UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

An Introduction to CAMAC

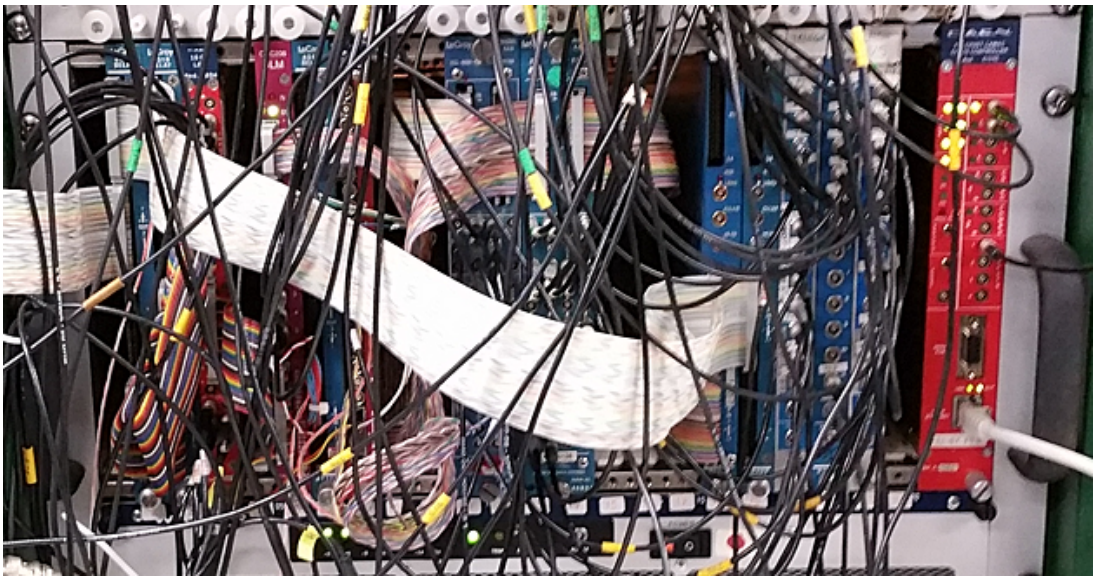
Alessandro Bravar
Alessandro.Bravar@unige.ch

Université de Genève
Section de Physique

Abstract

This note gives a basic introduction to the CAMAC data acquisition system. In our Lab we use a system consisting of one CAMAC crate interfaced to a PC by the CAEN C111 CAMAC crate controller. The communication between the controller and the PC (i.e. the data exchange) is based on the ethernet TCP/IP protocol. All high level CAMAC functions (the programming language is `C`) are implemented through the JENET Library, which has been specifically developed for the C111 controller. Several DAQ examples using different CAMAC modules are provided.

Before inserting or removing modules always turn off the CAMAC crate!



1 Introduction

Computer Automated Measurement And Control (CAMAC) is a modular system for data acquisition and control through a digital dataway connected to a data processing device such as a computer. For a readable introduction to CAMAC see for instance ch. 18 in Leo's textbook [1]. The system has been originally defined by the European Standards on Nuclear Electronics (ESONE) Committee in the late sixties. The current standard represents the joint specifications of the European ESONE and the US NIM Committees [2].

The CAMAC standard defines the mechanical, electrical, and logical specifications for the plug-in modules, crates, and the dataway. The dataway lines include digital data transfer lines, strobe signal lines, addressing lines, control lines, and power. Mechanically, a CAMAC crate consists of a crate with 25 stations with 86-way sockets on the backplane carrying a parallel bus. for data exchange and control. For the detailed pin allocation chart see Tables 18.3 and 18.4 in [1].

The crate can house a variety of plug-in modular instruments for a wide range of applications, like ADCs, TDCs, scalers, discriminators, logic modules, etc. to be interfaced to the dataway (the standardized backplane). The crate is controlled by a so called CAMAC crate controller, whose purpose is to issue CAMAC commands to the modules, check the status of the modules, and exchange data between the host computer and the CAMAC modules. The first 23 stations (the first station is numbered 1) can be filled with different plug-in modules, while the two rightmost stations (24 and 25) are reserved for the crate controller. Additions to a data acquisition and control system may be made by plugging in additional modules and making suitable software changes. Several crates can be connected on a data highway ending in a branch driver, which is interfaced directly to a data acquisition computer. A full scale system can consists of several parallel branches. Timing and protocol specifications permit up to 1 megaword/second transfers of 16 or 24-bit words. In reality, the data rate is limited by the crate controller, the data transfer protocol, and the host computer.

