STOCKHOLM UNIVERSITY

DEPARTMENT OF PHYSICS

SPECIFICATION AND DESIGN OF THE FERMI DIGITAL FILTERS

S. AGNVALL, C. BOHM and M. LJUNGGREN

USIP Report 97 - 06

May 1997

Specification and Design of the FERMI digital filters

M. Ljunggren, S. Agnvall, C. Bohm

Department of Physics, University of Stockholm, Box 6730,
S-113 85 Stockholm, Sweden

ABSTRACT

This paper describes the specification and design of digital filters usable for calorimeter applications in LHC experiments. The filters are used for energy and time measurements on sampled electronic pulses, providing information essential to the trigger and data acquisition systems. The optimization of the filter equations for a FIR filter and the corresponding architecture is important to limit latency in the first level trigger. Our solution is partly based on a scaling of the coefficient set for the filter. An architecture for an FIR-OS hybrid filter usable for high precision pulse height measurements is also described.

Introduction

The FERMI Multi Chip Module (MCM) uses digital filters to process data for the first- and second-level triggers (fig. 1). In the direct-trigger output, digitised and linearised data are summed and processed so that sample sequences corresponding to detector pulses are converted to amplitude values at a well defined clock transition. This means that there should be a fixed time interval between the pulse start and the amplitude report so that pulses originating from different bunch crossings are not confused. Since many of the FERMI applications require minimum latency for the production of trigger output, reducing the latency has been given a high priority (see appendix).

Determining when a pulse occurred, i.e. the bunch crossing identification (BCID), is a crucial operation. The method chosen is to detect maxima in a suitably filtered data stream, and when this happens extract amplitude information from a second digital filter. The choice of filter coefficients should be programmable so that the FERMI module can be adapted to different environments. The filter coefficients should, furthermore, be chosen to minimise the sensitivity to input pulse jitter. This can be achieved from theoretical arguments [5] or by using optimising procedures [6,7]. There are arguments that the same filter could be used for both purposes [8]. However, in FERMI it was decided to stay with two different filters, to keep maximum flexibility.

Pipeline memories stores data for a given time duration (2-3 μs) so that they can be retrieved in case of a first-level trigger accept is given. Sample sequences around accepted data, time frames, are stored in derandomizing buffers to be read out via the second level filter. This filter should more correctly be described as an inner product unit since it is only required to produce one value for each time frame recieved as input. It is equipped with different coefficient banks to be used in different situations (depending on flag settings). The filter is also triplicated using order statistic methods to chose which output to use. A programmable choice of the maximum, minimum or median value of the three FIR-results is given.

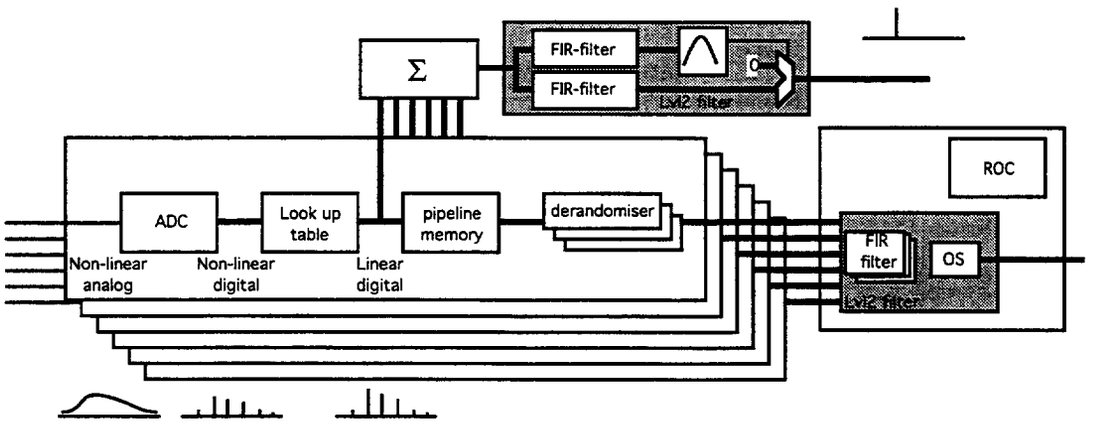

Fig. 1. FERMI overwiev

The level 1 filters

The level 1 filter is a parallel architecture of two independent FIR filters (fig. 2), where the energy filter is optimised for energy extraction, and the timing filter reshapes the pulses so that the pulse maximum can give precise time information. The latter filter operates with a somewhat reduced precision, using only the 16 most significant bits of

the 18 bits available to the energy filter. The peak finder enables data from the energy filter when a peak has been detected, provided that the energy is greater than a programmable threshold. Of 29 bits produced by the energy filter, a window of 11 bits is chosen as the standard output. The 11-bit window can slide over a range of 8 bits . If any bit above the window is set, an overflow error is reported. Raw data (29 bits) directly from the output of the energy filter is also available.

