Camac の使い方とデータ収集系について

新倉 潤

2002 年 12 月 9 日

目次

1 ドライバーのインストールと nfa 2
1.1 ドライバーのインストール 2
1.2 nfa とは? 3

2 ADC の使い方 5
2.1 Peak Hold ADC 5
2.2 Charge ADC 6

3 Camac Interrupt 8
3.1 割り込みルーチンの記述と書式 8
3.2 interrupt を考慮した回路の改良 10
3.3 回路系でモジュールをクリアする 11

4 PAW の使い方と HBOOK を使ったヒストグラム 12

5 偏極ビームを使う 13
5.1 Polarimeter Control Unit と偏極実験におけるデータ収集の概略 13
5.2 ハタの状態によって読み込む ADC を変えてみる 14

A 使用するモジュール 15

1　ドライバーのインストールと nfa

1.1　ドライバーのインストール

CAMAC のクレートコントローラーのドライバーは東京大学理学部の岡村さんの提供しているものを使います。クレートコントローラーは東陽テクニカ製 CAMAC クレートコントローラー CC/7000、AT 互換機用インターフェースは CC/7000AT-IF(ISA バス) を使います。

すでに AT 互換機用インターフェース CC/7000AT-IF(ISA バス) は chocobo に取り付けてあります。IRQ は 11、IO PORT は 0x310 のデフォルトのままで問題なく動くようなのでそのままで使うようにしてください。(一応みんなで同じものを使うことを決めておかないとユーザーごとの書き換えが面倒なので。)

ドライバーは各ユーザーごと使いやすいようにプログラムを書き換える必要があるので、使用する際はそれぞれ自分のアカウントでログインして適当なディレクトリにドライバーをコピーして使ってください。配布されているドライバーは、chocobo の/home/P42002/にある camac.tar.gz です。root になってここからコピーして自分のディレクトリ内に展開して使います。

解凍するときに root のままだとすべてのファイルの所有者が root になってしまうので、コピーしたファイルのグループと所有者をそれぞれのユーザーに変更してください。

```
% chgrp users camac.tar.gz
% chown **** camac.tar.gz
```

****はそれぞれのユーザー名です。これ以降の作業はすべて平ユーザーでできます。

まずファイルを解凍します。

```
% tar zxvf camac.tar.gz
```

すると同じディレクトリ内に camac というディレクトリができているはずなのでそこに移動してください。

まず中にある Makefile を適当なエディタで開いて編集します。おそらくデフォルトではハードウェアの設定は以下のようになっているはずなので一応確認して違っていたら以下のように変更してください。

```
# Modify the folowing values according to the hardware setting.
#--
DEVICE = dc
MAJOR = 61
IOPORT = 0x310
IRQ = 11
DELIM = 0x8001
#--
```

また、このまま使うとモジュールを組み込むときにエラーを起こすので install と uninstall および ANALIB の部分を以下のように書き換えてください。

```
install: $(OBJ)
                                                    # delete /sbin/rmmod $(OBJ)
```

```
        /sbin/insmod $(OBJ)
        cat /proc/devices

uninstall:
        /sbin/rmmod  $(DEVICE)                  # over write $(OBJ) を$(DEVICE)
        cat /proc/devices

#--              Sample of Data AcQuisition System               --

 ANASUB = event1.o
 ANALIB = -L/cern/pro/lib -lpacklib \
          /usr/lib/gcc-lib/i586-pc-linux-gnulibc1/egcs-2.90.29/libf2c.a -lm
                                                  # change directry of f2c compiler
     FC = g77
```

変更が終わったらとりあえずデバイスドライバをコンパイルしてカーネルに組み込んでみます。以下のコマンドを打ってください。

```
% make
% make install
% make clean
```

どこでもエラーが起こらなかったらデバイスドライバーは正しくインストールされています。確認するには

```
% dmesg
```

と打って出てきた表示の最後に

```
... ... ... <<< dc >>> ... ... ... installed.
 dc: Buffer size is 131072 bytes
```

の二行があればデバイスドライバは正しくインストールされています。テスト用に `test_led.c` というプログラムが用意されているので、ファイルに Swightch Registor のステーションナンバー (いつもは 22 番になっている) を書き加えてコンパイルして実行すると、Swightch Registor のランプが上下に点灯するはずです。

```
% gcc -o test_led test_led.c
% ./test_led
```

1.2 nfa とは?