Although relatively old, the CAMAC system is still widely used in nuclear and particle physics experiments and in industry because of its easy handling and because a large amount of expensive hardware (modules) is still available and functional.

In our Lab we use a system composed of one CAMAC crate interfaced to a PC by means of the CAEN C111 crate controller [4]. This module allows the PC to control the CAMAC dataway via ethernet using the TPC/IP protocol for the data exchange. All basic operations on the CAMAC bus (initialization, data reading and writing, etc.) can be performed using the high level functions defined in the JENET Library [5], which has been derived from the ESONE standard.

2 CAMAC Commands

The CAMAC standard uses geographical addressing to identify a specific module and a *channel* or register within that module, i.e. the address depends on the module location and not the module itself. Within the dataway, modules are addressed by branch (**B**), crate (**C**), slot (**N**) and sub-address (**A**):

B - branch number;

C - crate number in the branch;

N - position of the module in the crate (N = 1 to 23);
there is a dedicated line on the dataway for each station;
A - sub-address (A = 0 to 15, 4 bits and 4 lines on the dataway).
In our implementation, which uses a single crate, we will need only **N** and **A**.

One instruction or CAMAC command is specified by an address and a function code **F** (i.e. **BCNAF**). A command is composed of signals on the station number line, the sub-address lines and the function lines. It is accompanied by a signal on the busy line. The command signals are maintained on the dataway for the duration of the operation. In response, the module will generate the *command accepted* signal (**X**) and act on the issued command. In a CAMAC cycle one 24-bits word can be transferred in parallel on the dataway bus between the controller and the selected module (24 lines on the dataway are reserved for reading and 24 lines for writing). Status bits (**Q** and **X**) indicate the success of the operation. Note that the terms *read* and *write* apply to the controller, not the module.

During a dataway operation the controller generates a command consisting of signals on individual station number lines to specify a module, signals on the sub-address bus lines to specify a sub-section of the module, and signals on the function bus lines to specify the operation to be performed. The command signals are accompanied by a signal on the busy bus line, which is available at all stations to indicate that a dataway operation is in progress.

Each module in a slot provides up to 32 *functions* (function codes 0 to 31, 5 bits and 5 lines on the dataway) and defines 16 sub-addresses (address 0 to 15, 4 bits and 4 lines on the dataway). These signals are decoded in the module to select one of the 32 functions and to select one of 16 registers. Not all functions are necessarily implemented in a specific module and the module might have less registers. The term register is used for an addressable data source or receiver and not necessarily a memory unit. The function codes allow the register in a module to be divided into two distinct sets (**Group 1** and **Group 2**), therefore 32 different channels in a single module can be specified and addressed. Within each module, a certain functionality can be replicated up to 16 times using sub-addresses.

To find out which functions are supported by a particular module (i.e. which actions can be performed on/by the module), consult the module's data sheet or manual. Here you will find also details on what the module does, how it works, how to use this module, what kind of input signals it can accept (i.e. signal levels), what kind of signals it generates, and what kind of control signals it requires (i.e. gate, veto, clear ...).

CAMAC Function Codes

Of these functions, F = 0 to 7 are read functions (**F = 00XXX**) and will transfer data from the addressed module to the controller; for instance
F = 0 read Group 1 register;
F = 1 read Group 2 register;
F = 2 read and clear Group 1 register;
F = 3 read complement of Group 1 register.

The functions F = 16 to 23 are write functions (**F = 10XXX**) and will transfer data from the controller to the addressed module; for instance

F = 16 overwrite Group 1 register;
F = 17 overwrite Group 2 register;
F = 18 overwrite Group 1 register with a *mask*;
same as F = 16 except that a separate *mask* defines which bits in the selected register are set;

The functions F = 8 to 15 and F = 24 to 31 are control or test functions (**F=X1XXX**) without data transfer (however information may be conveyed by the **Q** bus line in any of these commands); for instance
F = 8 test the Look At Me line (response in Q);
F = 9 clear Group 1 register;
F = 11 clear Group 2 register.