Thresholding the data will give the expression:

$$if \quad \{y_e(i) >= e_t\} \quad \Rightarrow y_e(i) \quad else \quad \Rightarrow 0 \tag{1}$$

and enabling this with a peak detection gives the corresponding energy value

$$if \quad \{y_t(i) < y_t(i-1) >= y_t(i-2)\} \quad \Rightarrow y_e(i-1) \quad else \quad \Rightarrow 0 \tag{2}$$

which is then appropriately windowed.

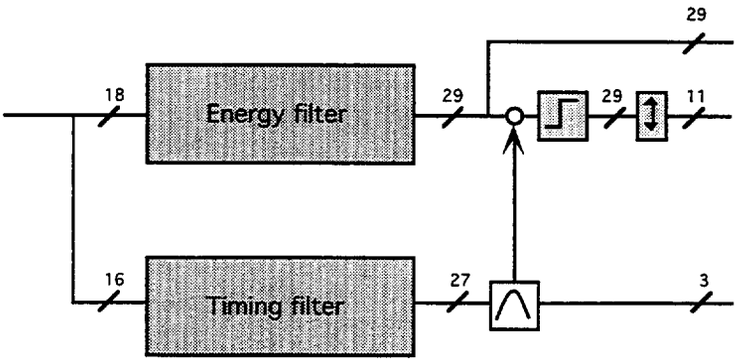

Fig 2. Energy and timing filters for the first level trigger output

Both filters have a semi-systolic architecture consisting of 6 taps (fig. 3).

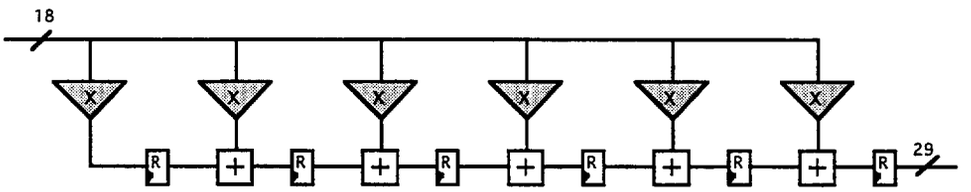

Fig 3. Architecture of the energy filter

A peak-detect flag from the peak finder and two pile-up flags accompany the ordinary data bits. The pile-up flags will be asserted to report a far-, or near overlap with the previous pulse if the internal counter (0-31 cycles programmable) detects two peaks within given time spans.

The main architecture of the level 1 filter is no different from the previous version. However, the implementation is. Some specific differences are worth pointing out. The width of the input data has changed from 14 to the 18 bits in the energy filter, and 16 bits in the timing filter. In both filters, the number of taps have increased from 5 to 6. However, in the timing filter, the last tap is set to a constant (hardwired to +0.5). This is to reduce processing time in the last tap, making it possible to perform a peak detection

within half a clock cycle (see appendix). As the coefficients still are 8 bits, the total number of bits will be 8 (from coefficients) + 18 (from input data) + 3 (from 6 taps) = 29 bits. No truncation is performed between the filtertaps to avoid roundoff errors.

The level 2 filters

When a time frame is requested, the level 2 filter produces an energy value from data stored in the derandomizer. The level 2 filter can use a variable time frame length (up to 15 samples) for the filtering. It uses full precision data, i.e. 18 bits. These samples are weighted with appropriate 10 bit coefficients (9 magnitude bits and a sign bit).

$$Y(i) = c_1 x(i) + c_2 x(i-1) + \ldots + c_N x(i-(N-1)) \tag{3}$$

where x is the time frame centred around the pulse, c the coefficients and the filter length N not greater than 16.