PC から Camac に命令を出す場合には基本的には nfa の 3 つのパラメータを指定します。n は Camac Bin におけるモジュールのステーションナンバー、f は命令の内容、a はモジュール内のサブアドレスです。モ

ジュールによって Camac Action に対する応答が違うので詳しくはモジュールのマニュアルを見てください。nfa を実行する一番簡単なほ方法は用意されている `nfa.c` というプログラムをコンパイルして実行します。例えば Swich Registor の値を読み込むにはコンソールに以下のように打ち込みます。

```
% gcc -o nfa nfa.c
% ./nfa 22 0 1
```

とくによく使うコマンドや Start/Stop に使うコマンドなどは tcl/tk などで GNU 化しておくと便利です。chocobo の/home/niikura/camac/camac/nfa.tcl においておくので自由に使ってみてください。

2　ADC の使い方

ここでは ADC を使って送られてくる信号のデータを PC で処理する回路を作ります。使用するモジュールの簡単な動作ついては付録 A に書いておいたので参照してください。また、いろいろな人が同じ Camac Bin を使うので、基本的にモジュールの入っているステーションはなるべく変更しないようにしましょう。

ADC には主に 2 種類あり、Slow 系の Peak Hold ADC と Fast 系の Charge ADC があります。今回の実験では主に Fast 系の電荷積分型の ADC を使用しますが、ここでは両方の ADC の使い方について解説します。

2.1　Peak Hold ADC

Slow 系の ADC はテクノランドの C-TM 405 8CH PH ADC です。この ADC は Gate が開いている間のパルスのピークをデジタル信号にして Camac のバスラインに出力します。また、データを出力する準備ができた時点で LAM を発生して PC に割り込みを要求することもできます。割り込みに関しては 3 節で詳しく解説します。

ここではあまりそれぞれの作業の詳細には立ち入らず、とりあえず Pulser からのデータを取ることを目標にします。何か問題が起こったときは以下の節を参考にして対処してください。まず、図 1 のような回路を組みます。

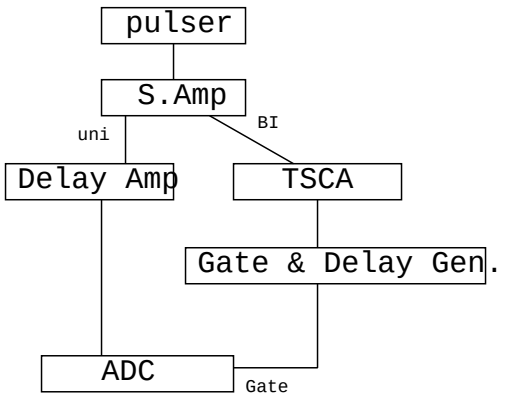

図 1: PH ADC 回路図

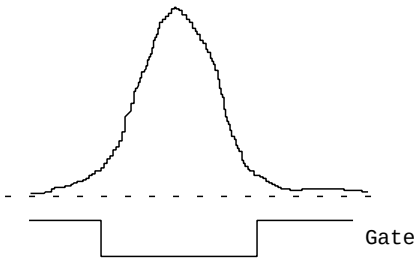

図 2: Gate と入力信号の関係

最終的には入力信号と Gate 信号との時間的な関係が図 2 のようになるように、オシロスコープで見ながら各モジュールのパラメータを調節してください。ADC の入力は最大で 2V までなので、それを超えないように Pulser や S.Amp の出力レベルを調整します。

回路の調整が終わったら PC のほうの設定をします。それぞれの設定の詳しい内容についてはあとのほうで説明します。まず、`int_phot1.c` を適当なエディタで編集します。なかに ADC の場所を定義しているところがあるので、そこに ADC の入っているステーションナンバーを書きます。いつもはステーションナンバー 14 に入っています。

```
#define ADC         14              /* define the station number of ADC
```

今はまだ OUTREG の定義はコメントアウトしたままでいいです。すでに `int_phot1.c` は `dc_int.c` にシンボリックリンクが張られてるいるはずなので、ここで

```
% make uninstall
% make install
% make clean
```

とすれば割り込みプログラムがカーネルに組み込まれます。
次に入ってきたデータの処理をするプログラムをコンパイルします。コンソールで

```
% make DAQ
```

を実行してください。これで、データ処理に使う commander.c、recorder.c、analyzer.c がコンパイルされます。
データ収集は rei という tcl/tk で書かれたスクリプトを使います。rei のはじめに tk のインタープリンタの場所を書き換える必要があるのですが、

```
% make wish
```