For the full list of function codes see Table 18.5 in [1]. The set of available functions is specific to a particular module. Not all functions are necessarily implemented in a specific module. There are also function codes that are not assigned and can be defined for special applications.

Status Information

The response **Q** (one dedicated bus line common to all modules) indicates the success of an operation or the result of a test function.

The response **X** (one dedicated bus line common to all modules) indicates that the module has recognized and accepted the combination **A** and **F** (i.e. the command).

LAM (Look At Me): any module can require attention by generating a signal on this line (there is one dedicated line on the bus for each module as for the stations). A module can generate a **LAM** for instance at the end of an ADC conversion to signal that data is ready for transfer. The 23 individual lines are connected to individual pins in the crate controller and can be connected to the interrupt protocol of the data acquisition computer. In our implementation we will not use the LAM signal, but rather poll a *flip-flop* module to initiate the readout (see Example 4).

The busy signal **B** (1 dedicated bus line common to all modules) is used to interlock various system activities, which can compete for the use of the dataway. Specifically, it is generated during dataway operations. Whenever **N** is present, **B** is present.

Common Controls

Three common controls are available to all modules in the crate. Each has a dedicated line on the dataway.

Z - the initialize signal has absolute priority; this signal resets all modules in the crate to default values and set the inhibit to ON;

I - the inhibit signal inhibits any activity; this line is often used as a veto;

C - this command clears all data registers.

An example will help to clarify the CAMAC addressing scheme: let's suppose that we want to read the number of pulses acquired by the channel 5 of a scaler module (see Example 2). A function code (F = 0) is provided to read the scaler module. Each channel is identified by the corresponding sub-address (A = 0 to 15). The module is installed in slot 12 of the CAMAC crate. In this case, reading the content of scaler channel 5 is accomplished by issuing a CAMAC cycle with F = 0, N = 12, and A = 5.

Digital Signal Standards

The potentials for the (binary) digital signals on the dataway lines have been defined to correspond with those for compatible current sinking logic devices (e.g. TTL). The signal convention has, however, been inverted to be negative logic. The high state (more positive potential) corresponds to logic "0" and the low state (near ground potential) corresponds to logic "1".

Pull-up current sources for all dataway bus lines are located in the crate controller (that's why we say that the crate controller reads and writes and not the modules). The minimum pull-up current when the dataway line is at +3.5 V is defined as 2.5 mA, but preferably not less than 6 mA.

Before inserting or removing modules always turn off the CAMAC crate!

3 Using the JENET Library

High level functions for the C111 crate controller are derived from the ESONE standard and are implemented in the JENET Library [5]. The JENET Library has been specifically developed for the CAEN C111 crate controller and will not work with other CAMAC controllers. A brief introduction to the most commonly used functions follows.

In order to exchange data between the PC and the C111 create controller, first the communication between the host PC and the controller itself must be established by calling the function

```
short CROPEN(char *address);
```

The function requires a string parameter with the IP address of the crate controller. All Lab controllers use the IP address "192.168.0.98". This address can be changed via the serial RS-232 socket on the controller. The function returns an integer (`short`) that identify the connection and has to be used in all subsequent function calls to identify the established communication channel. A negative value indicates that the connections failed. In this case you have to exit the DAQ program and retry.

The function

```
short CSCAN(crate_id, &scan_result);
```

scans for slots containing modules in the crate identified by `create_id` The `crate_id`

parameter is the value returned by the `CROPEN` function.
The result is returned in `scan_result` (`unsigned int`). If a bit is set, the station is filled with a module.
The initialization of the CAMAC crate is performed by calling the function

```
short CCCZ(short crate_id);
```

which puts all modules in the crate in their default state and sets the inhibit to ON (i.e to 1).

The operation of the CAMAC modules can be inhibited or resumed by a call to the function

```
short CCCI(short crate_id, char data_in);
```

The inhibit is set to ON if `data_in = 1` or to OFF if `data_in = 0` (see Example 2). The function

```
short CTCI(short crate_id, char *data_out);
```

returns in `data_out` the status of the inhibit.

