LARG-NO-77
June 25, 1997

A proposal of serial protocol for the LAr calorimeter of ATLAS

O. Le Dortz, P. Nayman
LPNHE, Universités de Paris 6 et 7, IN2P3-CNRS

Abstract

The electronics of the Liquid Argon calorimeters of ATLAS located in the detector area needs to be controlled externally by means of a serial communication. For this purpose, we propose an implementation of the WorldFIP fieldbus protocol. The slave front-end stations of the serial communication will be designed as radiation tolerant ASICs.

This note describes the WorldFIP protocol and its adaptation to the LAr electronics needs. It also points out the performances that can be reached with this system. For example, at a symbol rate of 2.5 Megabits per seconds, the useful data rate can reach about 2 Megabits per seconds, and the configuration of the whole calorimeter can be completely written and read back in about 3 seconds.

This document is available at http://www-lpnhep.in2p3.fr/atlas/gb/elec/

Contents

1 Introduction

Each front-end crate for the Liquid Argon calorimeters of ATLAS contains a *controller board* that achieves the following functions [1]:

- It receives the timing and trigger signals from the TTC system in the trigger cavern and distributes them to the front-end crate, with the appropriate programmable time delays for each *front-end* and *calibration* board. This function will not be covered in this note.
- It is connected, through a 300 meter-long optical link to the *system controller* in the VME readout crate. This link enables to configurate and control all the parameters required by the front-end electronics.

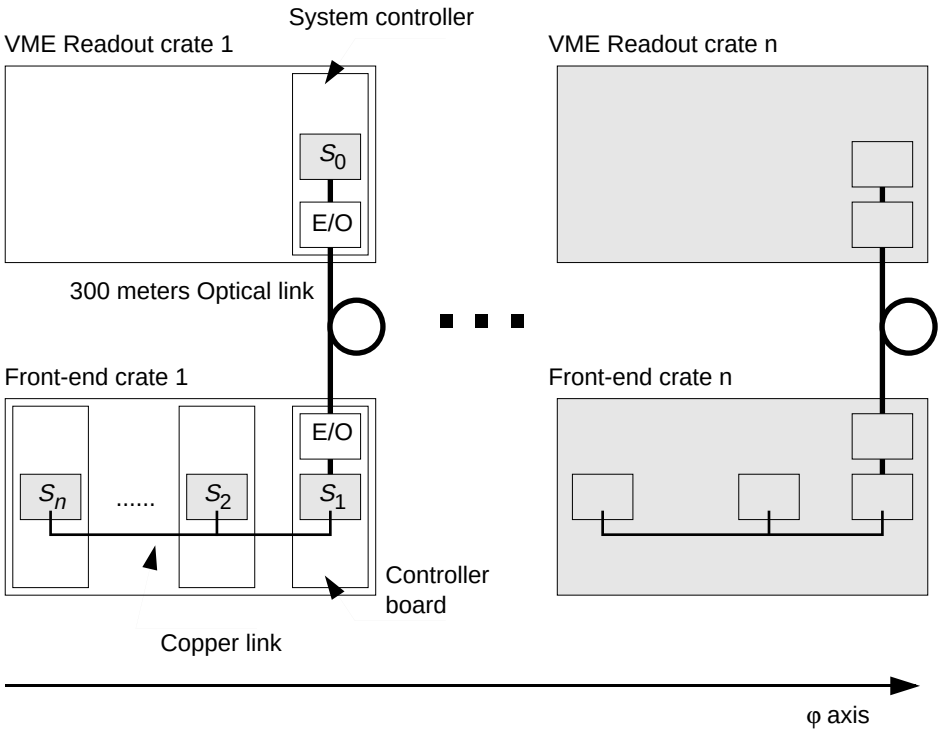

Figure 1: general architecture of the serial communication network

We propose a serial protocol that fulfills the second function, from the system controller in the readout crate to the electronic boards in the front-end crate (front-end boards, calibration, tower builder...). The serial communication is made of two branches, one optical branch from the controller board to the system controller, and a second local copper branch within the front-end crate from the controller board to the electronic boards. Figure 1 shows a scheme of the front-end and readout crates. The system controllers are interconnected (through VME, ethernet link...) and supervised by a high level master monitor not shown in this figure.

1.1 Requirements of the serial communication

- The components handling the serial communication in the front-end crate will be exposed to radiation levels (about 10^{12} neutrons/cm^2 and 20 Gy per year [2]) requiring a radiation-hardened technology.
- No electrical activity is authorised during data taking periods. In other words, the copper branch of the serial link presented previously needs to be completely quiet during data taking.
- The serial interface on the electronic board should use a clock derived from the Bunch Crossing clock, and not a private clock source, to minimise interferences.
- The serial protocol should enable read-write point to point register or memory accesses, and broadcast write operations on electronic boards of the same type.
- Care should be taken in defining the physical and the logical features of the protocol, for a better reliability, in terms of immunity to electrical noise and data integrity.

1.2 Generalities about the proposed protocol

We propose to develop an implementation of the serial protocol *WorldFIP* [3] that will comply with those requirements and that, moreover, provides the following features:

- It is a field bus. Robustness of the protocol is therefore guaranteed. The physical layer operates with very good electrical noise immunity (compliant with the EMC recommendations of the IEC organisation). For example, this physical layer is more reliable than RS485 twisted-pair based standard.
  The logical layer contains a mechanism to check the integrity of the received data.
- As a standard in industry, some 'off the shelf' hardware or software units can be used.
- It has the advantage to require only one signal pair carrying simultaneously the clock and the data.
- The physical layer enables data rates up to 2.5 Megabits per second.
- It is a deterministic protocol. Therefore, the limits of response time of the system are clearly stated.
- It is one of the three fieldbuses recommended by CERN for the LHC project [4].
- The same protocol is used for the optical and for the copper branches. This simplifies the electro-optical interface.

2 Description of the protocol standard

A WorldFIP network is made up of *stations* with two types of functions:

- bus arbitration, management of access to the transmission medium,
- production/consumption of data.

Only one station at a time can be the arbiter of the link. The arbiter asks for data (*variables*) to be transmitted on the line, by emitting the label of the variable; the unique *producing* station of the variable sends its value on the link; the *consuming* station(s) then just pick the variable value.

Every station can be either the producer of a specific variable or one of the consumers of this variable. For example, a station can be configurated to produce a variable V_1 and consume the variables V_2, V_3, V_4.

The protocol is made of three OSI communication layers:

1. the physical layer,
2. the data link layer,
3. and the application layer.

The aim of this section is not to describe completely the WorldFIP standard. However, we will describe the part of the standard necessary for our needs, namely the physical and a part of the data link layer.