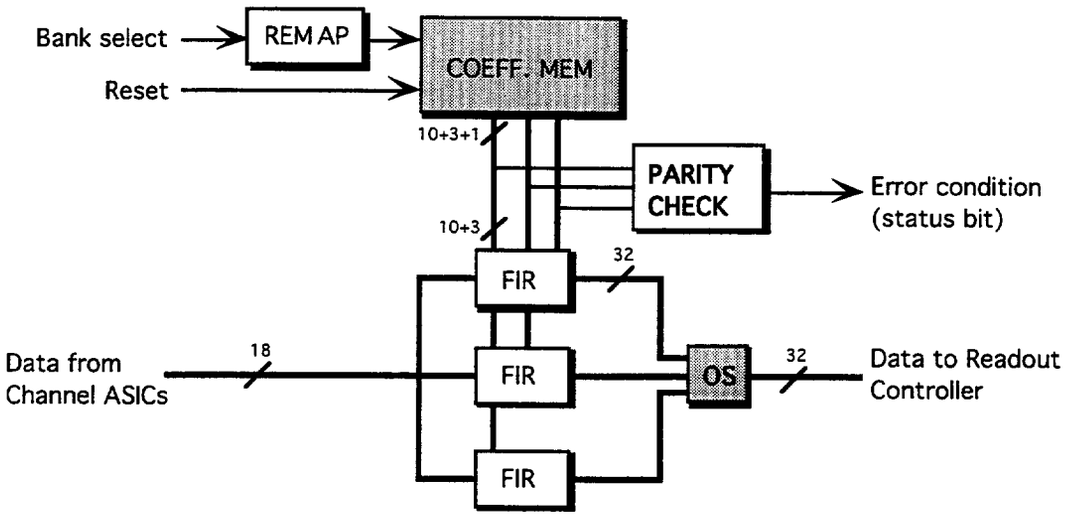

Fig. 4. Overview of the level 2 architecture

In case a pileup flag (far or near) accompanies the pulse, a coefficient set is selected so that the effect of the previous pulse can be diminished, or treated in some other suitable way. As the level 2 chip serves several channels, the possibility to switch coefficient sets have been introduced. This function provides a mean to correct for differences between channels.

Inside the level 2 filter 3 FIR filters are operating concurrently on the same set of input data. An Order Statistics (OS) operator, orders the three filter outputs according to value, in MAX, MID and MIN values, one of which is selected for output. A single filter output is 32 bits.

When disabling any of the three FIRs, its output will be set to the smallest possible value, thus not competing about being either of the MAX or MID value. If the smallest (MIN) value was wanted before disabling a FIR, then the smallest relevant value must be selected as the MID, as the disabled FIR automatically aspires to be the smallest.

Error protection in the LEVEL 1 and LEVEL 2 filters

The multipliers/adders are vital for the operation of the filters and therefore the main part to protect against radiation-induced errors. Preliminary layouts indicate that in the LEVEL 1, more than 75 % of the area is covered with components dealing with arithmetic operations. Arithmetic operations are combinatorial nets easy to protect from bit errors. Memories, on the other hand, are more sensitive, since data are stored for a long time. These are protected by a parity bit generated outside and checked inside the chip.

Modulo-3 code [9] versions of both filters have been implemented, producing a result with the same residue of modulo-3 as the real filters. A 3-bit representation is used for the modulo-3 code, to improve the fault tolerance. A zero is coded to "001", a one to "010" and a two to "100". The output from the real filters are then compared with the modulo-3 code filters and if there is a difference, a status register is notified. All adders and comparators have been duplicated, which means that no error correction is used, only error detection.

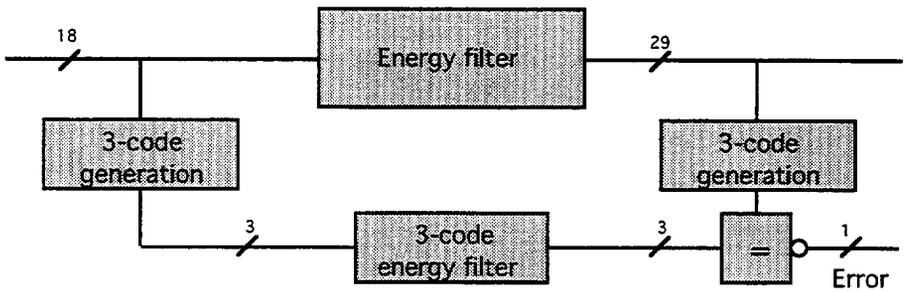

Fig. 5. A residue-3 version of the filter provides error detection for the nominal filter.

