SLAC–PUB–5303
July 1990
(I)

FAST TRACKING IN A TRIGGER SYSTEM BASED ON A MEMORY LOOKUP UNIT*

I. Navon[(a)] **and A. Navon**
Stanford Linear Accelerator Center,
Stanford University, Stanford, CA 94039, USA
and Tel-Aviv University, Ramat-Aviv, Israel

E. Piasetzky, Y. Mardor, and J. Alster
Raymond and Beverly Sackler Faculty of Exact Sciences,
School of Physics, Tel-Aviv University, Ramat-Aviv, Israel

S. E. Rock
The American University, Washington, DC 20016, USA

Abstract

One section of a fast tracking trigger system for a proposed large particle spectrometer is presented. This subsystem is based on a novel use of a Memory Lookup Unit. The electronic circuit and the associated software table are described in detail. The subsystem can accomplish the tracking for a complex event in a few microseconds. A prototype subsystem was assembled and tested successfully.

Submitted to *Nuclear Instruments & Methods A.*

⋆ Work supported by Department of Energy contract DE–AC03–76SF00515, by the US–Israel Binational Science Foundation, and by the Israeli Academy of Sciences.

(a) Mailing address: Physics Department, Tel-Aviv University, Ramat-Aviv, Israel 69978

1 Introduction

In this paper we describe a novel use of memory lookup units (MLU) in a trigger tracking system. This system was developed as part of the proposed PEGASYS[1] detector at SLAC. We choose to describe only a simple subsystem which performs linear tracking because it presents the basic ideas, yet is simple enough to be fully portrayed here. The same principles were used successfully to design more complex subsystems that perform curved tracking in a magnetic field.

1.1 General design considerations

The triggering system for PEGASYS has to cope with an initial expected hit rate of $50kHz$, remove the backgrounds and select the few events (no more than 200 per second) that will be read and written to tape for off-line analysis. The rejection of this high background rate required the inclusion of fast tracking in the trigger electronics. A multi-level trigger system was developed, with a sophisticated "high level" trigger processor to perform the tracking and to calculate the kinematical parameters. The expected rate for starting this level is a few kHz thus, to keep the deadtime reasonable, it should be able to decide whether a specific event is "good" in less than $20\mu s$. A calculation of tracking through several detector layers, involving energy readout from a few dozens of fast ADCs, is needed to reach this decision. Clearly, a microprocesor - based system would hardly have time to read in the data, and certainly would not be able to complete the tracking calculations within the required time limit. This was the principal reason that led to the development of this tracking system based on memory lookup units.

Memory lookup units were suggested previously as part of fast tracking logic [2,3], but normally it was limited to single step calculations. Looping, if any, was done externally under hardware control. Here we "looped" the MLU on itself so that it would perform the equivalent of a FORTRAN "DO loop", with self counting of the number of iterations. The MLU itself controls the looping (in this case two nested loops), and stops it under complex "software" exit conditions. The introduction of a new large MLU unit with 128 Kbyte memory (16 bit input address and 16 bit output data), which

was developed at the Cyclotron Laboratory of Michigan State University [4], made the implementation of this idea feasible.

The trigger system was built entirely from available CAMAC ECL modules and a few supporting NIM units. The heart of the system is that new larger MLU, while the rest of the modules are mostly standard CAMAC "ECLine" from LeCroy.

2 The Vertical Tracking subsystem

We choose to describe here the vertical tracking subsystem. The PEGASYS magnet spectrometer is designed to have horizontal deflection only, so the vertical tracking is done along straight lines (except for small second order corrections that can be neglected for the triggering system). Also, the target size is sufficiently small so that, for the accuracy needed in the triggering calculations, it can be considered to be a point source. These two facts make the vertical tracking exceptionally simple. In figure-1, we show the basic geometry for the vertical tracking, where all the detectors that are not relevant to the tracking are removed. There are two drift chambers, called top and bottom, with 40 horizontal wires each. There are also four quadrants of horizontal shower counter bars with 16 bars in each quadrant.