2.1 The physical layer

The bits transmitted by the physical layer are coded using a Manchester code. This codes makes it possible to transmit simultaneously the data and its attached clock on the same line. It enables to transmit four symbols: logical one, logical zero, positive violation (V^+) and negative violation (V^-). The violation symbols are only used to specify the beginning and the end of the data frames. The waveforms of these four symbols are shown in figure 2.

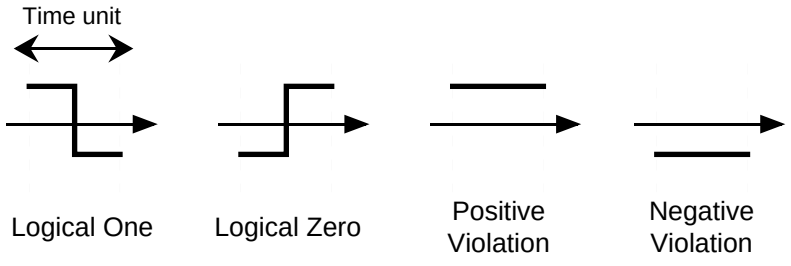

Figure 2: Manchester coding

Each WorldFIP frame is composed of four parts:

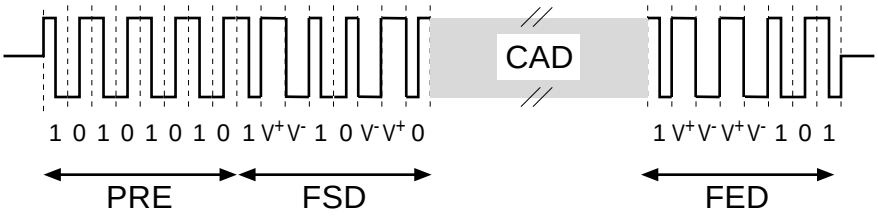

Figure 3: example of a WorldFIP frame

1. A *preamble* (PRE), used to synchronise the receivers clock: it is a sequence of eight symbols, '10101010'.

2. A *frame start delimiter* (FSD): this eight symbols sequence indicates the beginning of the control and data part.

3. *control and data part* (CAD). This field, containing the useful information provided by the data link layer, exclusively consists of logical one and logical zero symbols.

4. A *frame end delimiter* (FED): this eight symbols sequence indicates the end of the CAD part.

A scheme of a WorldFIP frame is shown on figure 3. The different fields are clearly visible and can be easily inspected with a digital oscilloscope or analyser.

The physical layer adds 24 symbols to every transmitted frame. Therefore, the longer the CAD field, the better the efficiency.

2.2 The Data Link layer

The WorldFIP protocol is based on a Producer-Consumer model. The data exchanged between the producers and the consumers are *buffers* (registers, memories). A buffer is identified uniquely by a logical label. We will adopt further in this note the WorldFIP terminology: a buffer will be called a *variable*; the logical label attached to a variable will be called its *identifier*, coded as a 16-bits word.

The exchanges of variables between the producers and the consumers can take place either cyclically or upon explicit request. We will only use variables exchanged cyclically because it offers more simplicity, more speed and the process is deterministic. We will explain in 3.2 how this cyclic mechanism can be used, in our case, for asynchronous commands on the electronic boards.

The data link layer specifies the format of the CAD field of the physical layer (figure 4). It adds to the useful data one *control* byte at the beginning of the CAD field that determines the type of this frame, and a two-bytes *Frame Check Sequence* (FCS) to verify the integrity of the frame.

The FCS is the result of a polynomial division of the sent data by a generator polynomial. It enables to detect errors in the received frame, with a very

high level of confidence. The probability to consider a corrupted frame as a valid one is about 10^{-15}. The encoding and decoding algorithms can be easily implemented on the fly by hardware, as explained in appendix A.

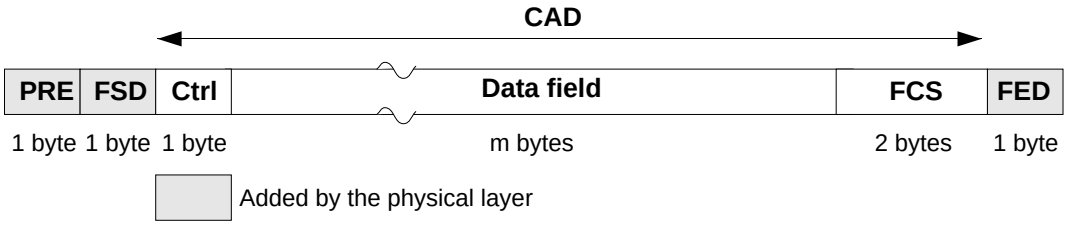

Figure 4: format of a WorldFIP frame detailing the CAD field

2.2.1 Principle of the exchange of variables

The bus arbiter decides when a variable must be transmitted. It emits a frame of type ID_DAT containing the identifier of the variable and starts a timeout counter. In consequence, the producer of the variable sends its value in a frame

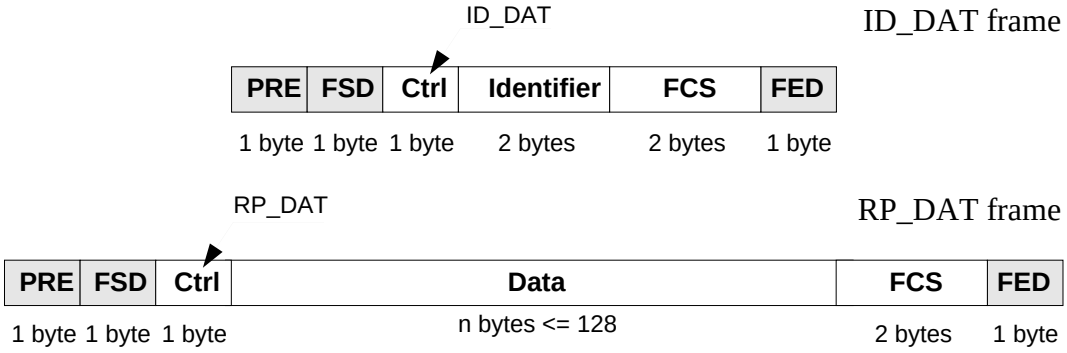

Figure 5: format of variable transfer frames

of type RP_DAT. The consumers of this variable read the frame emitted by the producer. If the producer of the variable has not put the data on the line after a certain delay, the arbiter warns the upper layers that a timeout error occured. A timeout error does not lock the system. With this scheme, the protocol enables the stations to be turned off or turned on again without deadlock.

The format of the ID_DAT and RP_DAT WorldFIP frames are detailed in figure 5. The data field of a ID_DAT frame only contains the two bytes of the identifier. The data field of a RP_DAT frame contains the value of the variable. The size of a variable is limited to 128 bytes.

