Configuration program for a correlator based on the NFRA cards

G.Comoretto

September 1992

Arcetri Technical Report N° **9/1992**

Abstract

A set of configuration routines for the new Arcetri autocorrelation spectrometer, based on the correlator chip designed at the Netherland Foundation for Research in Astronomy, is presented.

These routines allow to specify the configuration of the instrument in an abstract way (number of channels to correlate, input channels, etc.), and set an array of "objects" each representing a correlator chip. The configuration informations, including hardware control word and delay interval computed, are then retrieved from the "object" by using simple routines.

1 Introduction

The Arcetri correlator now to be built by the radioastronomy team is a very versatile spectrometer, capable of both auto and cross correlation, with a maximum instantaneous bandwidth of 160 MHz and up to 4096 resolution channels[4]. This instrument is based on the NFRA boards and chips, that are programmable devices. In this report are described the algorithm and programs used to generate the programming sequence.

The NFRA correlation card consists of an array of 8×8 correlation chips. Each chip has two sets of 4 inputs and two outputs. Each input set feeds an 8-bit shiftregister using a 4-to-1 multiplexer, and a complete correlation of the signals is performed corresponding to 16 relative delays of the two signals. The chip is described in detail in [1]. The correlation board is described in detail in [3], and its use is outlined in [2]. A general familiarity with both the chip and the board is assumed in this report.

The chips can be cascaded in a huge number of configurations to allow for auto- and cross-correlation of up to 8 different signals, and for a total of up to 1024 delays. More cards may be cascaded to allow for more delays to be calculated.

For homogeneity with the original documentation, each card in a system is referred as a "module". Thus, in this report, the words "card" and "module" are used with the same meaning.

The card is very complex: each of the chips in a module can receive inputs from either of two external inputs, from a cascade network or from a permutation network. The cascade network connects all chips to form a a fractal like pattern. The permutation network connects all chips in each column to those in the previous one using a different permutation for each column. It is used mainly to modify the cascade pattern for cross correlating different signals.

The cards may operate in time multiplexed schemes, to correlate signals faster than the chip basic clock (40 MHz). In this schemes the input signal is sampled at an adequate speed, typically 2, 4 or 8 times the correlator clock, and successive samples are cross correlated in parallel by resp. 2×2, 4×4 or 8×8 cross correlators. The cross products are then reassembled in an autocorrelation function with the appropriate step. It is also possible to cross correlate signals that are time multiplexed.

When more than one module is used in a cross correlation or time multiplexed configuration, different cards compute different products. This is accomplished using the permutation array to permute the input signals in different way for each module.

The routines were written with an aim of generality, i.e. the program needs not to be totally rewritten for a different configuration of the instrument. The hardware dependent routines are kept as segregate as possible. The program, in this implementation, makes the following assumptions on the hardware configuration:

- All modules have the external Y inputs connected with standard 4×4 straps
- All cascade inputs are connected with the standard cascade pattern, described in [2]
- The last module in the system has the YC inputs connected with an auto strap connection. Note that the number of modules in a system is not specified. Physically it implies that a "strap card" is always present after the last card.
- The X0 and X1 inputs of each card are connected, in parallel, to the ouputs of sampler cards 0 and 1, respectively.
- The XC inputs for the first card are left unconnected (grounded).
- The XT inputs of each card are connected to a test pattern generator, in parallel for each card. The YT inputs are connected to the CX7 output of the same module

This configuration allows to generate almost any auto and cross-correlation function of one or two inputs, with time multiplexing factors (TMF) from 1 to 8, and the autocorrelation of up to 8 signals without time multiplexing (TMF=1).

Due to all these possibilities, the final configuration of the correlator may become utterly weired, with adjacent chips carrying products for different cross correlations and successive delays computed by chips in different cards. It is therefore mandatory to use an automated procedure, both to configure the chips and to keep track of where the correlation products are computed.

The routines operate on an abstract description of a correlator cell, physically corresponding to an IC in a module. This description, an "object" belonging to a "class" in object oriented terminology, consists of a data structure and the routines necessary to manipulate it. A correlator module is an 8×8 array of such cells, and the correlator is represented by a 3-dimensional array.

The software is written using standard C, despite its object oriented nature. It can be easily ported to C++, taking advantage of the specific language features.

The higher level routine is `calc_configuration()`. It accepts an abstract description of the configuration desired, and returns an array of cells with all the informations needed. The configuration is described by the number of channels to correlate, the number of correlation cards needed, and the sampler channels to be connected to each correlator input. In addition, the routine determines the appropriate configuration to be used in each sampler card. The sampler configuration may be also specified in advance.