とすれば自動的に書き換えられます。
これで準備は完了です。rei を実行してデータを取ってみてください。
取れたデータは PAW を使って見ます。ここではあまり何も考えずに

```
% paw

 ********************************************************
 *                                                      *
 *            W E L C O M E    to   P A W               *
 *                                                      *
 *       Version 2.08/14      21 January 1997           *
 *                                                      *
 ********************************************************
 Workstation type (?=HELP) <CR>=1 :
 Version 1.23/07 of HIGZ started

PAW> hist/hio/global sample
PAW> hist/plot 1
```

とすれば入力パルスの高さのスペクトルがグラフで表示されるはずです。

2.2 Charge ADC

Fast 系の ADC は LeCroy の 2249W を使います。本来このモジュールは NaI のような信号のテールが長いパルスのデータを扱うためのものなのですが、ここでは入力信号はプラスチックシンチで宇宙線などのノイズを拾ってそれを ADC で拾うことにします。図 3 のように回路を組んでください。

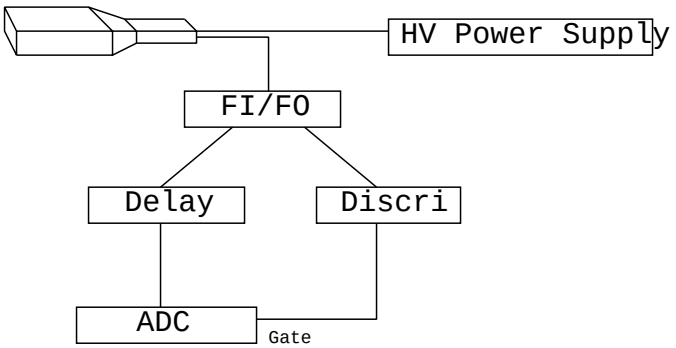

図 3: Charge ADC 回路図

今 P4 部屋にあるプラスチックシンチについている Photo Multiplier は、浜松フォトニクスの R329-01 です。最大加電圧は-2800V ですが、一般的には-1500V 程度で使用することになっています。電源供給は NIM Bin にある HV Power Supply の 1ch につけてあるので、電圧を-1500V 程度までかけてみてください。出力する信号をオシロで見ると 20 ~ 30mV 程度のパルスが出力しているのがわかるはずです。

また、この ADC は Gate 信号が 30nsec 以上の幅がないと Gate として認識しないので、それ以上になっていることを確認してください。そのほかの ADC の使用法は Slow 系のものとほとんど同じなので、割り込みルーチン `int_phot1.c` の ADC のステーションナンバーだけを変更してデータを取ってみてください。ノイズなのでエネルギーの低いところにピークが見えるはずです。

3 Camac Interrupt

3.1 割り込みルーチンの記述と書式

3.1.1 Interrupt の定義方法

Camac はモジュールによってはデータが来たのち、ある時点で LAM(Look-At-Me) を発生して PC に知らせることができます。一方 PC のほうで、LAM を確認したら優先的にデータを取り込むように割り込みの設定をすることで、次々に来るデータを効率的に処理できます。

割り込みに関する設定はドライバーのインストール時に dc_int.c というファイルを読み込んで、割り込みをかける条件や、割り込みがかかったときにする動作などを設定します。しかし、必要におうじて割り込みプログラムを書き換える必要があるので、他に割り込みの定義ファイルを作って dc_int.c にシンボリックリンクを張るという方法をとると便利です。

例えば int_hogehoge.c という割り込みルーチンを用意したとき、それにシンボリックリンクを張るには

```
% ln -s int_hogehoge.c dc_int.c
```

とすれば dc_int.c にシンボリックリンクが張れます。シンボリックリンクを張り直すときは、一度リンク先の dc_int.c を消してから、新たにリンクを張ってください。

3.1.2 割り込みルーチンの書き方

サンプルとしてあらかじめ用意されている int_phot1.c を例にとって説明します。(付録　参照) はじめの 14 行は LAM を監視するためのプログラムなので書き換える必要はありません。その下の部分を必要に応じて書き換えます。

まず、測定中に Camac Action を起こす対象となるモジュールのステーションナンバーを定義します。

```
#define OUTREG    10
#define ADC       14
```

次に LAM を見るモジュールを決めます。LAM_N、LAM_A という変数に LAM を見たいモジュールの N、A を代入すれば、それが割り込みを指示するモジュールとして定義されます。ここでは ADC の LAM を見ているようです。

```
#define LAM_N    ADC
#define LAM_A    0
```