A 24-bit CAMAC command is executed by calling the function

```
short CFSA(short crate_id, CRATE_OP *cr_op);
```

The data and the result of the operation are returned in the `CRATE_OP C` structure, which is defined as follows:

```
typedef struct {
   char F;
   char N;
   char A;
   char Q;
   char X;
   int DATA;
} CRATE_OP;
```

where **F** is the CAMAC function code to be executed, **N** is the slot number of the addressed module, **A** is the sub-address within the addressed module, DATA is the data read from or to be transmitted to the module (depending whether a read or write function has been issued), **Q** and **X** are status flags to verify the success of the CAMAC cycle.

Similarly, a 16-bits command is executed with the function

```
short CSSA(short crate_id, CRATE_OP *cr_op);
```

The status bits **Q** and **X** can be tested with the function

```
short CTSTAT(short crate_id, char *Q, char *X);
```

The communication with the CAMAC crate should be closed (it is good practice!) before ending the DAQ program with a call to the function

```
CRCLOSE (short crate_id);
```

The JENET Library provides also functions specific to the C111 crate controller. For more details and more functions, consult the JENET Library [5].

Moreover, we use the function `kbhit()` for instance to exit from the DAQ loop instead of killing the program with CTRL-C. `kbhit()` returns 1 if any key on the keyboard has been hit.

4 How to Install the Crate Controller

In order to use the C111 crate controller a second ethernet card has to be installed in the host computer (eth1) with IP address "192.168.0.1" (different IP address than the controller but on the same local network), mask "255.255.255.0", and gateway "0.0.0.0" (no DHCP).

5 The WEB Interface

To help setting up the system, the C111 controller comes with a Local Web Server (web based interface). This interface allows for instance to test the modules and the actions of different functions, (or the CAMAC controller itself) without the need of setting up the whole DAQ chain. The interface is almost self explanatory (for details see [4]). To access the interface, in the browser open the local web server with the IP address of the controller and the user/password = jenet/jenet.

6 How to Compile Under Linux

To compile under Linux, for instance the program `scaler.c` which reads out a scaler module (Example 2), type at the prompt

```
g++ -o scaler scaler.c crate_lib.c -lpthread
```

`create_lib.c` contains the JENET functions and the system library `pthread` is required for multi threading.

Do not forget to include the header

```
create_lib.h
```

at the top of your program. The header includes the declaration of the JENET functions and of the data structure.

To compile, you can also use a Makefile script (recommended).

Example 1: Using the Dataway Display

This example shows how to use a dataway display module, model BORER 1802, to monitor the activity on the dataway by *writing* a variable to the dataway. The dataway display is usually inserted into the leftmost slot in the crate (N = 1).

First the DAQ program establishes a communication between the host PC and the crate controller (`CROPEN`, line 19) and clears the modules in the crate (`CCCZ`, line 27). The DAQ loop starts at line 38. The variable to be written to the display is incremented in the loop at line 39. The data structure `cr_op` including the function code **F**, module location **N**, and data **DATA** is filled at lines 41 to 43. A CAMAC command is then issued at line 44 (`CFSA`). To exit from the DAQ loop, hit the keyboard (function `khbit()`, line 50). The connection between the PC and the crate controller is closed at line 57 (`CRCLOSE`).

After starting the program, have a look at the left column of leds on the dataway display. If correctly executed, the leds will start blinking.

The source codes are not provided on purpose (only listings). You have to write the programs yourself!

datawaydisp.c

```
1  //datawaydisp.c
2  //program to test the dataway
3  #include <iostream>
4  #include "crate_lib.h"    //JENET CAMAC Crate Library
5  #include "kbhit.c"    //definition of function kbhit()
6
7  using namespace std;
8
9  //put here the location (address N) of each module
10 const int N_DWDISP = 1;
11
12 int main(int argc, char *argv[]) {
13
14 //C111 data structure F, N, A, Q, X, DATA (char,char,char,char,char,int)
15    CRATE_OP cr_op;
16    short crate_id, res;
17
18 //open crate: default IP address for all Lab controllers 192.168.0.98
19    crate_id = CROPEN("192.168.0.98");
20    if(crate_id<0) {
21       cout << "ERROR: Unable to connect with specified IP address! \n";
22       return 0;
23    }
24    cout << "Crate opened " << crate_id << endl;
25
26 //clear crate
27    res = CCCZ(crate_id);
28    if(res<0) {
29       cout << "Error executing CCCZ operation: " << res << endl;
30       return 0;
```

```
31     }
32
33     cout << "Have a look at the dataway display! \n";
34     cout << "Press any key to interrupt ... \n";
35
36  //start the DAQ loop
37     int data = 0;
38     while(1) {
39        data++;   //increment the variable
40  //write data to the dataway display, CAMAC function code 16
41        cr_op.F = 16;
42        cr_op.N = N_DWDISP;
43        cr_op.DATA = data;
44        res = CFSA(crate_id, &cr_op);
45        if(res<0) {
46           cout << "Error executing CFSA operation: " << res << endl;
47           break;
48        }
49
50        if(kbhit()) {   //to exit from the loop use function kbhit()
51           cout << endl << "Ending DAQ loop" << endl;
52           break;
53        }
54     }    //end DAQ loop
55
56  //close crate
57     CRCLOSE(crate_id);
58     cout << endl << "Bye bye!" << endl;
59
60     return 0;
61  }
```