The relevant informations can be extracted from a cell structure by the routines `get_cell_cword()`, that returns the control word to be written in the chip control register, and `get_cell_offset()`, that returns the offset of the first correlation product in the chip and the delay between successive products. This last is used in time multiplexed schemes, where the signal is processed in parallel by several chip chains. The routine `get_sampler_id()` returns an identifier for the samplers whose signals are correlated.

The general structure of the configuration obtained can be extracted and printed on the standard output using the routine `report_status()`. The routine prints a short report (one line) for each of the subsections in which the instrument has been divided. The report includes:

- the first cell (chip) used in that section
- the sampler inputs connected to that section
- the number of delay steps computed
- the range of delays and the delay step, in shift clock periods

A graphic routine is also available for a PC with a graphics (CGA) adapter, or for a PostScript printer. This routine displays the connections for each card in the configuration.

To write the configuration routine, I extensively used modules and algorithms from a program previously developed by Albert Bos. I mus thank Albert Bos also for the explanations on the basic philosophy of the correlator cards.

2 Configuration routine

Synopsis:

```
int calc_configuration(
    int ncx, int ncy,  // Number of logical channels to
                       //      correlate. For autocorr. ncy=1
    int tmf,           // time multiplexing factor
    int nmodule,       // Modules in configuration
    int maxmodule,     // Modules in system
    int firstmodule,   // First module to use (0 = first)
    int sampler[],     // Samplers to use (nmodule values)
    int sampl_conf[2], // On input: sampler configuration if
                       //  assigned, or -1 if not assigned.
                       // On output, the same (may be changed)
    CELL state[MAXMODULE][8][8]) // computed cell configuration
```

The routine computes everything is needed to program the correlator chips and to use the resulting correlation products. All the informations needed are mantained in the CELL objects. Each object contains the informations relevant to one chip. CELL's are stored as an array in module, column, row order. For example, `state[1][5][3]` refers to the chip in card (module) 1, 5^{th} row, 3^{rd} column. Remember that in C arrays are numbered from 0, so that rows and columns run from 0 to 7, and cards from 0 to 3 (in a 4-card system). The macro MAXMODULE is set to the maximum possible number of modules in the program include file. The parameter `maxmodule` is a variable, that reflects the actual number of modules present in the system at a given time.

It is possible to compute the configuration relative to a subset of the correlation cards available in the system. The remaining cards can be configured with successive calls to `calc_config()`, providing that the configuration codes for the samplers returned from the first call are not modified. In this case, successive calls to `calc_config()` can configure the samplers in a consistent way, using the array `sampl_conf[]` to pass informations. The format for this array is explained later on.

The routine first checks for gross errors in the configuration specified. The program allows only for autocorrelations (`ncy = 1`) of 1, 2, 4, or 8 signals (maximum 2 if `tmf` $\neq$ `1`), cross correlation of 2×2 signals, and time multiplexing factors of 1, 2, 4 or 8. Cross correlation is not available with `tmf = 8`. More than 2 autocorrelations are available only if `tmf` is 1 and if the last card in the system is included in the configuration, since the auto strap connection is needed and it is only available on the last card in the system.

The input signals are then determined. The routine assumes that the samplers are either unconfigured, or that the configuration code is specified in the array `sampl_conf[]`. The routine `calc_x_inputs()` determine the appropriate sampler configuration for the used samplers, and, for each X input, the sampler to be connected. Sampler cards 0 and 1 are connected to X0 and X1 inputs respectively, in parallel for all correlation cards.

The array `sampler[]` specifies the samplers whose signals are correlated. Up to 8 values can be specified. The order in which samplers are specified is not relevant, but only the first `ncx` values are considered. Values 0 to 3 specify the 4 samplers in sampler card 0, 4 to 7 those on sampler card 1, and 8 specify the test pattern generator.

The configuration process is done in three steps, in the attempt to enucleate those aspects more dependent on the particular configuration of the Arcetri correlator from the general problem of configuring a NFRA board. These steps correspond to a progressive translation from the more abstract description given to the routine to the final almost physical description contained in the `CELL` array.