Initially, the shower counter data (after some pre-processing which is not relevant here) are stored in a Data Stack (LeCroy 2375). The stack can contain up to 256 words of 16 bit. Each data word corresponds to a hit in the shower counter, and its structure is as follows:

16	15	14	13	12 – 9	8 – 1
	H/V	T/B	L/R	sub-address	ADC (energy)

Bits 1-8 contain the ADC value (energy) of a shower counter hit . Bits 9-12 contain the subaddress, i.e. the number (0–15) of the bar that fired. The quadrant is identified as left/right and top/bottom by bits 13-14. Bit 15 identifies it as vertical information (number of the horizontal bar), so that in our example it will always be set. Bit 16 is not used. The number of words in the stack equals the number of bars that were hit in the event. For a typical event this number is 2-7.

The hit pattern of the Drift Chambers was transferred initially to a Data Array (LeCroy 2376A). In this array of 1024 bits, each wire is represented by

a single bit which is on when the corresponding wire fired. The Data Array can answer queries whether a specific address, or range of addresses, is on by switching a status bit on or off. This answer comes in response to a search address and search width (if necessary) which are present at the search input, in coincidence with a strobe pulse at the search strobe input.

The purpose of the vertical tracking subsystem is to determine whether a specific hit in the shower counter (represented as a stack word) has a matching drift chamber hit in the Data Array. If such a match is found, the vertical position in the input word (shower counter bar number) will be replaced by the drift chamber vertical position (wire number), and the result (energy and vertical position) will be written to the output stack. The hits in the shower counter without a matching hit in the drift chamber will be discarded. The result will have a better position resolution because the resolution derived from the drift chamber wire number is better than the one derived from the shower counter bar number. It is important to point out that for each shower counter bar we have to search several DC wires, but it cannot be accomplished in a single query of width larger than one. The reason is, that in order to improve the resolution, we not only wish to know whether a wire fired in a certain range, but also the specific wire that fired.

3 Circuit description

The circuit shown in fig. 2 accomplishes the task described in the previous section by performing a search loop for each word in the stack. For each stack word the MLU issues five search addresses to the Data Array in five consecutive cycles. The search starts at the "central address", which is the DC wire number closest to the position obtained by linear interpolation between the center of the SC bar and the target. Next, the search proceeds up and down, first by one wire and then by two, for a total of five wire checks. If a hit was found in the searched range, the updated information is written to the output stack and the loop is terminated. Thus, if there was more than one hit in the search range, the one closest to the "central address" is automaticlly chosen because it was tested first.

The circuit includes four CAMAC modules: two data stacks (LRS 2375), a data array (LRS 2376A) and a MLU [4]. Only a few coincidence units

(2/16 of LRS 4516) are used in the timing circuit. The delays are easily implemented by selecting the proper cable lengths. Two external signals are used: a $40ns$ start signal which starts the activity of the circuit and a $10MHz$ external clock with a width of $15ns$. A "fast clear" signal is generated at output pin 15 of the MLU in case that no valid event was found.

3.1 Timing considerations

The MLU operates in latched mode (internal control register=12). In this mode the input data are latched by a "Hold" signal, and are held there until a "Clear" signal will be applied. The timing of the circuit is illustrated in figure 3. A valid "Hold" signal has to be preceded by a "Clear" signal to the previous "Hold" by at least $18ns$. This is the reason for adding a $20ns$ delay to the Clock signal going to the "Hold". The output and the output "Ready" signal appear $40ns$ after the leading edge of the "Hold" signal. Since some of the MLU outputs are connected back to the input, the timing situation is somewhat like that of a cat trying to catch its own tail. To latch a new input we first have to clear the unit, but this will clear the outputs that we wish to latch at the input! Luckily, the output signal persists for $35ns$ after the "Clear", which is sufficient to latch the new input. As can be seen in the timing diagram the input lines must be stable for $15ns$ starting $5ns$ prior to the "Hold". This allows the previous output to be latched just before the "Clear" will erase it.