Example 2: Scaler Readout

The following example shows how to read a CAMAC scaler, model LeCroy 2551. First we open the communication with the CAMAC controller (l. 18), then we set the inhibit to ON (l. 37, the scaler stops counting) and clear the scaler (l. 45). Next we enable the scaler (the inhibit is set to OFF, l. 50) and pause the thread (program execution) with the function `sleep(nnn)` (l. 6) for a given number of seconds (`nnn`). During this time the scaler counts. Once this time has elapsed, we set again the inhibit to ON (l. 59) and read the content of scaler channel 5 (l. 67). Finally, the count rate is display (l. 71 and l. 72). Before ending the program, the connection between the host computer and the crate controller is closed (l. 75).

readscaler.c

```
1  //readscaler.c
2  //program to read a scaler
3  #include <iostream>
4  #include "crate_lib.h"    //JENET CAMAC Crate Library
5
6  using namespace std;
7
8  //put here the location (address N) of each module
```

```
9  const int N_SCALER = 14;
10
11 int main(int argc, char *argv[]) {
12
13 //C111 data structure F, N, A, Q, X, DATA (char,char,char,char,char,int)
14    CRATE_OP cr_op;
15    short crate_id, res;
16
17 //open crate: default IP address for all Lab controllers 192.168.0.98
18    crate_id = CROPEN("192.168.0.98");
19    if(crate_id<0) {
20       cout << "ERROR: Unable to connect with specified IP address! \n";
21       return 0;
22    }
23    cout << "Crate opened " << crate_id << endl;
24
25 //clear crate
26    res = CCCZ(crate_id);
27    if(res<0) {
28       cout << "Error executing CCCZ operation : " << res << endl;
29       returno 0;
30    }
31
32    cout << "Enter the duration of the measurement in seconds: ";
33    int ntime = 1;
34    cin >> ntime;
35
36 //set INHIBIT to ON for all modules in crate: the scaler stops counting
37    res = CCCI(crate_id, 1);
38    if(res<0)
39       cout << "Error executing CCCI operation: " << res << endl;
40
41 //clear scaler channel 5, CAMAC function code 9
42    cr_op.F = 9;
43    cr_op.N = N_SCALER;
44    cr_op.A = 5;
45    res = CFSA(crate_id, &cr_op);
46    if(res<0)
47       cout << "Error executing CFSA operation: " << res << endl;
48
49 //set INHIBIT to OFF for all modules in crate: the scaler resumes counting
50    res = CCCI(crate_id, 0);
51    if(res<0)
52       cout << "Error executing CCCI operation: " << res << endl;
53
54 //count for ntime seconds
55    cout << "Counting starts! Wait " << ntime << " seconds ..." << endl;
56    sleep(ntime);
57
58 //set INHIBIT to ON for all modules in crate: the scaler stops counting
59    res = CCCI(crate_id, 1);
60    if(res<0)
61       cout << "Error executing CCCI operation: " << res << endl;
62
63 //read scaler channel 5, CAMAC function code 0
64    cr_op.F = 0;
65    cr_op.N = N_SCALER;
66    cr_op.A = 5;
```

```
67      res = CFSA(crate_id, &cr_op);
68      if(res<0)
69          cout << "Error executing CFSA operation: " << res << endl;
70
71      cout << "Number of counts: " << cr_op.DATA << endl;
72      cout << "Counting rate: " << float(cr_op.DATA)/float(ntime) << " Hz \n";
73
74  //close crate
75      CRCLOSE(crate_id);
76      cout << "Bye bye!" << endl;
77
78      return 0;
79  }
```

Example 3: Using a *flip-flop*

Usually a *flip-flop* is used to stop (veto) trigger generation until the current event is not fully read out: the trigger sets the *flip-flop* to "1" and the *flip-flop* response **Q** (not the dataway bus line) is connected to the *veto* input of the trigger generation circuit to stop further trigger generation until the *flip-flop* is not cleared (see Figure 1). At the end of the event acquisition cycle the host computer resets the *flip-flop* through a CAMAC cycle and a new trigger can be generated.