In the first step, a more detailed, but still abstract description of the configuration is computed. This includes all the informations specific to the hardware configuration, but does not contain all the informations related to the standard algorithms of setting the NFRA modules. The informations computed are:

- the configuration of the X external inputs, specified row by row and the same for each module in a configuration block
- the configuration of the Y external inputs, specified module by module by a configuration code, that may assume a limited number of “Y configurations”.
- the status of the permutation connections, specified column by column in a “permutation code” for each module in the configuration block.

These informations are then used in a system independent routine to set the switch selectors of all chips in the configuration block, during the second step of the configuration process.

In the last step, the correlator chains so formed are traced, first forward, starting from those chips that have an external line connected to the X input, and then backward. In the trace process, the routine sets, for each cell object:

- the sampler attached to the X and Y paths
- the total delay in the X and Y paths, including the delay in the sampler time multiplexer (serial-to-parallel converter)
- the relative delay between the two paths, and thus the correlation range computed
- whether an extra delay is needed, both to compensate the anticipated input in columns 3 and 7 of each module and to correctly compute negative delays in cross correlations
- the hardware control word to be written in the chip control register

This last process requires the program to know about the hardware configuration. The relevant informations are put in an include file, so that hardware dependencies can be enucleated.

If at any point an error is detected, the routine returns an error code, that can assume one of the values in tab. 1. An error code of 0 indicates a successful exit.

ERR_NCHANNELS	1	error in the specified nch
ERR_TMF	2	error in the specified tmf
ERR_CHANNELS	3	failed in assigning the channels
ERR_NCARD	4	failed in assigning the channels

Table 1: Error return codes

3 Input assignment routine

The setting for the X input external selectors is specified, for each module row, in the routine `calc_x_inputs()`. Its synopsis is

```
int calc_x_inputs(
    int ncx,           // Number of X inputs
    int sampler[8],    // Requested samplers
    int tmf,           // Time multiplexing factor
    int sampl_conf[2], // On input: sampler configuration if
                       //  assigned, or -1 if not assigned.
                       // On output, the same (may be changed)
    int x_selector[8]) // Output. X selector for each module row,
                       //  or -1 if not used
```

Each card receives two sets of input signals, one from each sampler. Depending on the required correlator configuration, it is necessary to determine both the sampler configuration, i.e. which sampled signals are fed to each sampler output line, and the input setting of the correlator cards. When different cards use the same sampler, the sampler configuration cannot be chosen arbitrarily. Due to the philosophy of this program, the sampler configuration is determined by the first correlator block that use a given sampler. Successive blocks must conform to this configuration, or use a different sampler.

3.1 Sampler configurations

The Arcetri correlator has two identical sampler cards. Each card contains two sampler modules, with two sampler each. Thus a total of 8 samplers is present in the system, thus fulfilling the requirement of having up to 8 independent channels.

An extra "sampler", that generates a programmable test pattern, is present in the clock generation card. Its output is fed in parallel to the XT inputs of all modules.

Each sampler works at a fixed sampling rate of 160 MHz, and samples are slowed down to the correlator clock of 40 MHz by time multiplexing each sampler output into four separate lines. It is possible to have a faster sampling rate, and thus bandwidth, by interleaving the two samplers in the first module of each card. In this way, a time multiplexing factor of 8 is achieved.

Each sampler card has 8 output lines (each two bytes wide), and thus it is necessary to select which of the $4 \times 4 = 16$ sampler outputs are fed to the output lines. Moreover, the correlator cards require the input signals to be in appropriate lines. The selection is done by a multiplexer in the output circuitry of each sampler card. The multiplexer selects between 4 possible configuration, according to table 2. Each column describes one of the possible configurations, with the first digit denoting the sampler, and the second denoting the bit in a time sequence. Thus S2.0 denotes the first (least delayed) bit of sampler S2, and S1.1 denotes the second bit (1 sampler clock earlier than bit 0) of sampler S1. Delays are specified in units of the sampler clock period, 1/160MHz.

	C0	C1	C2	C3
Line 0	S0.0	S0.0	S0.0	S2.0
Line 1	S1.0	S0.1	S0.0	S2.0
Line 2	S0.1	S0.2	S1.0	S0.0
Line 3	S1.1	S0.3	S1.0	S0.0
Line 4	S0.2	S1.0	S2.0	S3.0
Line 5	S1.2	S1.1	S2.0	S3.0
Line 6	S0.3	S1.2	S3.0	S1.0
Line 7	S1.3	S1.3	S3.0	S1.0

Table 2: Configurations for sampler output multiplexer

Configuration C0 and C1 are used for time multiplexed configurations. C2 and C3 are used mainly for configurations with many input channels and no time multiplexing.

