HICALI の LabVIEW による制御 No.1

吉川智裕

2003 年 1 月 6 日

1 HICALI の制御

HICALI は、DSP(Degital Signal Processor) を介して PC との通信を行っている。DSP は、PC とは独立なプロセッサで、PC からプログラムをロードし、それに従って PC と通信し、HICALI に必要な信号を送る。すなわち、DSP 側と PC 側でそれぞれ最低でも 1 つずつのプログラムを必要とする。それらのプログラムは C で記述され、メーカが用意した関数を呼び出して、お互いに交信を行う。すでに、テキストベースの制御ではテスト用のプログラムは完成して動作も確認されており、今回はそのプログラムを LabVIEW から使えるようにして、GUI 的に HICALI の制御を行うことを目指した。

2 LabVIEW でプログラムを動かす

LabVIEW は、計測用の GUI プログラムを作るためのツールで、フロントパネルとブロックダイアグラムから構成される。ユーザはフロントパネルに配置された制御器および表示器を使って、仮想的な計測装置を扱うかのように GUI 的に計測機器を操作できる。プログラマはブロックダイアグラム上で、関数や変数の役割をするブロック同士をデータが流れるワイヤで組み合わせたり、制御文の役割をするストラクチャで囲んだりしてプログラムを組む。LabVIEW 上で、C で書いたプログラムを動かすには、以下のような手段がある。

- Sytem Exec.vi
- Call Library 関数
- Code Interface Node(CIN)

2.1 System Exec.vi

LabVIEW からテキストベースのプログラムを動かすための最も簡単な手段である。LabVIEW で最初から用意されているサブ VI で、Function Palette の Communication にある。入力端子にプログラムを呼び出すためのコマンドなどをつなぐと、出力端子からそのプログラムの標準出力、警告出力などが出力される。ブロックダイアグラムをみるとわかるが、その実体は下に述べる CIN であり、一度、呼び出すと、プログラムが終了するまで割り込みや中断はできない。また、プログラムが終了した後で標準出力などが出力されるので、対話的なプログラムを使うことは不可能である。

2.2 Call Library 関数

LabVIEW から共有ライブラリを呼び出すためのノード。サイズが可変な入力・出力端子を持ち、それを用いて関数に引数や返り値を受渡しする。自分で作成した共有ライブラリを呼び出すこともちろん可能である。基本的には、通常、C で使っている変数の型などをそのまま扱うことができる。

2.3 Code Interface Node(CIN)

LabVIEW から C で記述したプログラムを呼び出すためのノード。Call Library 関数と同様、サイズが可変な入力・出力端子を持つ。まず、C でプログラムを記述し、それを CIN で利用する実行形式にコンパイルする。コードのコンパイルには gcc もサポートされており、その時に使用する makefile は、lvmkmf(LabVIEW Make Makefile) というツールが自動的に生成してくれる。それを、ブロックダイアグラムに配置された CIN で load して実行する。

CIN で実行するプログラムには、LabVIEW で決められている変数型、関数を使うことができる。LabVIEW のブロックダイアグラムとデータを受渡しするときには必ずこの変数型を使用しなければならない。C では int 型のサイズは決まっていないが、この変数型では、例えば、32 ビット int は int32、16 ビット int は int16 などというように、ビット幅が型名に明記されている。そのため環境に依存しないコーディングが可能である。

前述のように、CIN は中断や割り込みができない。CIN でプログラムが止まると、LabVIEW 自体を強制終了しなければならないようである。そのため、CIN に時間のかかる処理を行わせるのは不向きと思われる。