The norm specifies the value of the control byte in case of ID_DAT or RP_DAT frames as indicated on table 1. Let us point out that only the six

least significant bits of the control byte are necessary to identify ID_DAT and RP_DAT frames.

Type	Binary value MSB sent first MSB LSB
ID_DAT	xx000011
RP_DAT	xx000010

Table 1: values of the control byte

2.2.2 Notion of macrocycle

At initialisation, the bus arbiter is assigned a list of cyclic variables to scan and the periodicities associated with each of these variables. Given this list, a periodic sequence called *macrocycle* is performed by the bus arbiter. Each element of the sequence, called an *elementary cycle*, enables to scan one or several variables. Details about the definition of the process in our case is described in 3.2.

As an example, let us consider an application where two variables A and B are used. The variable A needs to be updated every time unit, while the variable B needs to be updated every 2 time units. The macrocycle scanned by the bus arbiter will contain two elementary cycles of duration 1 time unit each:

1. transfer of variable A:
   - the arbiter emits the identifier A in a $ID\_DAT(A)$ frame,
   - the producer of A puts its value in a $RP\_DAT(A)$ frame.
2. transfer of variable A and then B.

Thus, the two variables will be updated at the appropriate periodicities. The process is completely deterministic. If the arbiter encountered a timeout error when it initiated a request, the macrocycle continues. The concerned variable will just be requested again in the next macrocycle (the duration of the macrocycle is always the same).

3 Application to the LAr electronics

3.1 Architecture of the network

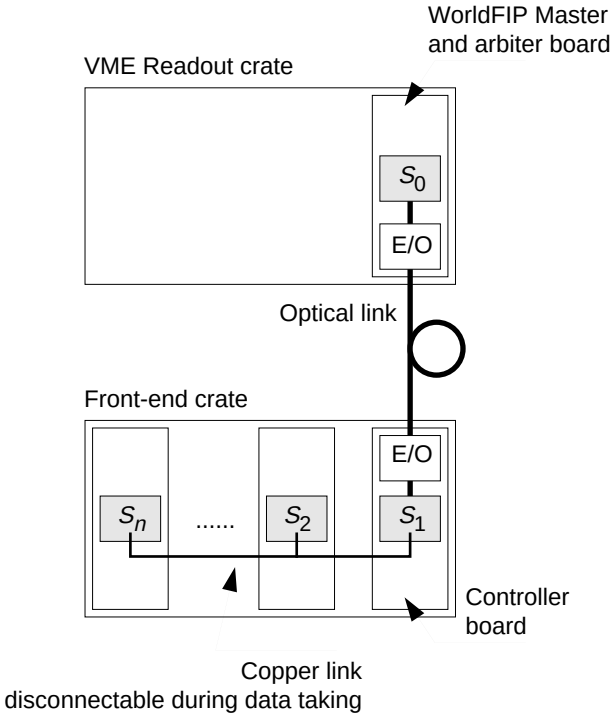

Figure 6: architecture of the network

We propose to develop WorldFIP stations inside the front-end crate fully compliant with the physical and data link layers specifications. This ensures the ability to use industrial products on the network (stations located in the VME readout crate).

The architecture of the proposed network is sketched in figure 6. The arbiter and 'master' station S_0 is housed in the VME readout crate. In the front-end crate, each board requiring a serial communication contains a 'slave' station S_i, where $i > 0$. The electro-optical interface from the top-down fibre link to the on-detector copper link is implemented in the controller board. The 'slave' station S_1 inside the controller board also plays the role of *gateway* between the two branches. Upon request of the master, the gateway will connect or disconnect the copper branch, to avoid traffic during data taking.

3.2 The user layer

Our user layer determines, in the data field of RP_DAT frames, a user protocol of exchange of data (board registers, memories...). It also uses the periodic service provided by the data link layer to propose to the user a service of asynchronous access to the electronic boards.

The 'master' station S_0 will produce the *Command* variable *COM* that contains orders such as read/write a given register/memory in a given board or group of boards.
The 'slave' stations S_i in the front-end crate will consume the Command variable COM and produce one or several *Answer* variables *ANSi* that contain the registers/memory values read from the board and/or status.
The arbiter (in station S_0) will scan a macrocycle consisting of one Command and the Answer(s).
We consider the possibility to use only one Answer variable for all the electronic boards or one per board. Examples are provided in section 4 and a discussion about the consequences of each option is presented in the performances section 5.

Each board in the front-end crate provides to the serial ASIC a hardwired *board number* and a *group type* (crate controller, calibration, tower builder, different front-end boards...).

3.2.1 The Command variable

The COM variable is set to have the following features:
Identifier: 0000 Hexa
Size: 128 bytes.
The different fields composing the COM variable are defined in table 2[1]. A command can contain one or several register access requests or a memory access request.

In case of register access, a register block consists of three consecutive bytes: the address of the register and the attached 16-bits data.

In case of memory download, a 4-bytes header is sent, followed by the memory data.

Description of the fields

- *Command Counter*:
  for each new command, the user increments this counter; the slaves ignore the command if this counter does not increment. This mechanism enables to do asynchronous control of the boards. The counter value will also be copied into the Answer frames sent back by the slaves, for cross-checking.

- *Slave address/type*:
  the most significant bit is the broadcast bit.

  - If it equals '1', the command will affect, in broadcast mode, all the stations of type matching the 7 least significant bits of this byte.
  - If it equals '0', the command will concern the station whose board number matches the 7 least significant bits of this byte.

[1] MSB sent first on the line

Byte	Field description	
0	Command Counter	
1	Slave address/type	
2	Function Code	
	Register Access	**Memory Access**
3	Number of registers	Number of bytes
4	Register 0 Address	NTA address
5	Register 0 high byte	NTA value high byte
6	Register 0 low byte	NTA value low byte
7	(Register 1 Address)	Write Data 0
8	(Register 1 high byte)	Write Data 1
9	(Register 1 low byte)	Write Data 2
...	...	...
124	(Register 40 Address)	Write Data 117
125	(Register 40 high byte)	Write Data 118
126	(Register 40 low byte)	Write Data 119
127	Not used	Write Data 120

Table 2: COM variable format

- *Function Code*: the type of access to be performed.
  Bit 7 Read operation
  (from slave in front-end crate up to the master)
  Bit 6 Write operation
  (master down to slave)
  Bit 5 '1' for Memory access
  '0' for Register access

  Both bits 7 and 6 set performs a write operation followed by a read operation at the same address.

Then follow, either register blocks, or the memory block. In case of register read-only access, the *write register data* fields of the register blocks shall be ignored by the slaves.

Memory block

The memories are accessible by partitions of maximum size 121 bytes. Each memory is assigned a 16-bits *Next Transfer Address* (NTA) register on the board. The address of this NTA register is specified by the byte number 4 of the COM variable. Its value (bytes 5 and 6 in table 2) is the start address of the block to download from (or upload into) the memory. The data block size in bytes is given in byte 3.
In case of read-only memory access, the bytes 7 to 127 shall be ignored by the slaves.

