HICALI の LabVIEW による制御 No.2

吉川智裕

2003 年 3 月 20 日

No.1 では、HICALI を LabVIEW から制御することの利点と、実際に HICALI を制御する VI を作成するためにはどのような方法を用いたらよいのかについて考察した。ここでは、従来のテキストベースのプログラムとの過程の違いを示し、さらに効率的にプログラムを行うためのプログラムのモジュール化の方針について考える。

1　従来のプログラムとの比較

1.1　従来のプログラム

従来のテキストベースで行っているプログラムは、以下のような流れで動作する。

1. PC 側のプログラムによって、DSP を起動・初期化する
2. DSP プログラムを PC からロードする
3. DSP、PC はそれぞれのプログラムに従い、データのやりとりをする
4. PC 側のプログラムから DSP を閉じる

これらは、全て C 言語のプログラムで書かれ、通信用の関数が DSP のメーカで用意されている。

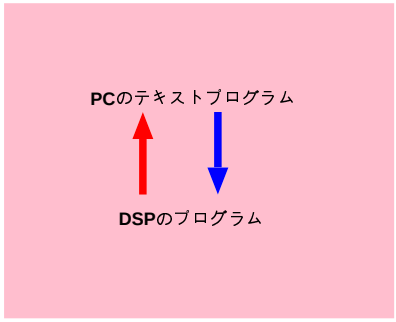

図 1: テキストベースのプログラム

今回の制御では DSP は 1 つしか使っていないが、一般には複数枚の DSP ボードを PCIbus にさして使うことができる。そのため、DSP ボードを区別するために、通信用の関数には以下のような control block 構造体を渡す必要がある。PC 側のプログラム用のヘッダファイルで以下のように定義されている。

```
typedef struct control_block {
```

```
        int             btotal;                         /* ボード枚数 */
        int             dtotal;                         /* ＤＳＰ数 */
        HANDLE  dspfd[MAX_DSP];         /* デバイスハンドル */
        int             bnum[MAX_DSP];          /* ボード番号 */
        int             dsptime[MAX_DSP];       /* タイムアウト値 */
        char    msg[128] ;                      /* エラーメッセージバッファ */
} CTLBLK;
```

1.2 LabVIEW を使ったプログラム

No.1 で述べたように、LabVIEW のブロックダイアグラムから直接 C で書かれた関数を呼び出すことはできない。そこで、CIN(Code Interface Node) を使い、DSP と通信するための関数を CIN の中で呼び出すことにした。このとき、テキストベースのプログラムで示したように、DSP と通信するための関数には control block 構造体を渡してやらなければならない。そこで、LabVIEW 上で構造体に相当する cluster に control block 構造体と同じものを格納し、CIN にはこの control block cluster を入力、出力するノードを設置して、VI の中で複数回呼び出される CIN 同士で control block cluster を伝えていくようなブロックダイアグラムを組んだ。

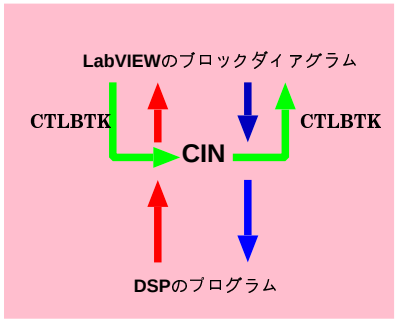

図 2: LabVIEW で CIN を使ったプログラム

2 プログラムのモジュール化

普通の C のプログラムでも行われるように、プログラムの中で、何度も繰り返して使うような部分がある場合、その部分は関数にしてサブルーチン化してしまうとソースを書くのも楽になり、完成したプログラム自体も小さくて済む。LabVIEW でも同じようなことを行うことができる。ブロックダイアグラムのある部分を抜き出し、その部分を別の VI として保存する。こうして作った VI を、メインの VI からサブ VI として呼び出すことができる。また、CIN も同様で、同じソースをロードしている CIN がサブ VI を含めてひとつの VI の中に複数あったとすると、LabVIEW はメモリ上に一つの CIN プログラムしか呼び出さない。そのため、メモリの節約になるのである。以上のようなことから、プログラムのモジュール化を以下のような方針で行った。