The following example shows how to set, reset, and read a *flip-flop* module, model DPNC 750, which houses 8 independent *flip-flops*. The function F = 18 (l. 40) overwrites the *flip-flop's* register with a *mask*, i.e. it clears the bit selected by the *mask*. After starting the program enter the pattern you want to activate on the module and verify that the corresponding leds turn on.

flipflop.c

```
1   // flipflop.c
2   // sample program to play with the flipflop
3   #include <iostream>
4   #include "crate_lib.h"    //JENET CAMAC Crate Library
5
6   using namespace std;
7
8   //put here the location (address N) of each module
9   const int N_FF = 8;     //FF slot
10
11  int main(int argc, char *argv[]) {
12
13  //C111 data structure F, N, A, Q, X, DATA (char,char,char,char,char,int)
14      CRATE_OP cr_op;
15      short crate_id, res;
16
17  //open crate: default IP address for all Lab controllers 192.168.0.98
18      crate_id = CROPEN("192.168.0.98");
19      if(crate_id<0) {
20          cout << "ERROR: Unable to connect with specified IP address! \n";
21          return 0;
22      }
23
```

```
24 //clear crate
25    res = CCCZ(crate_id);
26    if(res<0) {
27       cout << "Error executing CCCZ operation: " << res << endl;
28       return 0;
29    }
30
31 //start the DAQ loop
32    int ffin;
33    while(1) {
34       cout << "Enter FF bits you want to activate in hex (0X__): ";
35
36 //clear FlipFlop (overwrite FF registers) function code 18
37       cr_op.F = 18;
38       cr_op.N = N_FF;
39       cr_op.DATA = 0xFF;
40       res = CFSA(crate_id, &cr_op);
41       if(res<0)
42          cout << "Error clearing FlipFlop: " << res << endl;
43
44 //set FlipFlop, function code 16
45       cin >> std::hex >> ffin;
46       cr_op.F = 16;
47       cr_op.N = N_FF;
48       cr_op.A = 0;
49       cr_op.DATA = ffin;
50       res = CFSA(crate_id, &cr_op);
51       if(res<0)
52          cout << "Error setting FlipFlop: " << res << endl;
53
54 //read FlipFlop, function code 0, sub-address 0 reads all FF channels
55       cr_op.F = 0;
56       cr_op.N = N_FF;
57       cr_op.A = 0;
58       res = CFSA(crate_id, &cr_op);
59       if(res<0)
60          cout << "Error reading FlipFlop: " << res << endl;
61       cout << "FF pattern: " << std::hex << cr_op.DATA << endl;
62
63       cout << endl << "Continue or exit [y/n]? "
64       char ccc;
65       cin >> ccc;
66       if(ccc=="y") {
67          cout << "Ending DAQ loop" << endl;
68          break;
69       }
70    }  //end DAQ loop
71
72 //close crate
73    CRCLOSE(crate_id);
74    cout << endl << "Bye bye!" << endl;
75
76    return 0;
77 }
```