3.2 Configuration selection algorithm

The selection algorithm is probably the portion of the program most dependent on the specific structure of the Arcetri correlator. It depends both on how the samplers are connected to the correlation boards and on the specific sampler configurations listed in table 2.

The configuration routine, `calc_x_inputs()`, uses two different algorithms depending on whether time multiplexing is used or not.

For time multiplexed configurations, the routine first determines which samplers are to be used, and check if they are compatible with those already specified (if any). If both samplers are used, the same configuration must be used in both of them.

Only cross correlations of samplers S0 and S4 to samplers S1 and S5 are allowed. A cross product $S1 \times S5$, for example, is not allowed. It is possible to perform simultaneously two autocorrelations and a cross correlation, but the autocorrelations must refer to the same channels that are also cross correlated.

After this initial check, the routine looks in a list of permitted configurations. Each configuration is specified by the TMF, the number of X input channels, and the corresponding sampler configuration. If a valid combination is found, the sampler configuration is set, either from the value in the list or from that specified to the routine in `sampl_conf[]`.

The only valid sampler configurations are C0 and C1. For configuration C0, sampler S0 is connected to even rows, while for configuration C1, sampler S0 is connected to the first four rows. For each row, the routine thus determines at which sampler the row inputs are connected, and selects the appropriate external input connection for that row. For example, in a cross correlation of S0 to S5, using configuration C0, rows 0, 2, 4 and 6 are connected to sampler S0 (sampler card 0, thus external input X0), and rows 1, 3, 5 and 7 are connected to sampler S5 (sampler card 1, thus external input X1).

For configurations without time multiplexing (TMF=0), a "brute force" algorithm is used. The permissible sampler configurations are determined to be either those specified to the routine in `sampl_conf[]`, or to be in the range 1–3 (configuration 0 is not used), this for both sampler cards.

Then, fixed the sampler configuration, the routine looks at the samplers connected to the X0 and X1 inputs of each permissible row. If the right sampler is found, the routine notes it. If at the end of the row scan the correct samplers are not found, the routine skips to the next sampler configuration. Note that in this case it is not required that both samplers share the same configuration.

This process allows to cross correlate most of the input samplers among them or to autocorrelate up to 8 independent channels.

If an error is detected, the routine returns one of the errors listed in Tab. 1.

ERR_CHANNELS may be due to the fact that an incompatible sampler configuration has been specified. It may be also due to a limit in the routine, rather than a real error, since we have not demonstrated that the algorithm is capable to find *any* possible configuration.

The other errors are caused by erroneous parameters, either of `ncx` and `ncy`, or `tmf`, respectively.

4 Configuration parameters

Each card configuration can be completely specified by three sets of informations:

1. which inputs are used for the external X input connections (specified by row)
2. whether permutation connections are used (specified by column)
3. which inputs are used for the external Y input connections (specified by column)

The first information is computed during sampler configuration, as described in the previous section. The remaining two informations depend on the particular setting of both the correlator cards and the samplers. No simple algorithms have been found to determine them, and an exhaustive brute force search of all the possible combinations is too slow for any practical purpose.

Both these informations depend however only on a particular set of informations, that can be easily derived from the high-level configuration specification, namely:

1. the number of correlators to be implemented in each card, both in the X (column) and Y (row) direction (`nx` and `ny` resp.)
2. the number of modules connected in parallel, `np`
3. the number of modules connected in cascade, `nc`
4. for complex configurations, it may be useful to split the module in `ns` separate subsections, each occupying 8/`ns` consecutive rows.

If modules are split (`ns` > 1), a separate set of parameters are specified for each submodule. This program deals with up to two submodules per module. In this program we used `ns` $= 2$ for some particular configurations with 2 input channels. These configurations use the two subsections independently, to correlate the two signals, taking advantage of the high degree of symmetry present among the two module halves.

`nx` is equal to the number of input channels, `ncx`, times the `tmf`. The remaining parameters are determined by the conditions that: `ny` $\cdot$ `np` $=$ `ncy` $\cdot$ `tmf`, `np` $\cdot$ `nc` $=$ `nmodule`, that all parameters must be ≥ 1, and that it is possible to have cards in cascade only with `ny` $=$ `1`.

