lock judgement counter threshold by Tr*/
                           ロックしたと判定する時の光量設定（単位[V]）
double fgbefore  = 2.25;   /*filter gain before lock*/  ロック前のfeedback内部ゲイン
double fgafter   = 0.6;    /*filter gain after lock*/  ロック後のfeedback内部ゲイン
/************************************************************************/

double u[ORDER/2][2];           /* intermidiate buffer */
                                フィルターに用いる変数の設定
                                フィルタの次数が増えた場合ここも変更する
int    LC = 0;                  /* counter for lock signal */
                                ロックしたと判定するまでのカウンターリセット

line 40-65  フィルタの定数設定
line 66-73  （諸設定、省略）
line 74-100 main文(↓)

void main(void)
{
    /*define variables*/
    int k,m;

 /*program*/
    /*buffer clear*/
    for (m=0; m<ORDER/2; m++)
    for (k=0; k<2; k++)
    {
     u[m][k] = 0.0;
    }
ここまで、フィルタ演算に使う変数リセットプログラム
フィルタの次数が増えた場合ここも変更する
以下は特に変更の必要なし
    *(unsigned volatile int *)EMIF_CE2 = 0x0120C422; /* CE2 Bus timming control */

 enableSpecificINT(4);      /* enable INT4(External INT)         */
 enableNMI();        /* enable NMI(Non Maskable Interrupt)*/
 enableGlobalINT();       /* set GIE(Global Interrupt Enable)  */

    for (;;)
    {
            /* Keep waiting for Timer0 Interrupt */
    }
}
```

図 5: 変数設定プログラム

```
line102-126  入力部分
interrupt void int4(void) ここから制御プログラム開始
{
 /*define variables*/     演算に用いる変数定義
 double SI, G, Tr, Tg, un, cal;
    int m;
    short SI0, G0, Tr0, So, DA1, DA2, DA3;

この"/*"から"*/"までは演算速度を測定するためのプログラム
速度測定を行うと制御出来なくなるのでコメントアウト
/* clock_t start;
 clock_t overhead;
 clock_t elapsed;
 start = clock();
 overhead = clock() - start;
 start = clock();
*/

ADで入力した信号に変数を割り当てる
    SI0  = *( volatile short *)DSKIF_AD0;  /*Signal Input*/  エラーシグナルのインプット
//    LS0 = *( volatile short *)DSKIF_AD1;  /*Locked sign*/  現在は空き
    G0  = *( volatile short *)DSKIF_AD2;  /*Gain for feedback*/ feedbackゲインを外部からコントロール
    Tr0 = *( volatile short *)DSKIF_AD3;  /*Transmitted light*/  透過光量インプット
さらにここで演算しやすいように変換
 SI   = (double)SI0;     /*modified Signal Input*/
 G   = (double)G0;     /*modified Gain for feedback*/
 Tr  = (double)Tr0*3.051757e-5 + Troffset; /*modified Trs. light original0.015*/
           プログラム中で「V」で表現出来るように変換し、透過光量のオフセット加える
 DA3 = (short)(Tr * 32678.) ;   /*Trans cal. light monitor*/
           内部でオフセットを加えた透過光量をDAのch3からモニターとして出力
```

図 6: 入力設定プログラム

```
line127-132  ゲイン調整
/****************Gain value modify (G -> 0<G<1)*************/
外部入力電圧0～1Vがgeinで0～1倍になるように調整する
 if (G<0)  外部入力電圧が負の場合は0とする
  G = 0.;
 else      正の場合はGがlongで定義されているので0-1倍になるよう調整
  G = G*4.8828125e-4;
```

図 7: ゲイン調整プログラム

```
line133-151  トリガー、ロック判定
/****************Triger setting [Tg=Triger]*****************/
 if (Tr > Trtrigger) 透過光量がトリガーレベルより上なら、トリガーがオンになる（Tg=1となる）
  Tg = 1.;
 else
  Tg = 0.;