3.2.2 The Answer variable

When a command implies a response from a slave, the answer is put by the slave in a Answer variable, upon request of the bus arbiter. The ANSi variables have the following features:
Identifier: $00xy$ Hexa
Size: 128 bytes.
where xy is the station number i.e. the hardwired board number provided by the board to the ASIC[2]. An Answer variable has basically the same format as the COM variable (see table 3), except for the Function Code which is replaced by a *Status Code*.

Byte	Field description	
0	Command Counter	
1	Slave address	
2	Status Code	
	Register Access	**Memory Access**
3	Number of registers	Number of bytes
4	Register 0 Address	NTA address
5	Register 0 high byte	NTA value high byte
6	Register 0 low byte	NTA value low byte
7	(Register 1 Address)	Read Data 0
8	(Register 1 high byte)	Read Data 1
9	(Register 1 low byte)	Read Data 2
...	...	...
124	(Register 40 Address)	Read Data 117
125	(Register 40 high byte)	Read Data 118
126	(Register 40 low byte)	Read Data 119
127	Not used	Read Data 120

Table 3: ANSi variable format

- *Status Code*: type of data in the following fields.
  Bit 7 the data have been read from the board
  Bit 6 the data have been written to the board
  Bit 5 '1' : the data are a memory block
  '0' : the data are register blocks
  Bits [4-0] information about the transmission result (No error, FCS error,etc.)

  Both bits 7 and 6 are cleared when the transaction is just an acknowledge. This occurs when the command counter did not incremented from the previous command to the present one.

In the two-variable option, the ANS variable is produced by the unique target station implied in a point-to-point request. But in case of broadcast, the ANS variable will be produced by the controller board.

[2]or FF if a two-variables only option is used.

4 Concrete examples

4.1 Peer-to-peer register access, two-variable option

4.1.1 Write operation

Figure 7 shows an example of a two-variable macrocycle in which one writes into the front-end board number 9 the data 'FEC4' into the register of address '02' and 'ABCD' into the register '1A'. The transaction is composed of the four

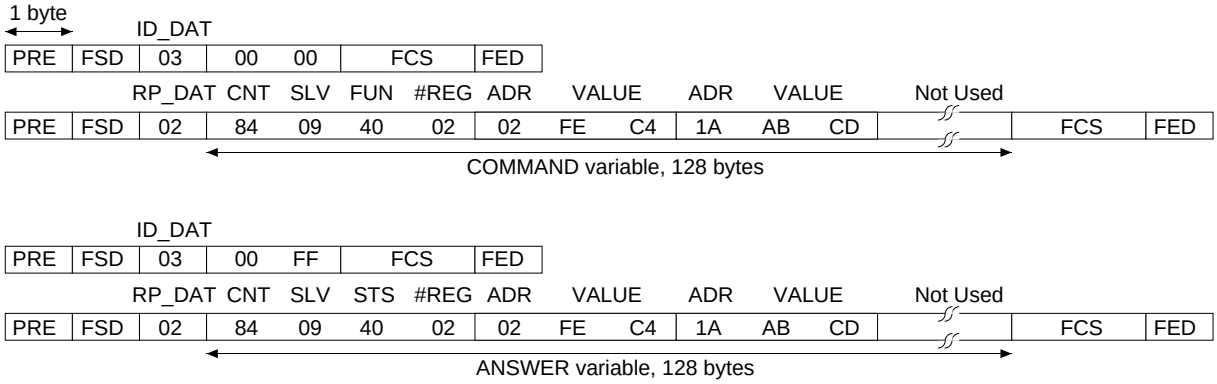

Figure 7: example of peer-to-peer registers write operation

following frames:

1. the arbiter requests the COM variable by emitting a frame ID_DAT(0000).
2. After a turnaround time, the master station writes the RP_DAT frame containing the COM variable value, containing:
   - the command counter; the previous value of the command counter was 83. It is incremented to announce a new relevant COM variable.
   - The target slave address; here one accesses the board number 09.
   - The function code; 40 for write registers.
   - Number of registers; 2 registers are to be written.
   - Address and value of the first register
   - Address and value of the second register
   - The other bytes are set to an arbitrary value. However, these bytes are considered in the final FCS calculation.

   The slave station 09 recognises that the command is new and that it is the target. It therefore interprets the command into a sequence of registers write into the board.
3. Then, the arbiter, after a turnaround time, requests the ANS variable by emitting a frame ID_DAT(00FF).

4. As the station 09 knows that it was the unique target station of the command, it produces the answer variable. It sends the command counter, its station address, the function code 40 (the following bytes are registers written into the board), and the registers blocks.

4.1.2 Read operation

Let us now consider the following macrocycle, where one wants to read back the registers previously written. Figure 8 illustrates the transaction. The process

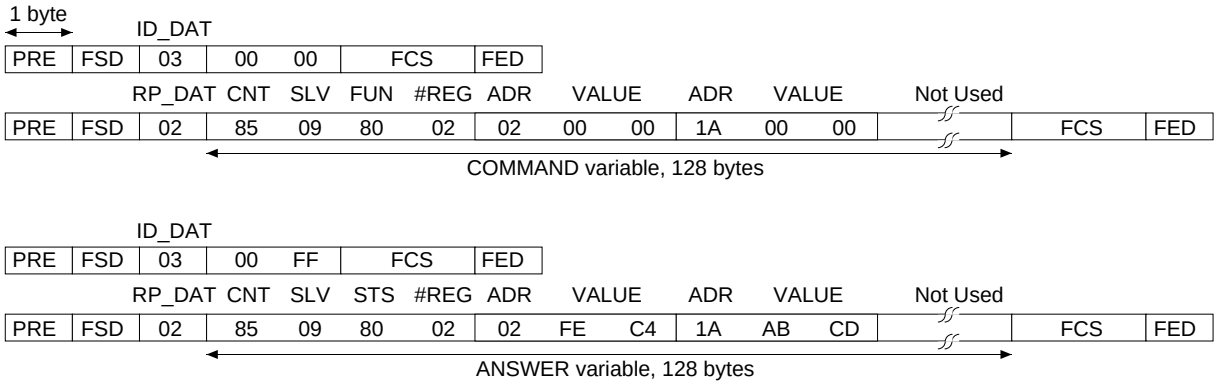

Figure 8: example of peer-to-peer registers read operation

is the same as in the previous example, but there are several differences

- the master station increments the command counter to announce that it is a new one. Here, its gets 85.
- The function code gets 80, for 'read registers'.
- The values fields of the register blocks, in the command variable, are useless.
- The target station 09 returns the ANS variable, containing the status code 80 and the registers values read from the board.