It is important to note that the internal delay of the DA, from search strobe to a valid output (Stat), is $60ns$. This means that when a specific MLU input generates a search query to the DA, the DA answer (Stat) may change state about $105ns$ after that MLU input was latched ($40ns$ for the MLU, $60ns$ for the DA and $5ns$ for wiring). This would happen right in the middle of the "Hold" of the next cycle when all the input lines should be stable. To avoid this ambiguity, a $20ns$ delay was added to the search strobe of the DA. In practice, this means that the answer to a specific input will not be present at the next input cycle, but only in the following one. This hardware delay has to be taken into account in the software that builds the lookup table. In contrast, we see that the response to "next SC" signal (bit 8 - MLU output) does arrive at the next input cycle, because the Data Stack needs only $30ns$ from "Read Enable" to "Read Ready". However, for the

sake of compatibility with other more complex subsystems, we delayed the use of the new SC subaddress to the following cycle.

The start is given by an external signal. Unless the start is synchronized with the clock the length of the first cycle will not be constant. To avoid the complication, the software was designed to be insensitive to this variation.

4 Circuit operation

The external "start" signal clears the output stack with a "Master Reset" and starts the cycles by giving the first "Hold" to the MLU. The subsequent cycles are generated by the MLU output "Ready" signal in coincidence with the external clock, until the process is stopped by the "done" signal (output bit 16). For every stack word the MLU uses the SC subaddress (bar number) in bits 9-12 to "calculate" the expected DC wire number. The resulting "predicted DC address" at the MLU output bits 9-14 is presented to the DA as a search address. The search strobe to the DA comes from output bit 5 with a delay of $20ns$. The DA's answer to this query (Stat) is received at the MLU input bit 7.

Bits 1-3 of the MLU serve as a loop counter. If there is no positive answer from the DA, the loop continues to query the neighboring addresses. A positive response from the DA causes the search address that initiated this response to reappear at the MLU output together with a Write Enable to the output stack. The address, together with the ADC value and the two bits that identify the quadrant (left/right and top/bottom) are written into the output stack. Concurrently with the Write Enable we set bit 6 ("at least one"), which is then latched by being fed back to the input. The software is designed to keep this bit "on" at the output whenever it is "on" at the input. After processing all the input stack words this bit will indicate whether there was at least one good event. If a DC hit was found (a positive answer), or if all five search attempts failed, a "next SC" signal is issued (output bit 8) to strobe the Read Enable of the input stack and to get the next SC. A "next SC" will not be issued in case that the Read Overflow (ROF) from the stack is "on", signaling that the last data word in the stack was read. In that case a "done" signal is turned on at output bit 16, which blocks the next "Hold", and thus stops the processing until the next start. If the bit "at least one

good" is not on at the end of the process, the "fast clear" (output bit 15) will be set.

5 The Lookup Table and related software

The lookup table (a 64K array of 16 bit words), which controls the operation of the whole circuit, is created by external software. Each MLU in the trigger system requires its own table-creating program. The programs were written in FORTRAN on a VAX computer. The program for the MLU in fig. 2 is listed in the appendix. A central point in the design is that the whole complexity of the required calculations is handled by these external programs, i.e. the "intelligence" resides in the resulting lookup tables. Therefore, the iteration cycle time, even for very complex calculation, can be $100ns$ or less. A common loading program is used to load the prepared tables into the MLU in the appropriate CAMAC slot.

The logic of the program is based on "cycle tables" which describe the desired operation at each hardware cycle. Table 1 presents two examples of such a table, which will be discussed in details below. It is important to note that in these tables each cycle corresponds to a specific time slice. The propagation time through the MLU takes up one cycle, so that the input at cycle 1 generates the suitable output at cycle 2, and so forth. On the other hand, the output lines that are looped back (1-3,6,7) take only a nanosecond to appear at the input. Thus, the outputs of the "Counter", "Write Enable" and "At Least One" appear during the current cycle at the input. This is contrary to intuition because the output of each cycle generates the input of that cycle and not vice versa!