- CIN の用途を整理し、必要最低限の CIN にまとめる

- CCD の駆動に必要なサブ VI を整理し、それらの組合せで VI を作るようにする

2.1 CIN の設計

CIN の設計を考える上で、CIN の特性について考えなければならない。前で述べた、CIN の種類を最低限にすることでメモリの節約になるということの他に、CIN の実行時間は極力短くするように気をつけなければならない。CIN の実行中は、LabVIEW は CIN からの出力があるまで (kill を送る以外には) 中断することもできず、また、実行時間があまり長引くと LabVIEW の動作が不安定になってくることが経験的にわかった。

このような点を考えて、4 つの CIN プログラムを作った。

- dspinit
- dspclose
- dspwrite
- dspread

dspinit と dspclose は、それぞれ DSP を起動して初期化し DSP プログラムをロードするものと、プログラム終了後に DSP を閉じるものである。

dspwrite と dspread は、それぞれ DSP にデータを書き込むもの、DSP からデータを読み取るものである。これらは、任意のデータ量の送受信を行うことができるようにできている。DSP とデータの送受信を行う関数は、データの先頭を示すポインタとデータの量を別々に与えなければならないので、単純にはいかない。まず、dspwrite では、LabVIEW からデータを渡すときに必ず配列で渡す仕様にした。LabVIEW の配列は、C の配列と異なりデータ量を含む。そこで、CIN のプログラムの中で配列のサイズを取り出し、DSP にデータを送る関数に入れている。一方で、dspread は、先に読み取るデータ量を CIN に教えてやらなければならない。これは省略のしようがなかったので、CIN の入力にデータ量を入力するノードを作った。

このように read と write を分けたことによって、CIN が DSP からのデータを待つ時間を短くすることができた。PC 側のプログラムで、DSP からのデータを読み取る関数まできた時、DSP 側のプログラムがまだデータを送り出す関数まできていないと、PC 側のプログラムは DSP がデータを送信してくるまで待つようになっている。テキストベースのプログラムの場合は待たせておけばよかったのだが、CIN の中で待たせるようにすると、前述のように LabVIEW を中断することができないばかりか LabVIEW が不安定になるという現象が起きる。そのため、データの受信を待たなければならないような場面 (露光中など) には、CIN に入る前に必要な時間待たせておき、CIN での待ち時間を最低限にするように工夫した。

2.2 サブ VI の設計

CCD を駆動する上で必要な機能にわけて、以下のようなサブ VI を設計した。

- dspinit.vi
- dspclose.vi
- hicaliExposure.vi
- hicaliReadout.vi
- hicaliwipe.vi
- readFits.vi

- writeFits.vi

dspinit.vi と、dspclose.vi はそのまま同じ名前の CIN が格納されているだけの VI で、それぞれ DSP の初期化し、閉じる。hicaliExposure.vi、hicaliReadout.vi、hicaliwipe.vi はそれぞれ、CCD の露光、読み出し、露光前の CCD array にたまっている電荷を除く wipe のための VI である。readFits.vi と writeFits.vi は、DSP とは関係ないが、やはり CIN を格納した VI で、CFITSIO を呼び出して、それぞれ FITS のデータから LabVIEW の配列への読み出しと、LabVIEW の配列から FITS データへの書き込みを行っている。

フローチャートを後に示すが、設計されているすべての VI はこれらの組合せで CCD を駆動させている。LabVIEW 側のブロックダイアグラムはこれらを CCD を駆動させたい順番に並べれば簡単に作ることができるが、ロードする DSP 側のプログラムは、これらのサブ VI が並んだ順番のとおりに PC と通信ができるように書かなければならない。

3　フローチャート

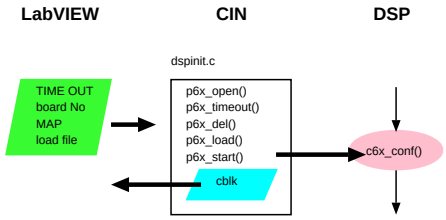

図 3: DSP の初期化

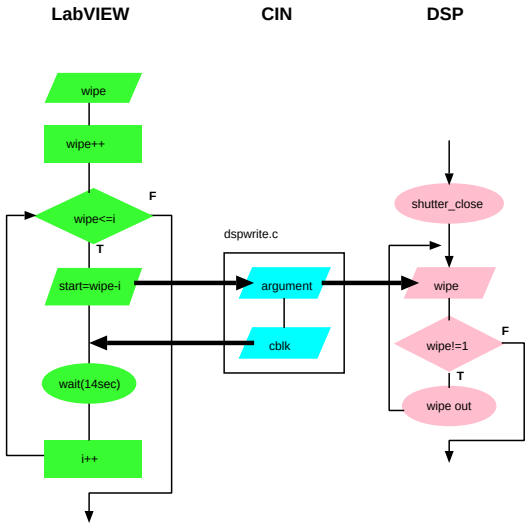

図 4: CCD の初期化 (wipe)