Then, follow several 'idle' macrocycles, where the command counter is not incremented by the master station. The slaves just ignore the command. The last target station 09 just produces answer variables with function code 00, indicating that nothing was done on the board.

4.2 Peer-to-peer memory access, two-variable option

We now consider a transaction where one wants to write a 16-bytes block of memory into the front-end board number 0A, at the memory offset '1C00'. Let also assume that NTA register of the memory is the register of address 'F0'. Finally, we also want to read back the contents of the memory within the same

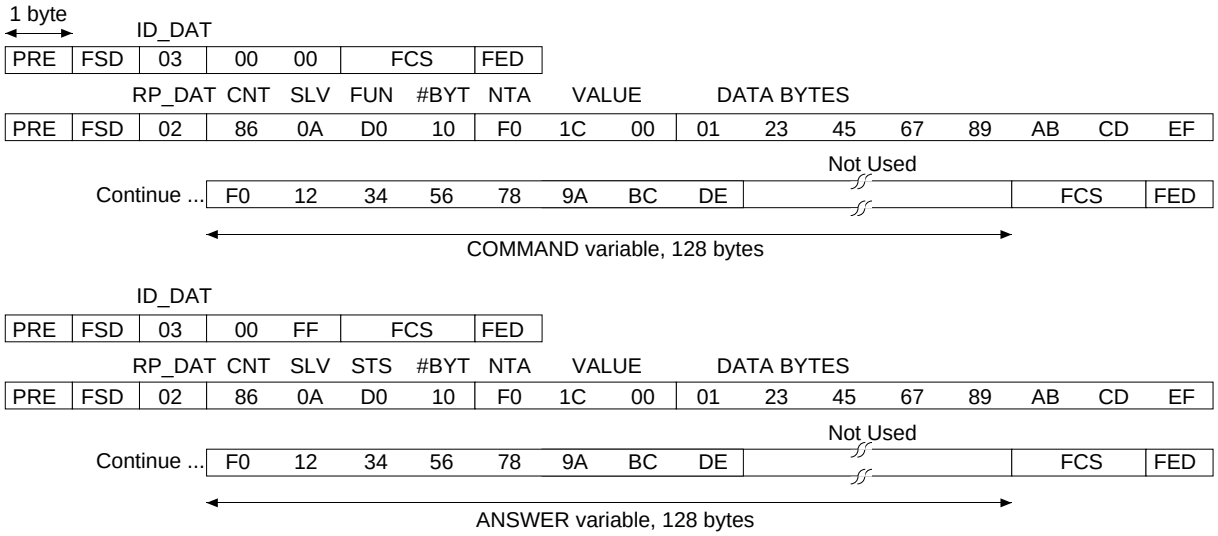

Figure 9: example of peer-to-peer memory write and immediate read access

macrocycle. Figure 9 shows the transaction. The macrocycle is composed of the four following frames:

1. the arbiter requests the COM variable by emitting a frame ID_DAT(0000).
2. After a turnaround time, the master station writes the RP_DAT frame containing the COM variable value, including:
   - the command counter, incremented to indicate a new command,
   - the function code; D0 for write + read memory,
   - the number of bytes of the memory block to write/read,
   - the address of the NTA attached to the desired memory,
   - the value of the NTA register: we want here to load the memory, beginning at '1C00',
   - the 16 bytes to write into the memory, beginning at offset 1C00.

   The slave station 0A gets the command, sets the NTA register to the offset value and writes the memory on the board.
3. Then, the arbiter, after a turnaround time, requests the ANS variable by emitting a frame ID_DAT(00FF).
4. The target station 0A sets again the NTA to the original offset 1C00, reads the memory contents back and sends the result into the RP_DAT frame.

4.3 Broadcast examples

4.3.1 Broadcast write, two-variable option

Figure 10 shows an example of broadcast write operation, when using the two-variable option. In this case, one wants to upload the memories of all the boards of type 2. The macrocycle contains four individual frames, as usual. These are

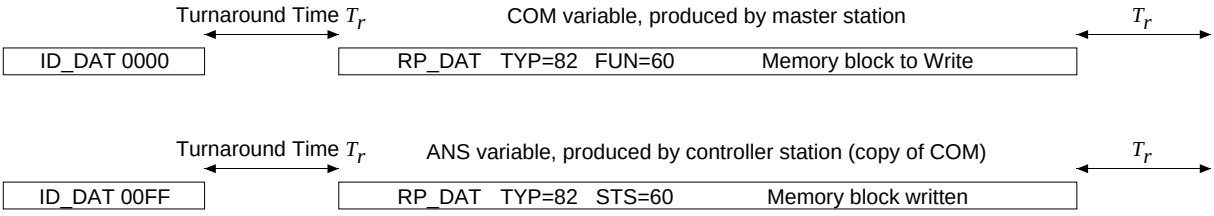

Figure 10: example of broadcast write macrocycle, two-variable option

the information specific to the broadcast mode:

1. The slave address field of the command contains 82, for a broadcast mode towards the stations of type 2.
2. The function code is 60, for memory write.
3. The target stations of type 2 will consider the command and write the memory block on the board. The others stations will ignore the command.
4. As several target stations might produce the answer variable, it is stated that none of them produces it. In broadcast mode, the controller station produces the answer variable. In this case, the controller station just copies the command variable into the answer variable.

4.3.2 Broadcast write + read, multiple-variable option

Figure 11 shows an example of multiple-variable macrocycle, where a new command of broadcast write + read memory is requested. Let us assume that 16 variables (i.e. 16 boards) take part of the macrocycle.

1. The arbiter first requests the command variable, that is produced by the master station.
2. The command variable is the same as in the previous example, except for the function code, that is 'D0' for memory write + read.
3. The target stations of type 2 recognise the command and achieve, on their boards, the write operation of the memory.
4. The arbiter sequencially scans all the ANSi variables.