Table 1 shows the status of selected inputs and outputs during consecutive cycles. We begin with example 1 of table 1 which describes the case where no matching hit is found in the DA. At the beginning, all the lines that are looped back are at logical zero because the unit is stopped after a "Clear" so that all the outputs are at zero. The operation starts at cycle 0 with an external start pulse which latches the specific SC subaddress (SUB) that is present at input lines 9-12. The counter (CNTR) at lines 1-3 is looped back and thus is at "0" indicating the number of iterations that were done. Similarly, the WE at input line 5 and the latch "at least one" at input line

6, are also "0". This input combination generates the output at cycle 1 and, in particular, the first search address (1 SRCH) that is then presented to the DA. Simultaneously we also set the DA search strobe (DA) at output line 5 to "1", so that the search is activated. The counter (CNTR) is set to 1 indicating the cycle number, and the write enable (WE) that strobes the output stack is still zero.

The input at cycle 1 gets the cycle number from the output. Together with the subaddress (SUB), that is now static at the input, they generate the second search address at the output of cycle 2. In the meantime, the DA's answer to the first search request arrived at input line 7 (we denote it: Y/N). What happens from now on depends on the value of the DA status bit (Y/N). In our example the DA answer is negative (Y/N=N) and we continue the search as before. The third search address (3 SRCH) will appear at the output of cycle 3, etc. The maximum number of queries (search addresses) to the DA is five, as explained in section 3. The last answer from the DA arrives at the input of cycle 6 which, in turn, generates the output at cycle 7 and sets NEXT=1. This causes the next subaddress (next_SUB) to be fetched from the input stack. After cycle seven the input counter is reset and the next search loop will start. Note that in our example, where no matching hit was found, the loop took up eight cycles.

Example 2 of table 1 describes a search loop which ends with a positive result in the second try. Cycles 0,1 and 2 are as in example 1. The positive answer arrives at the input at cycle 3 (Y/N=Y), indicating that the search address at cycle 2 (2 SRCH) found a matching hit in the DA. This address is regenerated at the output of cycle 4, together with a strobe signal to the output stack (WE). The signal (WE) strobes the hit address into the output stack together with the energy and the quadrant bits. In the same cycle we set NEXT=1, which causes the next subaddress (next_SUB) to be fetched from the input stack. The WE signal is also fed back to the input of cycle 4. There it instructs the MLU to clear the cycle counter (CNTR) for the next cycle so it will be ready to start a new search loop with a new subaddress at the input. The procedure ensures that the search loop will be truncated after the first match. In this example, where the match was found in the second try, the loop took up five cycles. In general, the positive answer from the DA may appear at the input in any one of the cycles 2-6. The loop will then take 4-8 cycles, respectively.

In summary the search consists of two nested "DO loops". The inner one is generated by the MLU and loops through the search addresses. The outer one loops through the SC bars (input stack words) and is controlled by the hand shake between the input stack and the MLU. Bit 6 acts as a latch. It is set if there is a positive answer from the DA for any of the search queries. The bit is returned to input line 6, and the software leaves this output on when the input is on. In this way we know whether the whole event in the input stack has at least one good track. Simultaneously with reading the last word in the event, the input stack turns on the read overflow (ROF) signal. This signal is connected to input line 8 of the MLU and, if it is on at the end of the search loop, the MLU issues a "done" signal at output 16. Then this signal stops the MLU activity by blocking the next strobe to the unit. If there wasn't "at least one good track", a "fast clear" pulse is issued together with the "done" signal. The MLU is stopped after a "Clear" but prior to the latch ("Hold"). Thus, $35ns$ after the "Clear" all the outputs will be reset to "0", and the unit will be ready to start processing the next event.