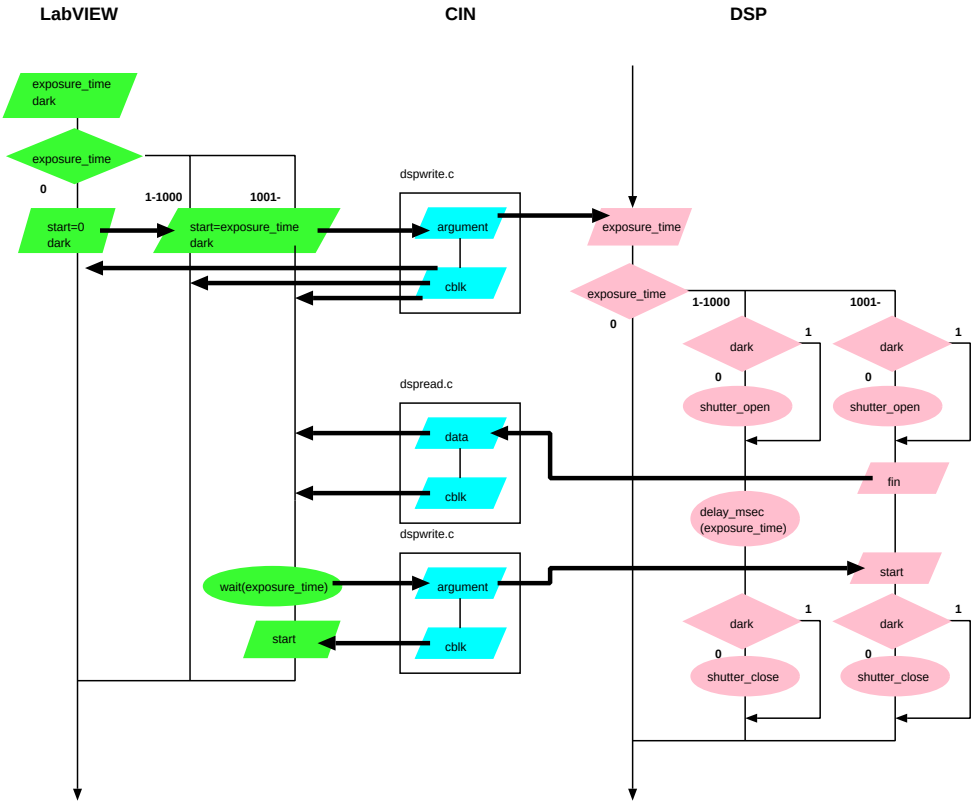
LabVIEW
CIN
DSP
exposure_time
dark
exposure_time
0
1-1000
1001-
start=0
dark
start=exposure_time
dark
dspwrite.c
argument
cblk
exposure_time
exposure_time
0
1-1000
1001-
dark
1
0
shutter_open
dark
1
0
shutter_open
dspread.c
data
cblk
fin
delay_msec
(exposure_time)
dspwrite.c
wait(exposure_time)
argument
cblk
start
start
dark
1
0
shutter_close
dark
1
0
shutter_close

図 5: 露光

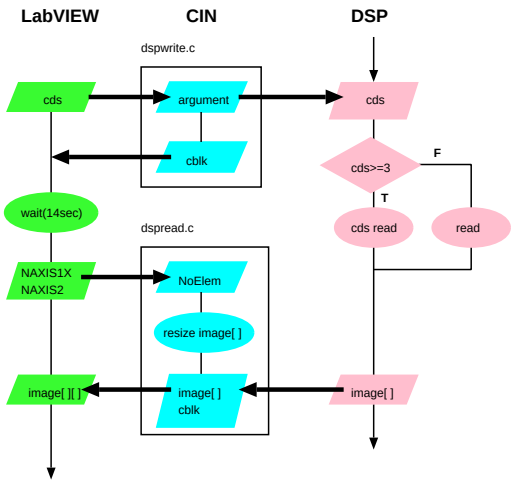
LabVIEW
CIN
DSP
dspwrite.c
cds
argument
cblk
cds
cds>=3
F
T
wait(14sec)
cds read
read
dspread.c
NAXIS1X
NAXIS2
NoElem
resize image[ ]
image[ ][ ]
image[ ]
cblk
image[ ]

図 6: 読み出し