このドライバーでは、rei の起動時に一度 Camac バスにクリア信号を送ってすべてのモジュールをクリアした後に、ここで定義したモジュールを LAM enable にします。そして、割り込み自体の実行はクレートコントローラーに送られてきた LAM を、その発信もとのモジュールが何であるかにかかわらずすべて受け付けるようになっています。つまり、LAM を enable しない限りモジュールは LAM を出さないはず、という発想の元に設計されています。ところが、LAM disable という機能を持たないモジュールも存在しそれが Camac Bin のなかにあると、ここで定義した、しないにかかわらず LAM を受け付けて割り込みを実行してしまうの

で、注意が必要です。ドライバーの書き方によっては、一度 LAM の発信もとのモジュールを確認してからそれが必要なものならば割り込みを実行するように書くこともできるので、時間があったらドライバーの書き換えに挑戦してみてもいいのではないかと村上先生はおっしゃってました。

また、この LAM の定義の仕方を見ると、モジュールのサブアドレスまで指定できるように見えますが、たいていのモジュールはサブアドレスごとの LAM enable/disable をすることができるようにはなっていないので、LAM を 0 にしておけばすべてのサブアドレスの LAM が見れるようになります。たとえばデータの処理に時間がかかる[*1]ADC や TAC のようなモジュールでは、Gate が閉じたあとデータの処理が終了した時点で LAM を発生しますが、データの入力が複数ある場合はすべてのサブアドレスに入ってきたデータの処理が終わった時点で LAM を発生するのが一般的なようです。詳しくはそれぞれのモジュールのマニュアルを見てください。

LAM を見るモジュールを定義した後で、PC が LAM を受け付け、割り込みを実行した際にする動作を指示します。まず読み込むワード数を記録する変数 dc_wp を用意します。

```
    wp0=dc_wp++;                    /* save current pointer for event-length */
```

ここは特に書き換える必要はないです。

次に割り込みの最中に読み込むデータを指定します。割り込み中で nfa と同じような動作をする関数は camacCcycle と camacCcycle です。camacCcycle は 24bit のデータを camacCcycle は 16bit のデータを扱います。モジュールのデータの量はマニュアルを見てください。例えばテクノランドの PH ADC の場合は 14bit 出力なので camacCcycle2 を使います。

```
/* Read ADC         N  A F  User should modify below line.*/
      camacCcycle2(ADC,0,0,&dc_evbuff[(dc_wp++)%EV_MAX],&stat);
```

ここでは単に ADC の値を読みに行って dc_evbuff という変数に入れています。dc_evbuff はあらかじめ用意した PC のバッファ領域のことです。拾ってきた値を代入する変数はポインタで指定することに注意してください。また、その際に読み込んだワード数をカウントするために、はじめに用意しておいた dc_wp という変数の値を一つ増やしています。

データの読み込みが終わったらそのイベントに関するデータ領域が終わったことを示す DELIMITER を最後に書き込み、データのはじめにそのイベントに関するデータが何ワードあるかをはじめに書き込みます。DELIM は 0x8001 です。これはあとでデータを処理して解析するときに使います。

```
      dc_evbuff[(dc_wp++)%EV_MAX]=DELIM;              /* store delimiter */
      dc_evbuff[wp0%EV_MAX]=(short)((dc_wp-wp0)<<1);  /* store event length */
```

最後にモジュールをクリアして、Output Registor の下位 8bit にパルスを出力しています。Output Registor の出力は PC の処理が終わったことを示すので、処理中には新たにデータが来ても Gate が開かないようにする VETO 回路[*2]を組んでいる場合や、モジュールのクリアをハード的にする場合[*3]に使います。

```
    camacCcycle2(ADC,0,9,0,&stat);                  /* clear module */
```

[*1] これを conversion time とよびます。
[*2] 3.2.1 参照。
[*3] 3.3 参照

```
#if defined(OUTREG)
    dc_outreg |= 0x00ff;
    camacCcycle2(OUTREG,0,16,&dc_outreg ,&stat);
    dc_outreg &= 0xff00;
    camacCcycle2(OUTREG,0,16,&dc_outreg ,&stat);
#endif
```

Linux システムでは割り込みによるほかのタスクの遅延をなるべく最小に防ぐためにも、また、dead time[*4]を最小に抑えるためにも割り込み処理をできるだけ短時間で終了させたいという要請があります。このクレートコントローラーを chocobo の CPU 性能で使用した場合、1 イベントの処理に必要となる時間は、