    DA2 = ( short)Tg * 8192 + (short)LC ;  /*Triger and LC monitor*/
      ch2でトリガーのオンオフをモニターし、トリガーがオンになった後にロック判定のカウンターが
      増加していく。
/****************Lock judgement [LC=Lock Counter]***********/
ロック後フィルタの低周波ゲインアップを行うまでの時間設定
 if (Tr > Trcounter)
   {
    if (LC < 1000) /*counter > 5msec*/   現在は5msecで設定
     LC++;
    else
        LC = 1000;
   }
   else
  LC = 0;  ロックが落ちたらカウンターもリセットする
```

図 8: トリガー、ロック判定プログラム

```
line152-179  制御フィルタ
/****************Error signal calculation*******************/
ロック前後で異なるフィルタを使用する
    if (LC < 1000)/*****Lock Before*****/  ロック安定前
    {
  cal = _rcpdp(Tr)*0.1;          /*Normalization function*/  透過光量の逆数を取る
     SI = SI * cal * Tg;   エラーシグナルを透過光量で規格化し、トリガーをかける
  DA1 = ( short)(SI*0.1) ;  ch1から規格化、トリガー後のエラーシグナルをモニター
   for (m=0; m<ORDER/2-1; m++)  フィルタ演算
        {
            un = am[m][0]*u[m][0] + am[m][1]*u[m][1] + SI;
            SI = bm[m][0]*un + bm[m][1]*u[m][0] + bm[m][2]*u[m][1];
            u[m][1] = u[m][0];          /* shift of data */
            u[m][0] = un;               /* shift of data */
        }
        SI = SI*5.094971566e-006*fgbefore;
    }

    else/*****Lock After******/  ロック安定後
                          ロック安定後は規格化は行わない
                          また、トリガーもかけない
    {
     DA1 = ( short)(SI*0.1) ;
   for (m=0; m<ORDER/2; m++)
        {
            un = cm[m][0]*u[m][0] + cm[m][1]*u[m][1] + SI;
            SI = dm[m][0]*un + dm[m][1]*u[m][0] + dm[m][2]*u[m][1];
            u[m][1] = u[m][0];          /* shift of data */
            u[m][0] = un;               /* shift of data */
        }
        SI = SI*4.216521791e-006*fgafter;
    }
```

図 9: 制御フィルタプログラム

```
line180-186  オーバーフロー対策

DACで変換して出力する際に、内部の値がレンジオーバーしていた場合、信号が狂ってくるので
オーバーフローが起きないように最後に対処しておく
→内部での演算はdoubleで行っているが、DAで出力する際にlongに変換しなければならず
  そのためこのような問題が生じてしまう
/**************Over flow problem*********************/
   SI= SI*G;
    if(SI>=32767) So = 32767;
 else if (SI<-32768) So=-32768;
 else So=(short)SI;
```

図 10: オーバーフロー対策プログラム

```
line187-196  出力プログラム

   *( unsigned volatile short *)DSKIF_DA0 = So; /*Signal Output*/  feedback信号
   *( unsigned volatile short *)DSKIF_DA1 = DA1; /*Normalized and triggerd error Output*/
                                    透過光により規格化、トリガーされたエラー信号モニター
   *( unsigned volatile short *)DSKIF_DA2 = DA2; /*Triger and LC monitor Output*/
                                    トリガーとロック安定判定カウンターモニター
   *( unsigned volatile short *)DSKIF_DA3 = DA3; /*Offsetted transmitted light Output*/
                                    オフセット調整された透過光量モニター
以下３行は演算時間測定を行った時用の時間表示プログラム
//    elapsed = clock() - start - overhead;
//    printf("Time = %ld cycles\n", (long)elapsed);
//    printf("Time = %ld %e cycles\n", (long)elapsed,(double)elapsed/CLOCKS_PER_SEC);

}制御用プログラムは以上ここまで

line197-223  （諸設定、省略）
```

図 11: 出力プログラム