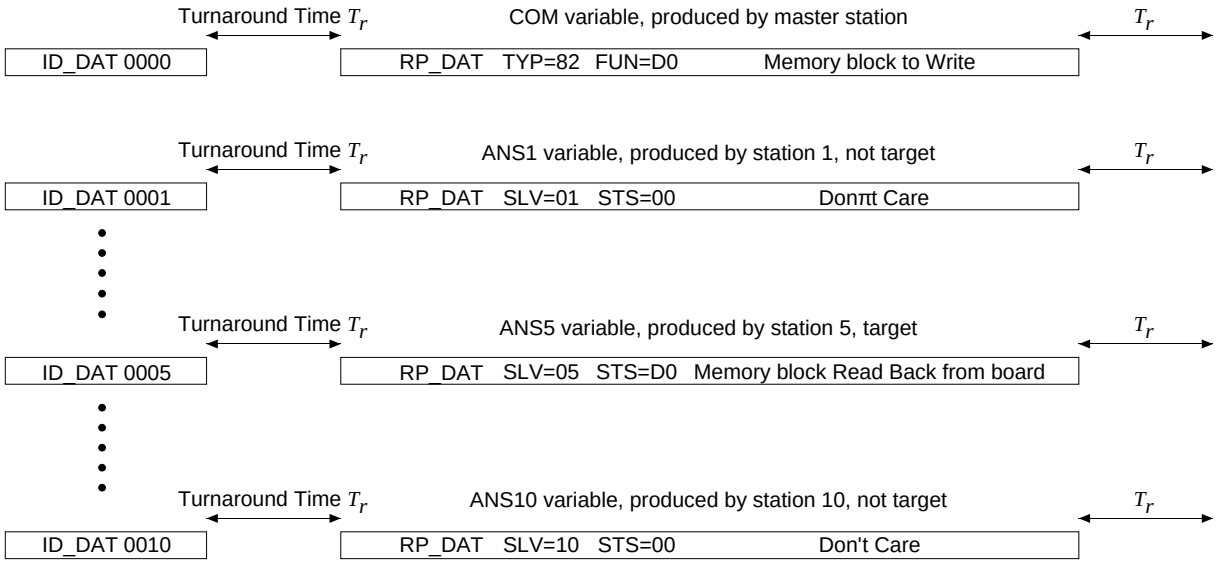

Figure 11: example of broadcast write and immediate read

- If the producer station of ANSi is not a target of the broadcast, its answer contains the status code 00, that means that nothing was achieved on the board.
- If the producer station of ANSi is one of the targets of the broadcast, it reads the memory back and returns, in the ANSi variable, the content of the memory on the board.

5 Performance considerations

The deterministic aspect of the WorldFIP protocol allows to estimate precisely the useful data rate accessible with the adopted architecture.

5.1 Variable transfer timing

We propose to use, for the whole network, a symbol rate of 2.5 Mbits.s^{-1}. Then, the time unit spent to emit a logical symbol (one out of the four provided by the Manchester coding system) is $T_u = 400ns$.
A complete exchange of variable consists in the emission, by the arbiter, of the ID_DAT frame and the production of a RP_DAT frame by the producer of the variable. The time between ID_DAT and RP_DAT is called the *turnaround time* T_r. The protocol specifies that:

$$10.T_u \leq T_r \leq 70.T_u.$$

In our case, taking account of the cable lengths, we can assume that:

$$T_r \simeq 10.T_u \simeq 4\mu s.$$

The total transaction time includes times needed for:

- the transmission of the ID_DAT frame (T_{ID_DAT}),
- the turnaround time,
- the transmission of the variable value in a RP_DAT frame (T_{RP_DAT}),
- the turnaround time.

An ID_DAT frame is 8-bytes long, thus $T_{ID\_DAT} = 64.T_u$.
As we will use 128-byte variables, a RP_DAT frame is 134-bytes long (figure 5): $T_{RP\_DAT} = 1072.T_u$.
The total time required to exchange one variable is:

$$64.T_u + 10.T_u + 1072.T_u + 10.T_u = 1156.T_u \simeq 460\mu s.$$

5.2 Comparison of the configurations

Two configurations of variables are considered. One option consists in using one answer variable ANSi for each board. Typically, about 15 boards per 'half-crate' will require a serial communication. In this case, the arbiter needs to scan, within a macrocycle, 16 variables. The other option consists in using only one answer variable ANS per 'half-crate'. In that case, the macrocycle is limited to two exchanges of variable. Each option offers different advantages, in term of speed performances, depending on the type of transfer (write broadcast, point-to-point access...).

The table 4 summarizes the specifications of the two options.

In this table, a block is 121 memory bytes or 1 to 41 individual 16-bits registers.

	16-variable option	2-variable option
Macrocycle duration	$7.3ms$	$0.920ms$
Broadcast Write Status of n boards	1 block in $7.3ms$ Available in ANSi	1 block in $0.920ms$ Not available
'Broadcast Read'	1 block in $7.3ms$	impossible
Individual Read or Write	1 block in $7.3ms$	1 block in $0.920ms$
Individual Read or Write of n boards same addresses	1 block in $7.3ms$	1 block in $n \cdot 0.920ms$

Table 4: comparison of the proposed options

In broadcast mode, the COM variable is asked by the arbiter (ID_DAT[0]) and produced by the master. The target slaves S_i consume this variable and write the broadcast data in their boards. Then, for the 16-variable option, within the same macrocycle, the arbiter scans the answer variables ANSi (ID_DAT[i]). Each slave, including the target slaves implied in the broadcast transfer, answers by producing their ANSi response. This way, it is possible to get the result of the broadcast operation in one macrocycle: the target slaves just put in their ANSi variables a status, or even the result from a read operation on their boards. The 'broadcast read' in the table 4 is the case when the slave stations put in their answer variables the data read back from their boards.

To summarize the table, as concerns write operations or point-to-point read operations, the 2-variables option is the fastest, in any case. However, the 16-variable option might be twice faster when performing broadcast write operations requiring a read back of the status or the data from the boards.

Anyway, whatever option is adopted, the time needed to transfer one register or a block of up to 41 registers is the same. In other words, the more registers are written/read on a board in the same transaction, the more efficient [3] the protocol is.

5.3 Useful data rates, realistic estimations

As an example, a front-end board could contain less than 40 individual registers and a memory (for FPGA programming) of less than 50 kilobytes [5].

5.3.1 Write mode

The 40 registers can be written on the board in a macrocycle, while the memory can be downloaded in about 415 partitions (each one taking one macrocycle). The time needed to download a complete board is 416 macrocycles, or:

- 382 ms in the two-variable option,
- 2.9 s in the 16-variable option.

[3] in terms of amount of useful data transmitted per time unit

When programming a whole front-end crate, it seems realistic to assert that the front-end board memories will be written in broadcast mode and the registers will be written in point-to-point access.
Given a number of front-end boards of about 15, the time needed to write data into the whole crate will be $415 + 15$ macrocycles, or:

- 396 ms in the two-variable option,
- 3.1 s in the 16-variable option.

5.3.2 Read mode

Let us consider now the case where we read back the data previously written. The time spent to read back all the registers and memory blocks of a front-end board is the same as the time spent in writting them. In the 16-variable option, the 'broadcast read' mode can be very useful to read back a whole crate, in the same macrocycles. Therefore, the front-end crate will be read back in the same time as the time needed to write it.
On the other hand, in the two-variable option, a point-to-point approach is necessary to read back a whole crate.
Finally, the time required to write a whole crate, and then to read it back is:

- $396\ ms + 15 \times 382\ ms = 6.1\ s$ in the two-variable option,
- $2 \times 3.1\ s = 6.2\ s$ in the 16-variable option.
- 3.1 s in the 16-variable option, when the broadcast write and then read in the same macrocycle option is used.