3 LabVIEW 使用の検討

HICALI を LabVIEW から制御することの利点は、以下にあると考えられる。

1. GUI から制御するため、操作が直感的にもわかりやすい
2. ブロックダイアグラムを使用したプログラムで、機能の追加、変更が容易
3. LabVIEW がもつ、豊富なデータ解析 VI をブロックダイアグラムの接続だけで使うことができる

以上の LabVIEW から制御する利点を生かし、かつ、既成のプログラムをなるべく利用することを考えた結果、CIN を採用することになった。すでにテキストベースのプログラムができているので、当初は System Exec.vi を用いてそのプログラムを呼び出すことを考えたが、プログラム実行中の中断ができないため却下した。天体の撮像中、例えば雲が通って露出を中断しなければならないとき、System Exec.vi ではプログラムが終了するまで待たなければならないためである。一方、CIN を使用すれば、長時間待つ部分や、複数回繰り返す部分は LabVIEW 側で処理し、HICALI の制御を行うときに適宜、CIN を使って関数を呼び出せば、CIN の実行が終わるのを長時間待つ必要が無くなるのである。Call Library 関数を使っても同じようなことは実現できる。このときは、変数型は LabVIEW が定義するものを使う必要もなくなるが、今回、コントロールブロックを扱う際に、配列のサイズ変更などで LabVIEW で用意されている関数を使うことになり、結局、そのためにはやはり LabVIEW が用意する変数型を使わなければならず、あまり Call Library 関数を使用する利点がない。Call Library 関数は、既存の共有ライブラリを使うときに有効な方法といえよう。

4　LabVIEW からの HICALI の制御

今回作成した VI は、以下のような流れで HICALI を制御する。

1. DSP の初期化
2. CCD array の準備 (wipe)
3. 露出
4. 読み出し・出力
5. DSP を閉じる

DSP 側には、初期化の際に一つプログラムをロードし、以後、それを使ってプログラムの終わりまで PC と通信する。LabVIEW はそれぞれの段階で、1 回または複数回 CIN を呼び出して DSP と通信している。LabVIEW から呼び出す CIN と CIN の間では、コントロールブロック (CTLBLK) という構造体を渡して、DSP ボードの初期化状態などを伝える。コントロールブロックは DSP 制御用プログラムのライブラリで型宣言されている型で、プログラムの中で DSP を制御する関数を呼び出すときは、コントロールブロックへのポインタを渡して関数に DSP ボードの初期化状態などを伝えている。LabVIEW では、ポインタを渡すことができないので、CIN のプログラムの最初と最後で値に戻して渡しているが、基本的には、普通のプログラムから DSP を呼ぶのと同じ要領である。

このような設計で、撮像用の VI を作成した。基本的にはこの設計で問題なかったが、CIN のプログラムにコントロールブロックを渡すところが難しかった。コントロールブロック内には、配列と文字列が含まれており、それらの大きさは、LabVIEW が用意する関数 (NumericArrayResize(), SetCINArraySize()) などを使ってプログラムの中で領域を確保してやらなければならない。マニュアル (Using External Code in LabVIEW) をよく理解して使わなければならなかった。

5　今後の開発

とりあえず、LabVIEW を用いて GUI 的に HICALI を動かす VI を作ることができたが、まだ、実用にはいくつか問題がある。今後、解決していく問題を以下に挙げる。

- 撮像がどの段階にあるか、わかるようにする
- データの出力を、FITS などで保存できるようにする
- フォーカス用、導入用など、撮像目的以外の VI も作成する

現在の VI の動作実験では、PC が出力する信号をオシロスコープで見て確認しているが、実際の観測の際にはディスプレイを見て、CCD の状態がわかるようにしなければならない。現在、ローカル変数を用いて、一つの段階が終わるごとにローカル変数の中身を書き換えていく方針でこれを実現しようとしているが、今のところ成功していない。

今回、作った VI は、配列でデータを出力してそれを強度分布としてフロントパネルに表示している。これは、配列でデータを出力しておくことによって、LabVIEW が用意している解析用の VI (2 次元フーリエ変換をする VI など) を利用して、実験データを解析できるようにするためである。一方で、IRAF などの強力な画像処理ソフトでデータを解析するには、データを FITS 形式に変換してやらなければならない。CIN を使っ

て DSP 用の関数を呼び出したのと同じ要領で、CIN から CFITSIO を呼び出してやれば、FITS 形式で保存することも可能なはずである。

その他、天体の導入やフォーカスのときに使うための、高速で明るいビニング画像を自動的に連続で入れ換えるためのプログラムも必要である。これらも、順次、作成する予定である。