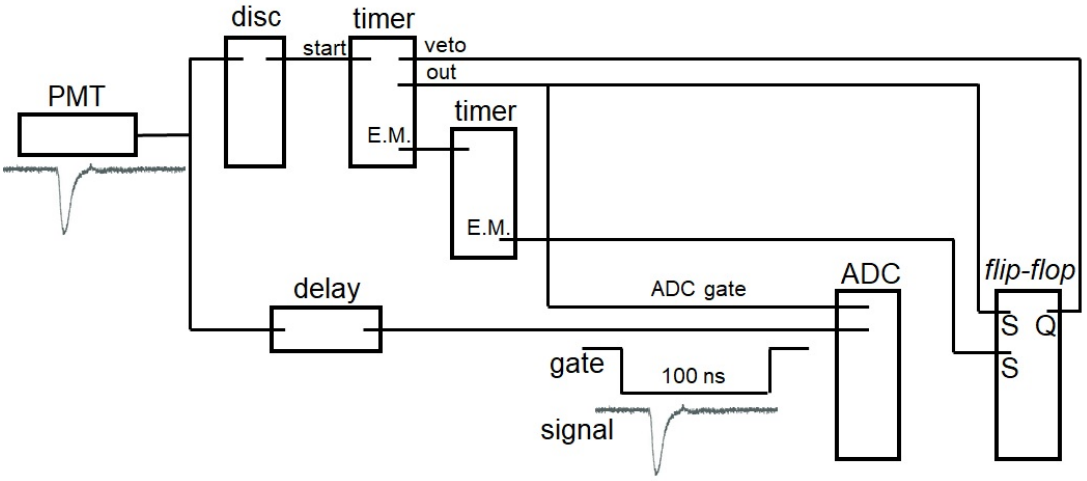


Figure 1: How to connect an ADC: The signal from the PMT is split in two parts. The first part goes to a discriminator and the discriminator output goes to a timer to generate the gate (100 ns) for the ADC module. A second output from the timer goes to a *flip-flop* to set the *flip-flop* (**S**). The *flip-flop* response **Q** is used to *veto* the generation of new gates until the *flip-flop* is reset by the DAQ. The second part of the PMT signal is delayed such to coincide with the gate before entering the ADC module. The gate has to open around 10 ns before the PMT pulse. The *end marker* (E.M.) of the first timer starts the second timer and the E.M. generated by the second timer is used to generate a *time out* (50 μs later) and sets the second *flip-flop* to signal to the DAQ that the ADC has completed the conversion of the PMT signal and is ready for readout.

Example 4: ADC readout

This example shows how the operate and read an Analog to Digital Converter module (ADC), model LeCroy 2249A. In order to digitize a pulse, a gate in coincidence with the input PMT signal has to be generated (see Figure 1) to open the sampling window of the ADC. A *flip-flop* is used to inhibit the generation of new triggers until it is not reset by the DAQ at the end of the event acquisition cycle. To signal to the DAQ that the ADC has completed the digitization of the input pulse a second *flip-flop* is set. The DAQ program polls the *flip-flop* module until the correct pattern is not seen and then acquires the digitized pulse. Otherwise the program continues polling the *flip-flop* module. Finally it resets the *flip-flops* and a new acquisition cycle can start.

When launching the DAQ program you can also enter the controller's IP address at the prompt (l.15 and l.22). After establishing the connection between the PC and the controller, the crate is scanned for modules at line 34. The output data file is opened at line 59. Then the crate is initialized and the modules are cleared. The DAQ loop starts at line 89. The *flip-flop* module is polled between lines 95 and 100. If a trigger has been generated and the ADC has completed the conversion, the event is acquired, i.e. the ADC module is read at line 106, and data is written to the screen and to the output data file. Otherwise the program continues polling the *flip-flop* module until the correct pattern is not identified. The *flip-flop* is reset at line 129, the *veto* is removed and trigger generation resumes, and a new acquisition cycle starts.

Although we are reading only one ADC channel, this is already a full DAQ program. To speed up the execution of the program, you can comment all unnecessary writing to the screen.