5.3.3 Summary

All these numbers are summarized in the table 5. The data rates estimations are based on the amount of *useful* data, namely:

- in write mode, 50 kbytes = 400 kbits of data sent in broadcast,
- in read mode, the data read back from each board, i.e. 15×50 kbytes = 6 Mbits.
- in write and read mode, the data sent + then data read back, i.e. 16×50 kbytes = 6.4 Mbits.

The best performance can be reached when using the 16-variable option, in 'broadcast write and immediate read' mode. In that case, a whole front-end crate can be configurated and checked in about 3 seconds. Provided that this operation is achieved in parallel for all the crates of the calorimeter, 3 seconds will be the time required to control and check the whole calorimeter.

	16-variable option	**2-variable option**
Memory simple access	132 kbits/s	988 kbits/s
Register simple access	90 kbits/s	713 kbits/s
Task		
write a crate	3.1 seconds 129 kbits/s	396 milliseconds 1.01 Mbit/s
write then read a crate	6.2 seconds 1.03 Mbit/s	6.1 seconds 1.04 Mbit/s
write and read in the same macrocycle the whole crate	**3.1 seconds** **2.06 Mbits/s**	

Table 5: performances summary

6 Implementation

We are designing an ASIC (figure 12) fulfilling the function of 'slave' station as described in section 3. It will be connected to the copper serial link and to a parallel port on the board (address, data and commands). It also receives from the board a hardwired board number and a board type (e.g. front-end board calibration board, tower builder, controller board, etc.). A general purpose output 8-bit register is also available. The controller board will implement the

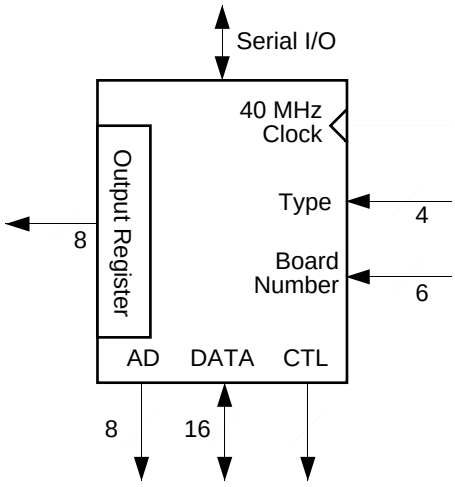

Figure 12: the slave WorldFIP ASIC

function of gateway between the optical link and the copper inner link. The state of the gateway will be controlled by one bit of the general purpose register of the controller board station ASIC. Thus, the copper link can be isolated from the external world upon request of the master station. The connections between the control station and the slave stations of the other boards are shown in figure 13.

Before the production of an ASIC prototype, a FPGA pre-prototype will be produced.

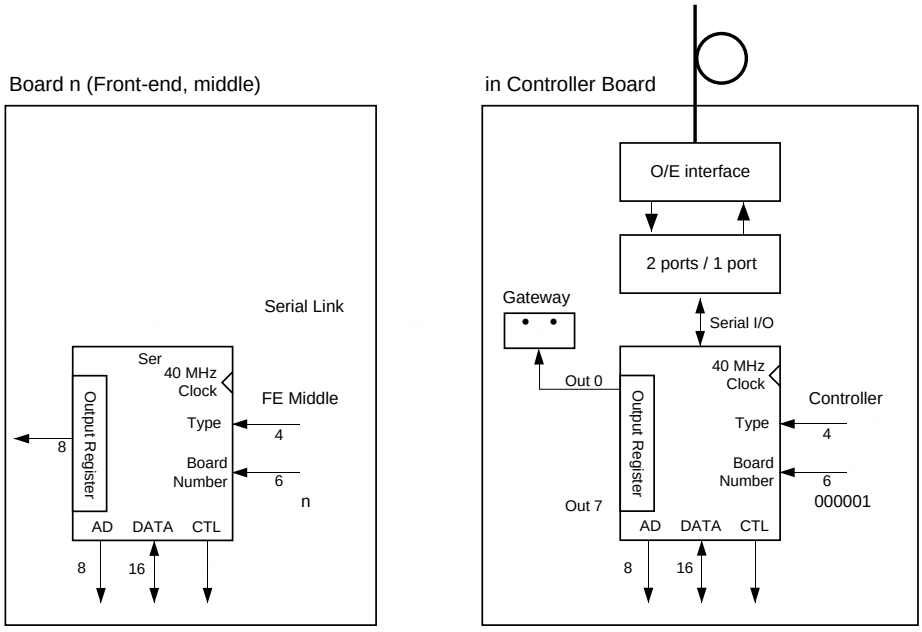

Figure 13: connections of the serial link in the front-end crate

References

[1] O.B. Abdinov et al., *ATLAS, Liquid Argon Calorimeter Technical Design Report*, CERN/LHCC/96-41, ATLAS TDR 2, 15 December 1996, Chapter 10, p.355-358

[2] *idem*, Chapter 11, p.467

[3] Information about the WorldFIP standard can be found at http://www.worldfip.org/

[4] The Working Group on Fieldbuses, *Recommendations for the use of fieldbuses at CERN*, CERN/ECP-DO, http://ecpcowww.cern.ch/fieldbus/report1.html

[5] Al. Gara, *private communication*

A Frame Check Sequence

We present here the principle applied to calculate the frame check sequence that is appended, in the data link layer (see 2.2), to the CTRL and DATA field of the frame CAD. The FCS is the result of a polynomial division by a *generator polynomial* of the input bit stream, composed of the CTRL and DATA bits of the CAD field.

A.1 Notations

Let d be the input k bits sequence on which the algorithm is to be applied. d_{k-1} is the first bit sent and d_0 is the last bit sent.
The encoding algorithm will append to the sequence d a frame check sequence f of l bits, where f_{l-1} is the first bit appended and f_0 is the last one. The complete sequence r will be n-bits long, such that $n = k + l$ and

$$r_{n-1}, r_{n-2}, \cdots r_{n-k}, r_{n-k-1}, \cdots r_0 = d_{k-1}, d_{k-2}, \cdots d_0, f_{l-1}, \cdots f_0.$$

Let us regard the bit sequences as polynomials $D(X)$, $F(X)$, and $R(X)$, such that:

$$\begin{aligned} D(X) &= \sum_{i=0}^{k-1} d_i \cdot X^i \\ F(X) &= \sum_{i=0}^{l-1} f_i \cdot X^i \\ R(X) &= \sum_{i=0}^{n-1} r_i \cdot X^i \\ &= \sum_{i=0}^{k-1} d_i \cdot X^{n-k+i} + \sum_{i=0}^{n-k-1} f_l \cdot X^i \\ &= D(X) \cdot X^{n-k} + F(X) \end{aligned}$$