Filter implementation

The LEVEL 1 TAP

A "tap" in the level 1 and level 2 filters is not a generic multiplier and adder pair in the usual sense. The two components have been intimately integrated to save time during operation. The "multiplier" is a simple scheme that prepares the 7 rows of data, defined by the magnitude part of the coefficient, that then are added together in a tree of carry save adders (CSAs). The output from the previous tap is also included in the tree. A CSA has the ability to reduce processing time when adding many vectors together. When adding *three* vectors, the time consuming ripple of the carry can be eliminated by keeping the sum-, and carry output separate. Each bit in the two sums then originate from corresponding three bits of the operands. As a CSA sums 3 bitvectors to 2, we can only use such a tree to produce 2 vector outputs. However, this is not a big problem. In order not to loose time, we refrain from adding the last two vectors to one, but consider both bit-vectors as the "output" of the tap. This of course means that we have to include two additional rows to the CSA tree of the next tap. In the level 1 filter where 7 bit-coefficients (plus a sign bit) in the "multiplier part" produces a set of 7 row data which are combined with the two output-vectors from the previous tap. Worth noticing, is that, as long as you can use a "2 row output", the tap processing time will be independent of

the bitwidth of the input data, i.e. a 7 x 18 bit multiplication is as fast as a 7 x 512 bit one. In the end however, there must be a 2 => 1 adder to get a single vector as output. (See fast adder implementation)

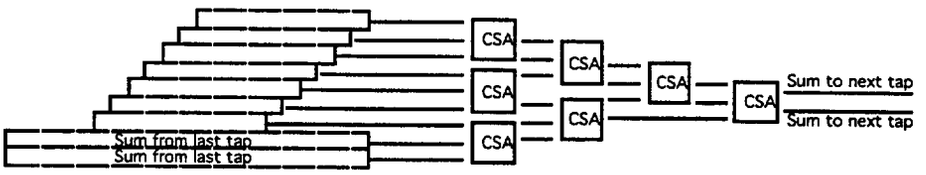

Fig. 6 The rows deduced from the multiplier part are summed in a CSA tree, producing a two vector output, which are together considered to be the result.

The level 2 TAP

In the level 2 filter, a slight modification of the level 1 multiplier was made to fit the recursive architecture of the filter. The output is fed back to the input of the adder, which have been separated from the multiplier part with a pipeline stage (fig. 7). The function of the filter is

$$y(i) = c_0 x(i) + c_1 x(i-1) + \ldots + c_{N-1} x(i-(N-1)) \tag{4}$$

where x is the input, and c the coefficient set. The filter allows up to N=16 samples. The output y, consist of two vectors. The multiplier and adder part of the tap takes one cycle each, and the output summation together with the OS-operation takes an additional 1.5 cycle (data is asserted on the rising edge). The fact that the width of the input data is 18 bits and the coefficients are 10 bits implies that each tap result will be 28 bits. Due to the 32 bit inner bit width there must not be more than 16 feedback loops (causing additional 4 bits) else the result will overflow. It is up to the ROC to make sure that this condition is always fulfilled. Furthermore, the tap is reset before every "filter-operation" to clear the feedback loop.

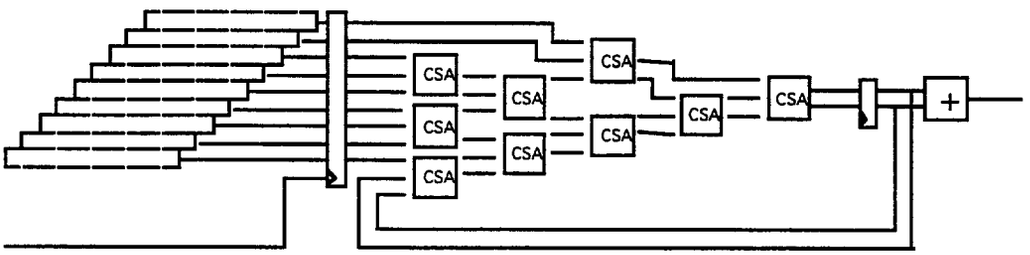

Fig. 7. The sum of the tap is fed back to the adder-part

Fast adder implementation

In some parts of the filters, addition is very time critical. This is especially true when adding the two output vectors in the last tap of each filter, where a CSA architecture cannot be used. It is also the case in the timing filter of the level 1 filter chip, where a comparison is performed in every clock cycle to check if a maximum has occurred. Figure 8 below shows how the last stage of the timing filter performs a multiplication with a coefficient set to +0.5 (1-bit right shift), a 2 => 1 addition, and a 32 bit comparison in just half a clock cycle. When performing the comparison, an adder is used (to subtract) to find out if A minus B is equal to or greater then zero, i.e. the minus sign bit is checked. A parallel addition is performed using a carry select structure. E.g. when implementing a