約 $20 + (10 \times$ 読み込みワード数$)\mu sec$

で算出できます。そのため OFF LINE でも出来るような解析はここではせずに、ひたすらデータを溜め込むことだけを考えたルーチンを書くようにしてください。

3.2 interrupt を考慮した回路の改良

3.2.1 VETO 回路

不定期に来るデータを処理する場合、モジュールや PC での処理中に次のデータが来て新たな割り込みがかかるとそれまでのデータの処理が止まってしまいします。そのために、データの処理中は新たなデータが来ても inerrupt がかからないように回路を組んでやる必要があります。つまり、モジュールの Gate が開かないようにすればいいので、Gate に入力する信号に図 4 のような回路をかませます。

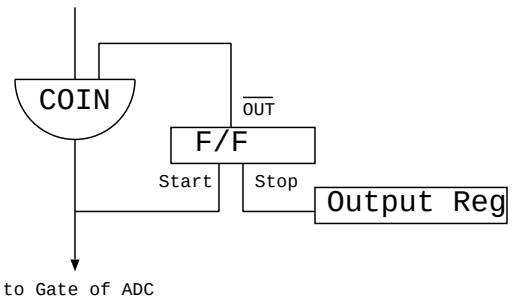

図 4: VETO 回路

データ処理の終了の合図は割り込みルーチンで定義してある Output Registor の下位 8bit の出力を使います。この状態でデータを取るとき、rei でデータ収集を Start したときに Gate が開かないように F/F が Latch された状態にあるので、データ収集を Start したあとで Latch を解除するために Gate & Delay Gen. の stop ボタンを押してやる必要があります。スタートのためにはじめに Output Registor の下位 8bit にパルスを送るようなプログラムをあらかじめ作っておくと便利です。[*5]

[*4] 3.2.1 参照

[*5] /home/niikura/camac/camac/start.tcl においてあります。自由に使ってみてください。

この回路を使うと、データが来ていても測定できない時間が存在してしまいます。実際に来ているデータ数に対する測定したデータ数のことを live time と呼び、逆に測定できない時間のことを dead time と呼びます。データを解析して反応の断面積を計算するときにはこの live time を考慮することを忘れないでください。

3.2.2 Interrupt Registor を使う

今までは割り込みを ADC でかけていましたが、他にも LAM を発生するモジュールはあります。本実験では処理に時間がかかるモジュールとして ADC のほかに TAC も使います。データの種類によって conversion time が異なるようなモジュールを LAM に使っていると LAM 発生時に両方のデータ処理が終了している保証がないので、一般的には他のモジュールで統一して LAM を出すほうが便利です。

ここでは ADC のデータを Interrupt Registor で LAM をかけて取り込むような処理ルーチンを書いて実際にデータを取り込んでみます。使用する処理ルーチンは chocobo の/home/niikura/camac/camac/int_test1.c です。割り込み中の処理の中で Coinsidence Registor の値もとっていますがあまり気にしないでください。Coinsidence Registor の使い方については 5.1 で解説します。

図 5 のように Gate 信号を FI/FO で分岐してひとつは Gate & Delay Gen. で conversion time 分だけ遅らせて Interrupt Registor に入力するように回路を組んでください。

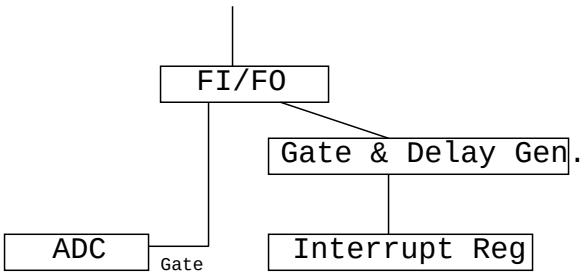

図 5: Interrupt Registor への入力信号

ADC の conversion time はテクノランドの PH ADC で最大 25μsec、LeCroy の Charge ADC で最大 106μsec です。オシロで Gate & Delay Gen. の入出力を見ながら Delay time が conversion time より大きくなるように調節してください。

3.3 回路系でモジュールをクリアする

前にも書いたように、割り込み処理はなるべく短時間で終了させたいという要求があります。たくさんのモジュールがある場合にはいちいち F=9 のファンクションを実行してそれぞれのモジュールをクリアしていくとそれだけで非常に時間がかかります。また、割り込みルーチンで使用する関数の中には `camacCc()` というものがあって、Camac のバスにクリア信号を出してすべてのモジュールをクリアするというものもありますが、この関数を使ってしまうと、Scaler や Output Registor の値がクリアされてしまい、これらのモジュールを使用している場合には不適当です。そのため、処理の最後に Output Registor の下位 8bit にパルスを出力することを利用して、その信号を各モジュールの Clear の入力に入れることで Camac Action の数を減らすことが出来ます。