6 Conclusions

The concept of looping back MLU output bits, and using them as a loop counter, proved to be easy in design and implementation. In this way the MLU can make several iterations over a specific input, thus enabling more sophisticated processing. We used it extensively in other sections of the trigger system (e.g. to check the tracking through several detectors layers for a given single hit). In a sense it amounts to building a special purpose computer without a CPU, that can complete a loop iteration in less than a $100ns$. Such a scheme should find wide use in fast intelligent trigger systems.

Acknowledgments

It is a pleasant duty to thank F.Dietrich and M. Perl, whose support and encouragement made this work possible. We thank C.Hyde-Wright for loaning us the computer. The hospitality of SLAC and of group E, in particular, created a pleasant working environment. All the collaborators of the PEGASYS project have acted to stimulate this work. Special thanks are due

to I.Mardor who helped us with the simulation programs, and to Z.Szalata for advice on computer problems.

References

[1] PEGASYS proposal, Stanford Linear Accelerator Center, 1989.

[2] A. Fucci, Sr. Amendolia, E. Bertolucci, U. Bottigli, C. Bradaschia, L. Foa, A. Giazotto, M. Giorgi, M. Givoletti, P. Lucardesi, A. Menzione, D. Passuello, M. Quaglia, L. Ristori, L. Rolandi, P. Salvadori, A. Scribano, R. Stanga, A. Stefanini, M.L. Vincelli, Nucl. Instr. and Meth. **147** (1977) 587.

[3] L.B. Levit, M.L. Vincelli, Nucl. Instr. and Meth. **A235** (1985) 396.

[4] The Fast Decision Module, M.R. Maier, M. Robertson, A. vanderMolen and G.D. Westfall, to be published in Nucl. Instr. and Meth.

Figure captions:

Fig. 1 Schematic geometry for vertical tracking. The spectrometer magnet and all the detectors that are not relevant were omitted.

Fig. 2 Circuit diagram of the electronics. The input Stack, output Stack, Data Array and MLU are all CAMAC modules.

Fig. 3 Timing diagram for the circuit in fig.2. All the signals are drawn in positive logic.

TABLE 1. MLU logic for the search loop cycles

(example 1: no matching hit in DA)

cycle	0	1	2	3	4	5	6	7	0
In	ROF=0 SUB CNTR=0 WE=0	SUB CNTR=1 WE=0	SUB CNTR=2 WE=0 Y/N=N	SUB CNTR=3 WE=0 Y/N=N	SUB CNTR=4 WE=0 Y/N=N	SUB CNTR=5 WE=0 Y/N=N	SUB CNTR=6 WE=0 Y/N=N	SUB CNTR=7 WE=0	ROF=0 next_SUB CNTR=0 WE=0
Out	0000 CNTR=0 DA=0 WE=0	1 SRCH CNTR=1 DA=1 WE=0	2 SRCH CNTR=2 DA=1 WE=0	3 SRCH CNTR=3 DA=1 WE=0	4 SRCH CNTR=4 DA=1 WE=0	5 SRCH CNTR=5 DA=1 WE=0	0000 CNTR=6 DA=0 WE=0	0000 CNTR=7 DA=0 WE=0 NEXT=1	0000 CNTR=0 DA=0 WE=0

(example 2: hit found in second try)

cycle	0	1	2	3	4	0
In	ROF=0 SUB CNTR=0 WE=0	SUB CNTR=1 WE=0	SUB CNTR=2 WE=0 Y/N=N	SUB CNTR=3 WE=0 Y/N=Y	SUB CNTR=4 WE=1	ROF=0 next_SUB CNTR=0 WE=0
Out	0000 CNTR=0 DA=0 WE=0	1 SRCH CNTR=1 DA=1 WE=0	2 SRCH CNTR=2 DA=1 WE=0	3 SRCH CNTR=3 DA=1 WE=0	2 SRCH CNTR=4 DA=0 WE=1 NEXT=1	0000 CNTR=0 DA=0 WE=0