Once these parameters are computed, the two routines:

```
calc_permutation_array(nx, ny, np, nc, ns, perm)
calc_y_states(nx, ny, np, nc, ns, y_selector)
```

determine the arrays `perm` and `y_selector`, both of dimension `nmodule` $\cdot$ `ns`, that specify the module setting in a system independent way. If $ns > 1$, parameters are specified with submodule index running faster, i.e. first all codes for the first module are specified, then for the second, and so on.

4.1 Permutation codes

Each element of `perm[]` is interpreted as a bit pattern, each bit corresponding to a column of cells in a module. If a bit is set, the permutation connections for the X inputs of the corresponding column, and for the Y input of the previous one, are used. Bit 0 selects the cascade X connections for the first column, thus cascading this module with the previous one, while bit 8 selects the cascade Y connections, thus cascading this module with the next one (remember that column numbering start from 0). For example, a code of 4 (bit 2 set) specifies that permutation connections must be used in connections going from column 1 to column 2. In this way, the X and Y connection patterns are automatically set in a consistent way.

The permutation code `perm` does affect only those connections going "forward" (from successive columns for Y connections, or from previous columns for X connections). The remaining permutation connections are used to implement "non square" correlators, i.e. configurations with `nx` $\neq$ `ny`, and are selected automatically by the system independent configuration routine.

The algorithm used to assign the permutation codes is very general. First we describe the case without split or cascaded modules. For each of the 10 possible `nx` $\times$ `ny` combinations (they can only assume power of 2 values, and `nx` $\geq$ `ny`), a base value and up to 3 modifiers are specified. For each module in the configuration the code is computed by xor-ing the base value and a different set of modifiers. The first modifier is used for odd modules, the second for the last two modules on every four, the third for the last 4 in each group of 8. In this way, the signal reaching the Y inputs is different for each module in the set.

If modules are connected in cascade, the permutation code for each group of cascaded modules is computed as above, and the successive modules are connected with a standard cascade pattern, depending only on `nx`. This pattern, together with the standard cascade interconnection in the backplane, preserves the order of the signals in the last column of each module. The cascade bits in the permutation code (bits 0 and 8) are also set as appropriate, to select the cascade connections for the first and last columns in the affected modules.

If modules are split in submodules, different Y connections are specified for the upper and lower half of each module. Permutation codes are assigned as

code	connections
0	Y0 input set (state 2)
1	Y1 input set (state 3)
2	Y0 for col. 0..3, Y1 for col. 4..7
3	Y1 for col. 0..3, Y0 for col. 4..7
4	Y1 for row. 0..3, Y0 for row. 4..7
5	Y0 for row. 0..3, Y1 for row. 4..7
6	Y0 for row. even, Y1 for row. odd
7	Y0 for row. odd, Y1 for row. even
8	Y0 for row. 0145, Y1 for row 2367
9	Y0 for row. 2367, Y1 for row 0145
10	auto strap connections

Table 3: Codes for Y input selection

above for the first `nmodule` submodules (half of the modules), and in reversed order for the second half. In this way, in the second half modules upper and lover submodules are interchanged.

4.2 Y selector codes

Each element of `y_selector` specifies a predefined pattern of connections to the Y0 and Y1 external inputs. Existing codes are specified in table 3. The code is specified for each submodule, as explained above. In this program, codes 2 and 4 are not used.

The configuration code depends mainly on the quantity `ncx/ncy`, that is equal to the number of autocorrelators if no cross correlations are computed, and is equal to 1 for cross correlation configurations. In fact, a complete cross correlator is exactly the same than an autocorrelator with higher time multiplexing factor.

For all configurations with 2 autocorrelators, code 5 is used, i.e. the upper half of the card uses Y0, and the lower part uses Y1. Input signals are routed so that the first sampler goes to the upper half of the X inputs, and the second sampler goes to the lower half. If many (> 2) correlators are present, the modules are configured as parallel, long correlators, running on all modules and terminated using the auto strap connection on the last module (code = 10). Due to the nature of the module connections, correlators are permuted after each column, and the whole resembles a spaghetti thread, but each correlator is guaranteed to starts on column 0 of module 0 and to end in some row on column 7 of the last module.

For a single autocorrelator, and for crosscorrelators, if just one module is present, code 1 is used (1^{st} half columns use Y0, 2^{nd} half uses Y1). If more modules are used, the first half use Y0 (code 0), and the second half uses Y1.

If split modules are specified, a rather complex structure is used. For 4×4 configurations, codes 8 and 9 are used on alternate cards, while for 2×2 configurations, codes 6 and 7 are used. This have been found by hand, trying to match X and Y inputs, and I have not found a simple justification.