A.2 Definition of the algorithm for WorldFIP frames

Encoding

In our case, the FCS is a 16-bits word. The resulting sequence r is the whole CAD field of a WorldFIP frame, and the input sequence d is the k first bits of the CAD field represented by the CTRL byte and the DATA bytes.

$$l = 16 \text{ and } n = k + 16.$$

The frame check sequence is defined as the remainder of the division of a polynomial derived from $D(X)$ by the generator polynomial $G(X)$:

$$F(X) = L(X)(X^k + 1) + D(X) \cdot X^{16} \text{ modulo } G(X) \qquad (1)$$

where:

$$\begin{aligned} G(X) &= X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^3 + X^2 + X + 1 \\ L(X) &= \sum_{i=0}^{15} X^i \end{aligned}$$

Decoding

The received encoded sequence, after transmission, $R'(X)$ can differ from the encoded sequence $R(X)$. A syndrom $S(X)$ is calculated on the received sequence:

$$S(X) = L(X) \cdot X^{k+16} + R'(X) \cdot X^{16} \text{ modulo } G(X) \tag{2}$$

If no error occured during the transmission, i.e. $R'(X) = R(X)$, the syndrom polynomial $S(X)$ is independant from the input sequence and given by:

$$S(X) = X^{15} + X^{14} + X^{13} + X^9 + X^8 + X^7 + X^4 + X^2 \tag{3}$$

A.3 Hardware implementation

An example of hardware implementation of the FCS calculation, as well as the encoding and decoding procedure are presented on figure 14.

Encoding

The input sequence is sent on 'Data Input'. The 16 D-latches are preset before the emission of the data to encode. When the input sequence is fed to 'Data Input' and 'IsData' is set, the outputs $\{\overline{Q}_{15}, \cdots \overline{Q}_0\}$ of the D-latches will contain the resulting FCS of the input sequence. When all the input bits have been provided, i.e. after k clock cycles, the result FCS can be retrieved

- either in parallel by reading, in one clock cycle, $\{\overline{Q}_{15}, \cdots \overline{Q}_0\}$ values,
- or sequentially, by resetting 'IsData' to inhibit the feedback action and reading, during 16 clock cycles, the output signal 'Serial Output'.

Decoding

The decoding process is equivalent to the encoding process. The received sequence is fed to 'Data Input', as for encoding. When all the $k + 16$ received bits (including the 16 FCS bits) have been provided to the circuit input, the 16-byte word $\{Q_{15}, \cdots Q_0\}$ is the syndrom sequence, and should respect the following conditions if no error occured during the transmission:

$$\{Q_{15}, \cdots Q_0\} = \text{ E394 Hexa .}$$

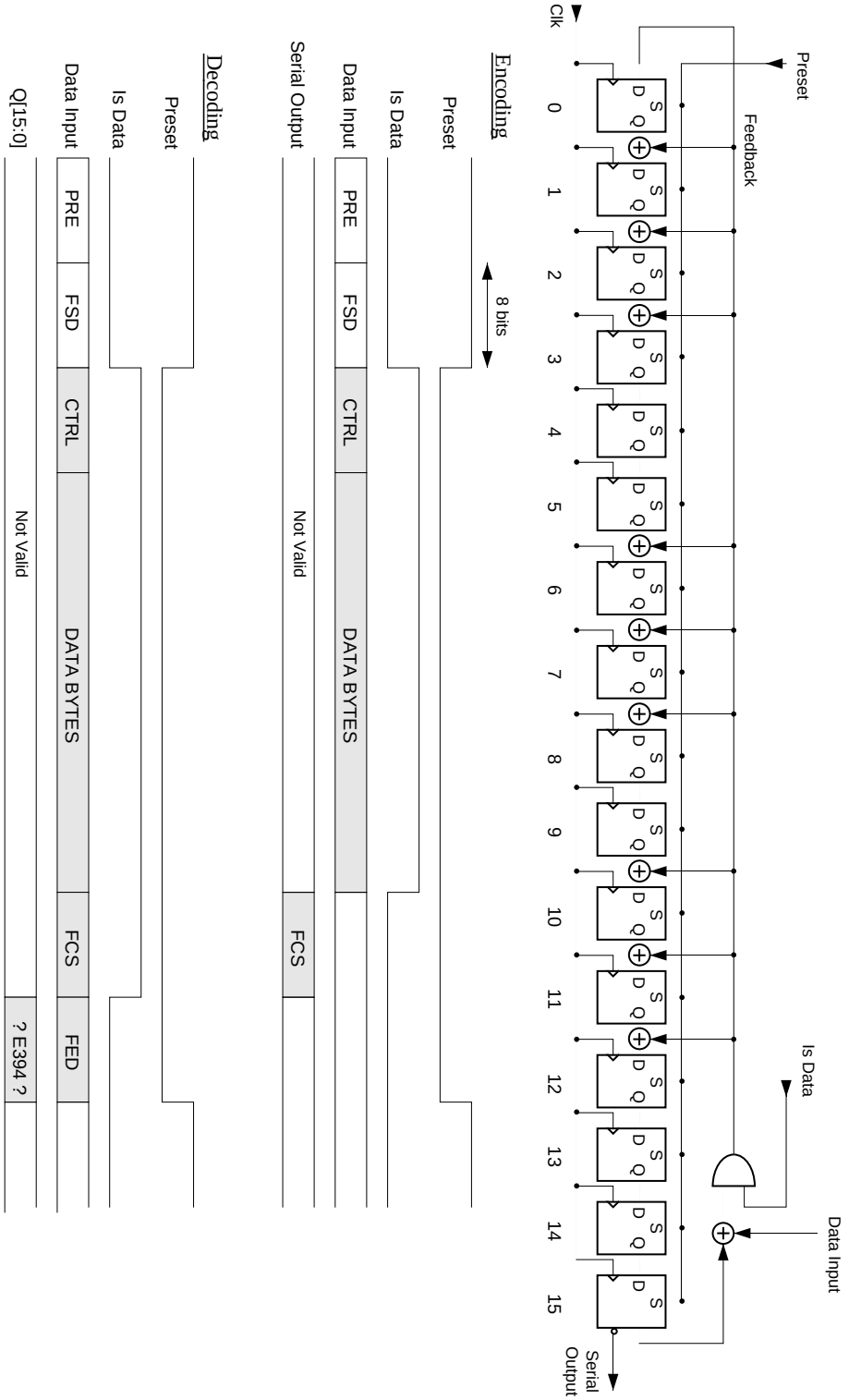

Figure 14: example of hardware implementation of the frame check sequence, and application to WorldFIP frames