APPENDIX

```
      program VERT1
c     -------------
c
c     Vertical Tracking.
c     The program calculates the lookup tables that would be loaded into
c     the MLU, so the electonics would be able to find good vertical tracks.
c     "Good track" means that for a SC address there is at least one DC wire
c     that fired out of the five wires that are nearest to the linear track
c     to that SC.
c
c Input Stack : (bits 1-16)
c ------------
c             1-8     ADC energy (PR+3TA)
c             9-12    Sub Address (of SC)
c             13      Left / Right
c             14      Top / Bottom
c             15      hor./ver.
c
c Output Stack : (bits 1-16)
c -------------
c             1-8     Energy Ey (from horizontal SC bars)
c             9-14    Y (from search address)
c             15      Left / Right
c             16      Top / Bottom
c
c MLU Input : (bits 1-16)
c ----------
c             1-3     Loop counter (from MLU Output)
c             4       -
c             5       WE: Write to output stack (from MLU Output)
c             6       At least 1 Good vertical track (from MLU Output)
c             7       DA Yes/No       (from DA)
c             8       ROF (from Input Stack)
c             9-12    Sub Address (from Input Stack)
c             13-16   -
c
c MLU Output : (bits 1-16)
c -----------
c             1-3     Loop Counter
c             4       -
c             5       DA strobe
c             6       At least 1 Good vertical track
c             7       WE: Write to output Stack
c             8       Next SC (Goes to Input Stack)
c             9-14    Search Address
c             15      Fast Clear - Not even 1 Good Track
c             16      Done
c
c-----------------------------------------------------------------------
```

```
c
c
        implicit none
c
        integer*2       MLU(0:65535)    ! 64K memory
c
        integer*4       MLU_Output, I
        integer*2       MLU_Output_Short(2)
        equivalence     (MLU_Output, MLU_Output_Short(1))
c
        integer*4       IN$Loop_Counter,        OUT$Loop_Counter,
        1               IN$Wrt_enable,          OUT$Wrt_Enable,
        2               IN$At_Least_1,          OUT$At_Least_1,
        3               IN$YN_DA,       IN$ROF,         IN$Sub_Adrs,
        4               OUT$DA_strobe,  OUT$Next_SC,    OUT$Search_Adrs,
        5               OUT$Fast_Clear, OUT$Done

        integer*4       SEARCH_ADRS     ! functions

        integer*4       B(16)           ! Bit value = 2**(i-1)
        integer*4       valid_YN_DA     ! when IN$YN_DA value is valid

        integer*4       search_offset (0:4)     ! how much to offset
        common /lp/     search_offset           ! each search adrs
        data            search_offset   /0, 1, -1, 2, -2/
c
c-----------------------------------------------------------------------
c
        do I =1, 16
                B(I) = 2**(I-1)
        end do
c
c------------------- open output file ----------------------------------
c
        open (1, name = 'MLU_V.TBL', status = 'NEW', form = 'UNFORMATTED')
c
c------------------ fill the table -------------------------------------
c
        do I = 0, 65535                 ! go over all Input addresses

                                        ! get relevant bits from input

                IN$Loop_Counter = I .and. '7'X          ! bits 1-3
                IN$Wrt_Enable   = I .and. '10'X         ! bit 5
                IN$At_Least_1   = I .and. '20'X         ! bit 6
                IN$YN_DA        = I .and. '40'X         ! bit 7
                IN$ROF          = I .and. '80'X         ! bit 8
                IN$Sub_Adrs     = I .and. 'F00'X        ! bits 9-12


c----- valid_YN_DA ----------
                valid_YN_DA = 0
                if      ((IN$YN_DA .ne. 0)
        1           .and. (IN$Loop_counter .ge. 2)
        2           .and. (IN$Loop_counter .le. 6)
        3           .and. (IN$Wrt_Enable .eq. 0))