5 Chip input selector assignment

Each chip has two switches, one for the X and one for the Y signal. Each switch can assume one of four states, being connected to the cascade network, to the permutation network, or to one of two external inputs.

The correlator status can be (almost) fully specified once the selector switch is set for each correlator chip. Once these switches are set, the instrument is divided in a set of well-defined correlators, each correlating well-defined signals.

An extra degree of freedom remains, in that each chip has an extra delay that can be added before its input. This delay is assigned in the final step of the configuration, and is used mainly to offset by one lag the negative portion of a cross correlation, in such a way that the zero lag correlation is not computed twice. The same mechanism allows for NOT computing negative autocorrelations in time multiplexed configurations.

The switches are assigned by the routine:

```
void calc_cube_states(int nx, int ny, int np, int nc, int ns,
        int perm[], int x_selector[], int y_selector[],
        int firstmodule, CELL state[MAXMODULE][8][8])
```

The meaning of the parameters is the same used in the previous chapters. The `perm[]` and `y_selector[]` arrays contain the permutation and external Y selector codes for each submodule in the configuration, computed in the previous step. One element is specified for each submodule, for a total of $\mathtt{np} \cdot \mathtt{nc} \cdot \mathtt{ns}$ elements. The `x_selector[]` array contains the external X selector codes for each cell row, for a total of 8 elements, equal for all modules.

The routine first initializes all the elements in the `CELL` array, and computes a base state with $\mathtt{nx} \times \mathtt{ny}$ correlators on each module. Then, for each submodule in a module, permutation code is applied and external Y connections are set according to the appropriate codes in `perm[]` and `y_selector[]` arrays.

Finally, for each row in a module, external X inputs are applied, row by row, according to the `x_selector[]` array.

The base state algorithm first sets the switches for a $\mathtt{nx} \times \mathtt{1}$ configuration. Then these long correlators are broken setting the switch for an external connection for the appropriate cells. For example, a 4×2 configuration is set first computing a 4×1 configuration, then breaking in two parts the four correlators.

External connections are set to a "undefined external" state in this process, since they will be set later on.

The permutation routine applies the permutation code, described in chapter 4.1, to the appropriate cells. The code applies only to connections between adjacent columns, as described there.

The external connection assignment routines look for "undefined external" switch states, and set them according to the codes in the `x_selector[]` array (X inputs) or to the codes in table 3

6 Delay determination

At this point, the correlator configuration is defined, but in a rather implicit way. It is impossible, for example, to identify the correlator and the delay range for each cell.

The routine:

```
void calc_trace_array(int sampl_status[2],
        int firstmodule, int nmodule,
        int tmf, int maxmodule,
        CELL state[MAXMODULE][8][8])
```

computes all the relevant informations for each cell, so that it can be used later on.

The routine scans the specified range of `nmodule` modules, starting with module `firstmodule`, looking for a cell with the X input connected to a sampler. The sampler and initial delay is retrieved, using informations in the `sampl_status[]` array, and the cell thread starting there is followed in forward until a strap connection, or another sampler, is found. In the process, the delay corresponding to the first stage of each crossed cell is stored in the CELL object, together with the sampler informations for that thread.

When the forward scan is completed, the process is repeated, this time looking for cells with the Y selector connected to a strap connection or to a sampler. For strap connections, the cumulative delay and the connected sampler for that strap is reported. An extra delay is inserted if: a) the delay on the sampler connected to the X input is higher than that of the sampler on the Y input; or b) the sampler connected to the X input has a higher code number than that on the Y input, and the delays are equal.

In this way, no negative autocorrelations are computed, and the zero lag correlation is computed for correlator $a \times b$ only if $a < b$.

The routine then scans backward the correlator thread, and on each step it computes the effective correlation delay computed by the chip and the chip control word.

All delays are computed in units of the final correlation step, i.e. `tmf` times the shift clock. For example, if `tmf` = 2 and the shift clock is 40 MHz, each delay step is 12.5 ns.

The chips in some columns output the signal with one less delay than usual, to compensate for external syncronization stages that are present on the module inputs. These anticipated delays are set by jumpers on the board. On most board configurations, the X outputs are anticipated for columns 3 and 7, and the Y outputs for column 0. If the signal does not exit (and reenter) the board after these columns, the anticipated delay must be compensated by adding an extra delay in the following cell in the thread. This task is performed both during the forward and the backward scan, using a configuration vector defined in the include file.