16 bit adder it is internally divided into four isolated 4 bit segments (fig. 9). Addition between the four groups of 2 x 4 bit vectors are performed in parallel. Since the value of the carry from the previous segment is not known, two additions are actually performed in each segment, one which assumes that the carry-in is zero, and one which assumes carry-in one. Finally, the correct sum is assembled according to the actual values of the carry bits calculated in the process.

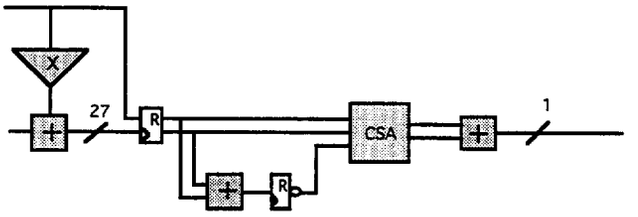

Fig. 8. Architecture of the last tap in the timing filter

The last combinatorial network is functionally a multiplexor tree where the carry bits are used to select the correct segment for the final word. Semi-analog simulations shows that a 32 bit adder is almost as fast as a 16 bit one (about 6 ns).

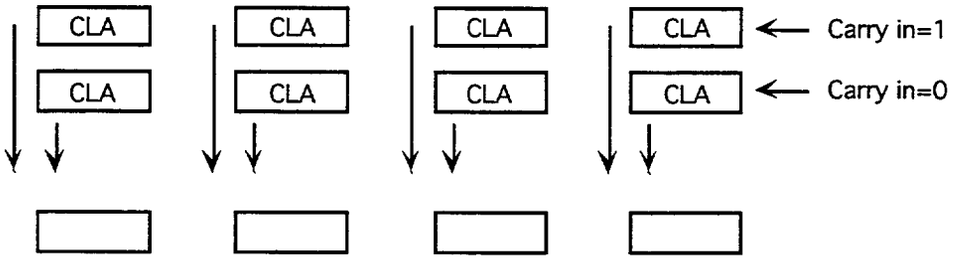

Fig. 9. For fast addition, (the figure shows a 16 bit addition), for each 4 bit segment, two versions are computed, each assuming the carry from the previous segment is either 1 or 0. Determining this, will easily select the right version.

VHDL code optimisation

All VHDL models have been synthesised and subsequently converted to a layout description, in a place and route tool, at an early stage to get a rough estimation of the final timing. From these simulations a component-wise back annotation have been used for later simulation on presynthesised VHDL level. As some parts of the filters are highly time critical, a VHDL approach must be used that will verify all such constraints. The choice of architecture of the multipliers and the adders, greatly depend on how they fulfil the 40MHz goal. The combination of setting the last timing tap to a constant and using fast addition made it possible to obtain a latency of the minimum 6+0.5 cycles without introducing extra pipeline steps in any part of the level 1 filters.

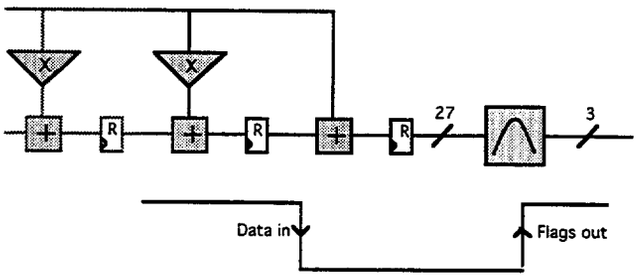

Fig. 10. The latency of the level 1 filter is 6+0.5 clock cycles, as the peak is detected using the data in stream immediately on the falling edge. See fig. 8 for details..

Conclusion

The specification and design of the level 1 and the level 2 filters in FERMI have been described and discussed. A minimum latency solution was achieved for level 1. Simulations show that it will fulfil the 40 MHz speed requirements with a safety margin. Previous work on the level 1 filter has investigated high speed architectures, especially the filter taps.