4　PAW の使い方と HBOOK を使ったヒストグラム

鋭意作成中

5　偏極ビームを使う

5.1　Polarimeter Control Unit と偏極実験におけるデータ収集の概略

今回の実験では偏極ビームを使用します。編曲したビームでも 100% 完全にスピン状態のそろったものがくるわけではなく、実験中にある周期でポラリメータを使って偏極度を測定する必要があります。

データ収集系では、まずイオン源に対してスピン状態を指定する信号を送る必要があります。これは Gate & Delay Gen. を図 6 のように使うことで上下の信号を作ることが出来ます。

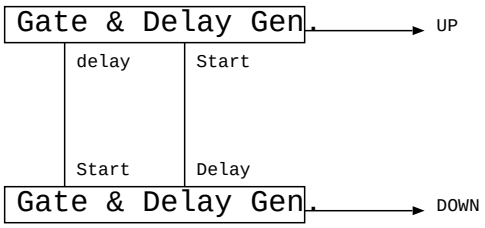

図 6: Spin を指定する信号の作り方

今回の実験でポラリメーターを操作するのに、現在理研におられる市原さんの製作した Polarimeter Control Unit を使用します。使用方法についての詳細はマニュアルが用意されているのでそれを見てください。

図 6 のようにスピン状態を指定する信号を作ってそれを Polarimeter Control Unit 入力すると、スピンの UP/DOWN の回数をカウントして指定した回数に応じてハタの IN/OUT をします。ハタの IN/OUT はポラリメータのハタ操作の入力端子に 24V の電圧をかけることで操作でき、Polarimeter Control Unit の背面の出力からは指定したハタの状態に応じて 24V の電圧が出力されるようになっています。また、前面の出力端子からはハタの状態とスピン状態が TTL レベルの論理信号で出力されます。この出力信号を利用して計測系の回路を設計します。出力レベルが TTL なので Level Adoptor を使って NIM の信号に変換してから使ってください。

データ収集はおおよそ以下のような流れで行います。

LAM は前述したのように Interrupt Registor から読み込んで割り込みをかけます。割り込み処理の段階では、まず Polarimeter Control Unit から出力されるハタの状態を Coinsidence Registor に入力してそれを読み込みます。ポラリメーターからのデータと Focal Plane Detector からのデータはかける Gate が違うので、別々に ADC を用意しておき Coinsidence Registor から読み込んだハタの状態に応じて読みに行く ADC を変えます。Coinsidence Registor にはそのほかにスピンの状態も入力しておきますが、収集段階ではスピンの状態は区別せず他のデータと一緒に読み込んでおき、解析のとき、ヒストグラムをする段階でスピン状態を読み込んで、その状態に応じて別々にヒストグラムします。

ポラリメータからは左右につけられた NaI シンチレーターからきます。1 つのイベントに関しては左右どちらかの NaI からしかデータは来ないので、FI/FO を使って OR を取って Gate 信号にします。一方で、データを受け付けたほうの NaI を識別する信号を Coinsidence Registor に入力しておき、割り込み中の処理の段階でこの情報から読み込む ADC のサブアドレスを変えます。

5.2 ハタの状態によって読み込む ADC を変えてみる

まず、ハタの状態を Polarimeter Control Unit からの出力で読み込んで、読みに行く ADC を変えるような割り込みルーチンを書いて、実際にデータを取ってみることにします。ADC は Slow 系と Fast 系の ADC を使いそれぞれハタの IN/OUT に相当することにします。次の図 7 ような回路を組んでください。