7 Cell parameter retrieval routines

According to the object oriented programming philosophy, the "object" is never manipulated directly. This rule has the important effect to make the informations in the object effectively "read-only" outside the restricted number of routines that must modify them. The exact content of the structure is also irrelevant to the general programmer, and must be kept separate (the *data encapsulation* concept).

The only informations that are important for the programmer are:

- the hardware control word that must be written to the chip, returned by the function `int get_cell_cword(CELL *state)`.

- the interval of delays correlated by the chip, returned by the function `void get_cell_offset(CELL *state, int *offs, int *doffs)`
- the sampler codes for the signals correlated by the chip, returned by the function `void get_cell_samplers(CELL *state,int *xsampl, int *ysampl)`

All these routines accept as the first parameter a pointer to the cell to be queried.

`get_cell_cword()` returns the 16-bit value to be written in the hardware control register. Only the low 16 bits are significant, the remaining are usually unset.

`get_cell_offset()` returns in `offs` the offset of first correlation product computed by the chip in its channel 0, ad in `doffs` the offset increment between successive correlation products.

This offset may be (and usually is) negative, i.e. the first correlation product is the most delayed of those computed by the chip. Its absolute value is equal to the time multiplexing factor. For time multiplexing factors, more than one chip compute the same delay interval. The integration routine must therefore *sum* the chip content to the integration memory, rather than copying it. In this way, the products for the same delay are automatically summed together.

Delays are always positive. For cross correlations, negative delays are "folded back" after positive ones, as normally required by Fast Fourier Transform algorithms.

For autocorrelations, the number of delays computed is given by the expression $N_{max} = (1024 \times \texttt{nmodule})/(\texttt{ncx} \times \texttt{tmf})$. For cross correlations, the number of delays for autocorrelation products is $N_a = (1024 \times \texttt{nmodule})/(\texttt{ncx} \times \texttt{ncy} \times \texttt{tmf})$, and for cross correlations is $N_x = 2N_a$. The delay indices range from 0 to $N_a - 1$ for autocorrelations and for the positive part of the crosscorrelations. The negative portion of the crosscorrelation function is stored at indeces from $N_A = N_x/2$ to $N_X - 1$, corresponding respectively to delays from $-N_x/2\tau$ to $-\tau$.

`get_cell_samplers()` return in `xsampl` and `ysampl` the codes for the two samplers whose signals are correlated. These are small integers, in the range 0–8, as described in chapter 2. The order is important for cross correlations. Positive delays refer to products where the first sampler is less delayed than the second.

8 Display and report routines

Once the configuration has been computed, it may be useful to print some informations in human readable format, or to show graphically the cell connections.

Three routines have been written for this purpose:

```
void report_status (CELL state[][8][8], int nmodule);
void report_cell   (CELL state[][8][8], int nmodule);
void graph_showcard(CELL state[][8][8], int maxmodule,
  char * title);
```

Each of these routines analyze the correlator described by the `CELL` array, and composed of `nmodule` modules, and show the relevant informations in an appropriate format.

```
Cell 0:0.0 - 0.0 x 1.1 - max. delay = 1:255 (2)
Cell 0:0.4 - 0.0 x 1.0 - max. delay = 0:254 (2)
Cell 0:2.0 - 0.1 x 1.1 - max. delay = 0:254 (2)
Cell 0:2.4 - 0.1 x 1.0 - max. delay = 1:255 (2)
Cell 0:4.0 - 1.0 x 1.1 - max. delay = 1:255 (2)
Cell 0:4.4 - 1.0 x 1.0 - max. delay = 0:254 (2)
Cell 0:6.0 - 1.1 x 1.1 - max. delay = 0:254 (2)
Cell 0:6.4 - 1.1 x 1.0 - max. delay = 1:255 (2)
Cell 1:0.0 - 0.0 x 0.1 - max. delay = 1:255 (2)
Cell 1:0.4 - 0.0 x 0.0 - max. delay = 0:254 (2)
Cell 1:2.0 - 0.1 x 0.1 - max. delay = 0:254 (2)
Cell 1:2.4 - 0.1 x 0.0 - max. delay = 1:255 (2)
Cell 1:4.0 - 1.0 x 0.1 - max. delay = 257:511 (2)
Cell 1:4.4 - 1.0 x 0.0 - max. delay = 256:510 (2)
Cell 1:6.0 - 1.1 x 0.1 - max. delay = 256:510 (2)
Cell 1:6.4 - 1.1 x 0.0 - max. delay = 257:511 (2)
```

Table 4: Example output of routine `report_status`

8.1 Correlator report

`report_status()` prints on the standard output a brief summary about the configuration, i.e. for each (non multiplexed) correlator in the system it prints delay range, input channels and length.