The two chips will be synthesized and layouted by CAEN microelettronica.

Appendix Optimised filter latency

Assume that the timing and energy filters can be expressed as FIR filters with 6 taps

$$p(n) = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3} + a_4 x_{n-4} + a_5 x_{n-5} \tag{5}$$

$$q(n) = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} + b_3 x_{n-3} + b_4 x_{n-4} + b_5 x_{n-5} \tag{6}$$

The BCID condition can then be stated as follows:

$$if \quad \{q(n) > th \quad \& \quad p(n+1) < p(n) \geq p(n-1)\} \quad \Rightarrow q(n), \quad else \quad \Rightarrow 0 \tag{7}$$

The conditional expression in (7), can be expanded:

$$th < q(n) \quad \& \quad \neg p(n) < p(n-1) \quad \& \quad p(n+1) < p(n) \tag{8}$$

The inequality:

$$p(n+1) < p(n) \tag{9}$$

can be combined with (1):

$$x_{n+1} < \frac{a_0 - a_1}{a_0} x_n + \frac{a_1 - a_2}{a_0} x_{n-1} + \cdots + \frac{a_4 - a_5}{a_0} x_{n-4} + \frac{a_5}{a_0} x_{n-5} \tag{10}$$

or

$$x_{n+1} < r(n) = c_0 x_n + c_1 x_{n-1} + \cdots + c_4 x_{n-4} + c_5 x_{n-5} \tag{11}$$

We can now rewrite (8) once more as:

$$th < q(n) \quad \& \quad \neg x_n < r(n-1) \quad \& \quad x_{n+1} < r(n) \tag{12}$$

Let us further assume that a multiply and add operation can be executed in one clock cycle. The value r(n-1) can then be made available at time t=n. At t=n+1 we will have

the results of $x_n < r(n-1)$ and also x_{n-1}, q(n) and r(n), i.e. all the components required to determine (12).

If the comparisons $th < q(n)$ and $x_{n+1} < r(n)$, the 3 input and-operation between their Boolean results and the output control of q(n) can be accomplished in half a clock cycle we will achieve minimum latency, which is halve a clock cycle after all required data are available. In this we have assumed the convention that input data are latched on a positive flank of the system clock while the output data are asserted on a negative flank.

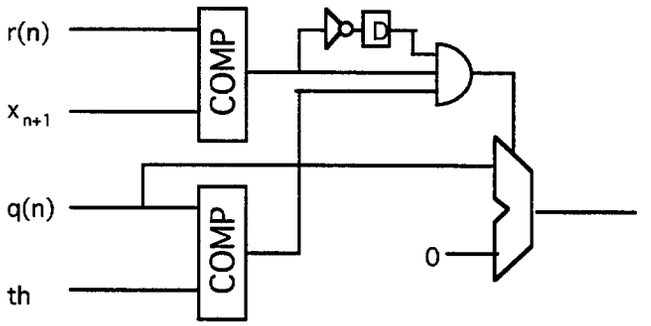

Fig 11. The output stage in an optimized filter structure.

References

1. P. Bailly et.al., "VHDL implementation of high speed algorithms for the level 1 trigger filter in FERMI", FERMI Note #23, Aug 1993
2. A. Dell'Acqua et al., "A Digital Readout Microsystem for Calorimetry at LHC", NIM A 344, vol. 1, pl 80- 184, 1994.
3. H. Alexanian et al, "FERMI : a digital Front End and Readout MIcrosystem for high resolution calorimetry." NIM A 357 (1995) (306-317)
4. H. Alexanian et al., "FERMI : Optimized digital feature extraction in the FERMI microsystem" NIM A 357 (1995) (318-328)
5. W.E. Cleland et al., "Signal processing considerations for liquid ionization calorimeters in a high rate environment", NIM A 338 (1994) (467-497)
6. S.J. Inkinen,"A Trainable FIR - OS Hybrid Filter for Precision Pulse Height Measurement of Particle Detector Signals", FERMI Note #19, July 1993
7. S.J. Inkinen et al.,"Trainable Hybrid Filter structures in Particle Detector Readout Systems", FERMI Note # 29, Feb 1994
8. J.Garvey et al.,"Bunch Crossing Identification for the ATLAS level-1 Calorimeter Trigger", ATLAS internal note DAQ-NO-51, May 1996
9. Lucas pek om felkoder i aritmetik