readADC.c

```
1  //readADC.c
2  //sample program to read one ADC channel (LeCroy 2249A)
3  #include <iostream>
4  #include <fstream>
5  #include "crate_lib.h"    //JENET CAMAC Crate Library
6  #include "kbhit.c"    //definition of function kbhit()
7
8  using namespace std;
9
10 //put here the location (address N) of each module
11 const int N_DWDISP = 1;
12 const int N_ADC = 8;    //ADC slot
13 const int N_FF = 16;    //FF slot
14
15 int main(int argc, char *argv[]) {
16
17 //C111 data structure F, N, A, Q, X, DATA (char,char,char,char,char,int)
18    CRATE_OP cr_op;
19    short crate_id, res;
20
21 //open crate: default IP address for all Lab controllers 192.168.0.98
22    if(argc==2)
23       crate_id = CROPEN(argv[1]);
24    else
25       crate_id = CROPEN("192.168.0.98");
26    if(crate_id<0) {
27       cout << "ERROR: Unable to connect with specified IP address! \n ";
28       return 0;
29    }
30    cout << "Crate opened " << crate_id << endl;
31
32 //scan the crate for modules (optional)
33    unsigned int scan_result;
34    res = CSCAN(crate_id, &scan_result);
35    if(res<0)
36       cout << "Error scanning the crate : " << res << endl;
37    for(int i=0; i<24; i++) {
38       if(scan_result & (1 << i))
39          cout << "Station " << i+1 << " is filled with a module. \n";
40    }
41
42 //open output data file
43    bool done = false;
44    string adcname;
45    cout << "Enter filename for data file: ";
46    cin >> adcname;
47 //check status of data file
48    done = false;
49    while(!done) {
50       std::ifstream fdata(adcname.c_str());
51       if(fdata.is_open()) {
52          cout << "ATTTENTION: You might erase data!" << endl;
53      cout << "Enter a different filename: ";
54          cin >> adcname;
55       }
```

```
56          else
57              done = true;
58      }
59      std::ofstream fdata(adcname.c_str());    //open data file
60
61  //clear crate
62      res = CCCZ(crate_id);
63      if(res<0) {
64          cout << "Error executing CCCZ operation: " << res << endl;
65          return 0;
66      }
67
68
69  //clear ADC (all channels) function code F9
70      cr_op.F = 9;      //clear ADC
71      cr_op.N = N_ADC;
72      cr_op.A = 0;
73      res = CFSA(crate_id, &cr_op);
74      if(res<0)
75          cout << "Error clearing ADC: " << res << endl;
76
77  //clear Flip/Flop function code 18
78      cr_op.F = 18;
79      cr_op.N = N_FF;
80      cr_op.A = 0;
81      cr_op.DATA = 0xFF;
82      res = CFSA(crate_id, &cr_op);
83      if(res<0)
84          cout << "Error clearing FF: " << res << endl;
85
86  //start the DAQ loop
87      int patt;    //FF pattern
88      int xadc1;    //ADC data
89      while(1) {
90
91  //check if trigger generated (check FlipFlop)
92          cr_op.F = 0;
93          cr_op.N = N_FF;
94          cr_op.A = 0;
95          res = CFSA(crate_id, &cr_op);
96          if(res<0)
97              cout << "Error reading FlipFlop: " << res << endl;
98          patt = cr_op.DATA;
99          if(patt == 3) {
100             cout << "FF 1 and 2 are set \n";
101
102 //read ADC first channel
103             cr_op.F = 0;
104             cr_op.N = N_ADC;
105             cr_op.A = 0;
106             res = CFSA(crate_id, &cr_op);
107             if(res<0)
108                 cout << "Error reading ADC: " << res << endl;
109             xadc1 = cr_op.DATA;
110
111 //write data to file
112             cout << "ADC: " << xadc1 << endl;
113             fdata << xadc1 << endl;
```

```
114
115 //clear ADC (all channels)
116           cr_op.F = 9;
117           cr_op.N = N_ADC;
118           cr_op.A = 0;
119           res = CFSA(crate_id, &cr_op);
120           if(res<0)
121              cout << "Error clearing ADC: " << res << endl;
122           cout << "ADC cleared \n";
123
124 //clear FlipFlop
125           cr_op.F = 18;
126           cr_op.N = N_FF;
127           cr_op.A = 0;
128           cr_op.DATA = 0xF;
129           res = CFSA(crate_id, &cr_op);
130           if(res<0)
131              cout << "Error clearing FF: " << res << endl;
132           cout << "FlipFlop cleared \n";
133
134           if(kbhit()) {   //to exit from the loop use function kbhit()
135              cout << endl << "Ending loop" << endl;
136              break;
137           }
138        }   // end waiting for trigger
139     }   // DAQ end loop
140
141 //close crate
142     CRCLOSE(crate_id);
143     cout << endl << "Bye bye !" << endl;
144
145     return 0;
146 }
```

References

[1] W. R. Leo, *Techniques for Nuclear and Particle Physics Experiments*, 2nd Ed., Springer-Verlag (1994).

[2] CAMAC standard.

[3] *LeCroy 1997 Research Instrumentation Products Catalog.*

[4] CAEN, *Mod C111C Technical Information Manual.*

[5] *C111C JENET C Library.*