As an example, the output for a 2×2 cross correlator, with TMF= 2, and using two modules, is shown in Table 4.

In this example, for example, the third line means that cell in row 2, column 0 of the first module (module 0), computes the cross correlation of delay 1 of sampler 0 to delay 1 of sampler 1. The correlation is computed for delays from 0 to 254 in step of 2 delays.

Negative delays are "folded back" as it is usually done with conventional Fast Fourier Transforms. Thus autocorrelation channels have positive delays ranging from 0 to 255 delays, while cross correlations have a total length of 512 delays, the positive ones always in the range from 0 to 255, and the negative delays, from -256 to -1 delays, are stored at indices from 256 to 511.

8.2 Cell report

`report_cell()` prints for each cell: the input channels, the delay range computed, whether extra delays have been used, and the hardware control word.

With the same example used in chapter 8.1, the output of `report_cell` would be that showed on Table 5. Only 10 cells were actually shown, since the complete report is 130 lines long.

The first line, for example, shows the status of the first cell in the system. It computes the cross correlation of samplers 0 and 1, in the delay range from 255 (channel 0 of the chip) to 225 (channel 15), in steps of -2 delays. No extra delays are inserted before the inputs, and the control word for the chip is `0xfc02`, corresponding to the binary `1111 1100 0000 0010`.

As explained in the previous chapter, negative delays are stored with positive indices, as customary for FFT algorithms.

```
Cell   In   Delay range  Extra delays Cntl word
m:r.c  s-s  frst:last(d)  x     y     (hex)
0:0.0  0x1   255:225 (-2)  off   off  fc02
0:0.1  0x1   223:193 (-2)  off   on   fc80
0:0.2  0x1   127:97  (-2)  off   off  fc00
0:0.3  0x1    95:65  (-2)  off   off  fc00
0:0.4  0x1   254:224 (-2)  off   off  fc02
0:0.5  0x1   222:192 (-2)  off   off  fc00
        (118 lines omitted here)
1:7.5  0x1   353:383 (+2)  off   off  fc04
1:7.6  0x1   448:478 (+2)  on    off  fc40
1:7.7  0x1   480:510 (+2)  off   on   fc8c
```

Table 5: Example output of routine `report_cell`

8.3 Graphic output

`graph_showcard()` displays in graphic format the cell interconnections. For each module a complete display, showing cells, input connections, extra delay added, and input samplers connected to each X and Y external input is shown.

Each cell is represented as a square box, with the X input on the left side and the Y input on the right side. Along the left edge of the card are aligned the X external inputs, and along the bottom the Y external inputs. In each pair of external inputs, the uppermost (leftmost) is X0 (Y0), and the other is X1 (Y1).

Cascade external connections run "in between" the X0 and X1 blocks, and on the right edge of the card outline.

Each cell is connected to another one, or to an external input, by solid lines. When these lines connect two cells, they represent both the X and the Y connection, that must be the same for the correlator to operate correctly. The routine actually draws two superposed lines, so that a configuration error may show up as a garbled connection pattern (more than one line for a given input).

When an extra delay is used for a cell, the corresponding input is marked with a small square. The square is placed at the left side if it is inserted at the X input, and at the right side of the cell if inserted at the Y input.

In the postscript version, the routine annotates each used external line with the sampler number and the sampler delay present on that input. This is especially useful for strap connections, in which it is difficult to follow the correlator thread back to the original input. A title, specified in the last parameter, is printed above the graph, together with the module number.

An example of the output of the routine, always with the parameters specified in paragraph 8.1 is shown in fig. 8.3.

References

[1] A. Bos (1989): *The N.F.R.A. Correlator Chip,* NFRA Internal Technical Report 176

[2] A. Bos (1989): *A General Purpose Correlator Board: Functional Description,* NFRA Internal Technical Report 178

Figure 1: Example output of routine `graph_showcard`

[3] R.P. Millenaar, S. Zwier and A. Bos (1990): *The General Purpose Correlator Board: Hardware description,* NFRA Note 544

[4] G. Comoretto, M. Catarzi, M. Felli, F. Palagi, G. Tofani (1990): *Basic design for a 2048 channel autocorrelator based on the NFRA chip,* Arcetri internal report n. 9/90

Contents