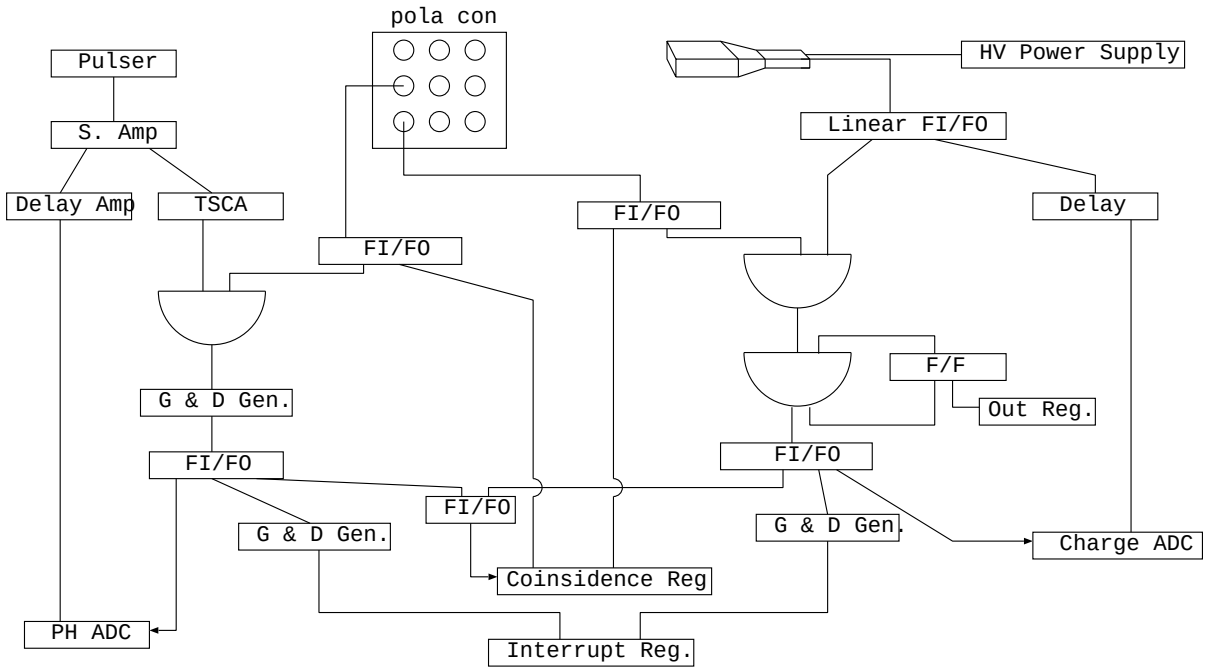

図 7: ハタの状態によるデータの振り分け 回路図

割り込みルーチンは付録 のように書きます。[*6]

ここではまずはじめに short int として hata という変数を用意しています。割り込みがかかったら、まず Coinsidence Registor の値を読みこの変数に代入します。camacCcycle2 で読み込んだデータを渡す変数は、ポインタで指定することに注意してください。Coinsidence Registor に対しては F=2 の read and clear を使って読み込んだ後にクリアしてしまいます。その後ハタの状態によって読み込む ADC を変えて、そのあとに hata として読み込んでいるデータも一緒にバッファに代入します。3 節でも書いたようにクリアは回路のほうですることにしてます。

また、拾ってきたデータをヒストグラムする際にもハタの状態によって別々のグラフにしたいので、そのための HBOOK の処理ルーチンを書きます。使った Fortran のルーチンは/home/niikura/camac/camac/eve_test.f です。

これで準備は整ったはずです。実際にデータを取ってみてください。

[*6] /home/niikura/camac/camac/int_ test.c にもあります。

付録A　使用するモジュール

使用するモジュールの簡単な説明を載せておきます。詳しい内容についてはモジュールカタログをコピーしてファイルしているので自分で調べてください。

NIM

- SWICH REGISTOR (豊伸電子 C003)
  A=0 が LED によるレジスタ、A=1 にスイッチレジスタを持っています。それぞれのデータは 24bit で Camac による読み出しや書き込みが出来ます。主に Camac Bin のステーションのチェックに使います。LAM スイッチも付いていて、ボタンを押すことで LAM を出します。

- PULSER (ORTEC 480)
  文字通りパルス信号を出すモジュール。パルスは立ち上がり時間が 61nsec、立ち下がり時間が 400μsec、信号の周期が 10msec ほどの信号を発生させます。

- AMPLIFIER (EG & G ORTEC 571)
  入力信号を増幅して、積分した信号 (UNI OUT) と微分した信号 (BI OUT) をそれぞれ出力します。パルスの Shaping Time がだいたい μsec のオーダーなので Slow Amp と呼んでいます。当然世の中には Fast Amp と呼ばれるものも存在してプラスチックシンチレーターのような立ち上がりの早い信号をそのまま増幅して Shaping Time が nsec オーダーの出力を出すようなアンプも存在します。RCNP にあったので、現在 1 台 W 実験室においてあります。
  一般に BI OUT の信号は GATE に使うので UNI OUT の信号にあとで DELAY をかける必要があるのですが、適当にある程度内部で DELAY をかけることができます。そのため、単純な回路では DELAY AMPLIFIER を使う必要はないです。

- DELAY AMPLIFIER (EG & G ORTEC 427A)
  入力信号に Delay をかけて出力します。μsec の Delay をかけると入力信号が減衰してしまうのでそれを増幅して出力しています。入力信号は最大 11V までです。

- TIMING SINGLE CHANNEL ANALYSER (ORTEC 553)
  S.Amp からの BI OUT 信号からタイミング信号を NIM のロジック信号として出力します。

- GATE AND DELAY GENERARATOR (テクノランド N-TM 307) (海図製作所 KN1500)
  NIM の論理信号が入ったときから設定した時間だけ、Logic 1 の論理信号を出力します。これで GATE の幅を調節したり、出力する時間を入力信号に対して遅らせることも出来ます。
  また、出力の幅を設定するところに Latch モードというのがあり、Start 信号が入ると出力は Logic 1 で固定され、そのあと Stop に信号が入ると Logic 0 に戻ります。

- HEIGHT VOLTAGE POWER SUPPL (REPIC model RPH-031)

4ch の高圧電源です。各チャンネルの出力電圧は、0V から-3.0KV (2mA) まで可変することができ、背面の SHV コネクタより出力されます。電圧をかける PM の最大加電圧を調べてから電圧をかけるようにしてください。

- DISCRIMINATOR (LeCroy Model 4608C)
  入力信号が閾値 (threshold) よりも大きい場合に NIM の論理信号を出力します。閾値は-15mV から 1V までの範囲で、出力信号の幅は 4.5nsec から 100nsec の範囲で設定することが出来ます。

- CABLE DELAY
  信号はケーブルを伝播する際におおよそ 20cm で 1nsec の時間がかかります。それを利用してケーブルをつなげて Delay として使用しているものをモジュール化したもので、今 P4 部屋にあるものはおそらく誰かの自作でしょう。上のチャンネルの 2nsec の部分が壊れているので on にしないでください。構造は簡単なので暇な人が直してくれるとありがたいです。

- COINCIDENCE (Phillips Scientific model 755)
  入力論理信号が、COIN. LEVEL の設定した数だけ重なった時点で出力信号を出します。入力・出力論理信号は NIM 信号レベルです。入力信号は 4 つまで受け付け、出力信号の幅は 4nsec から 1μsec の範囲で設定して出力されるので、重なっている間はずっと信号が出続けるような AND 回路ではないので注意してください。

- FAN IN FAN OUT (linear: LeCroy 428F; logic: テクノランド N-RY 012)
  入力信号の OR を取って出力します。入力信号がパルスのものを扱う Linear FI/FO と、論理信号を扱う Logic FI/FO があります。単に信号を分岐させたいときにも使用します。

- LEVEL ADOPTOR (海津製作所 KN200)
  NIM 信号を TTL に、TTL を NIM に変換します。理由はよくわかりませんが、スイッチは COMPL. 似して使ってください。

Camac

- Pulse Hight sensitive ADC (テクノランド C-TM 405)
  Gate が開いているとき、入力信号のパルスの高さに比例したデジタル信号を出力します。入力は 2V までなので入力信号がそれを超えないように PULSER や S.Amp の出力調節をしてください。conversion time は最大で 25μsec、resolution は 14bit です。

- Charge ADC (LeCroy 2249W)
  Gate が開いているときの入力信号の電荷を積分してデジタル化し、10bit のデータとして出力します。model 2259W は特に Gate が最大で 10μsec とかなり長く設定できるので、信号のテールが長い CsI や NaI シンチレーターからのデータを扱うときに使います。最大入力電圧は-2V までで、conversion time は最大で 106μsec です。

- OUTPUT REGISTOR (海図製作所 KC3471)
  Camac Function F=11 で subaddress=0 に 16bit の書き込みができます。出力信号は NIM 信号レベルで出力されます。

- INTERRUPT REGISTOR (テクノランド C-TS 209)
  入力信号があるとすぐに LAM を出すモジュールです。全体で割り込み処理をしたいときに使います。また、入力があったサブアドレスは別々に記憶されているので、Camac バスラインから 8bit のデータとして読み込むことが出来ます。

- COINSIDENCE REGISTOR (海図製作所 KC3220)
  16ch の入力信号を記憶して Camac バスラインから読み込むことが出来ます。入力信号は Gate が入った時の入力状態を記憶します。

- SCALER (LeCroy 2551)
  入力信号の数をカウントします。入力は Camac バスラインから読み込みます。