        4       valid_YN_DA = 1


                                ! calculate output bits

c----- OUT$Loop_Counter -----
                OUT$Loop_Counter = IN$Loop_Counter + 1
                if (IN$Loop_Counter .eq. 7)     OUT$Loop_counter = 0
                if (IN$Wrt_Enable .ne. 0)       OUT$Loop_counter = 0

c----- OUT$DA_strobe -----
                OUT$DA_strobe = 0
                if ((IN$Loop_counter .eq. 0) .or. (IN$Loop_counter .eq. 1))
        1                                       OUT$DA_strobe = 1
                if ((IN$Loop_counter .le. 4) .and. (IN$Loop_counter .ge. 2)
        1           .and. (valid_YN_DA .eq. 0))  OUT$DA_strobe = 1

c----- OUT$At_Least_1 -----
                OUT$At_least_1 = 0
                if ((IN$At_least_1 .ne. 0) .or. (valid_YN_DA .ne. 0))
        1          OUT$At_least_1 = 1

c----- OUT$Wrt_Enable -----
                OUT$Wrt_Enable = 0
```

```
                if (valid_YN_DA .ne. Ø)  OUT$Wrt_Enable = 1

c----- OUT$Next_SC -----
                OUT$Next_SC = Ø
                if (valid_YN_DA .ne. Ø)       OUT$Next_SC = 1
                if ((IN$Loop_Counter .eq. 6) .and. (IN$Wrt_Enable .eq. Ø))
        1                                     OUT$Next_SC = 1

c----- OUT$Search_Adrs -----
                OUT$Search_adrs = SEARCH_ADRS (IN$Sub_Adrs,
        1                                      IN$Loop_Counter,
        2                                      valid_YN_DA)

c----- OUT$Done -----
                OUT$Done = Ø
                if ((IN$ROF .ne. Ø) .and. (OUT$NEXT_SC .eq. 1))  OUT$Done = 1

c----- OUT$Fast_Clear -----
                OUT$FAst_clear = Ø
                if ((OUT$Done .eq. 1) .and. (IN$At_least_1 .eq. Ø))
        1                        OUT$FAst_clear = 1

c
c-------- fill MLU value for I --------------
c
                MLU_Output  =     OUT$Loop_counter      ! bits 1-3
        1               + B(5)  * OUT$DA_strobe         ! bit 5
        2               + B(6)  * OUT$At_least_1        ! bit 6
        3               + B(7)  * OUT$Wrt_Enable        ! bit 7
        4               + B(8)  * OUT$Next_SC           ! bit 8
        5               + B(9)  * OUT$Search_adrs       ! bits 9-14
        6               + B(15) * OUT$fast_Clear        ! bit 15
        7               + B(16) * OUT$Done              ! bit 16

                MLU (I) = MLU_Output_Short (1)  ! equiv. to lower 16 bits
                                                ! of MLU_Output, for overflow
                                                ! reasons
        end do
c
c---------------- write table to output file ---------------------------
c
        write (1) MLU
c
c               do i=Ø,15
c                 type *,search_adrs (i*256,Ø,Ø)
c               enddo

        end
c
c
c
c-----------------------------------------------------------------------
```

```
c
        integer*4 function SEARCH_ADRS (sub_adrs, loop_counter, YN_DA)
c       ------------------------------------------------------------
c
c calculates search address from sub address
c Input :
c--------
c loop_counter - bits 1-3  (value Ø-4)
c YN_DA        - bit  7    (Yes/No for search adrs in DA, Ø-No)
c sub_adrs -     bits 9-12 - SC #  (value Ø-15)
c
c Output :
c---------
c search_adrs - bits 1-6 - DC #  (value Ø-33 (no. of wires in each chamber))
c-----------------------------------------------------------------------------
c
        implicit none
        include 'PEGASYS_GEOM.PRM'              ! Geometry def. file

        integer*4       sub_adrs, addr, loop_counter, SC_num
        integer*4       YN_DA, central_addr
c
        real*4          Y_SC, Y_DC
        integer*4       search_offset (Ø:4)     ! how much to offset
        common /lp/     search_offset           ! each search adrs
c
c
c-----------------------------------------------------------------------------
c
        SC_num = ISHFT (sub_adrs, -8) .and. 'F'X    ! 4 bits for adrs (Ø-15)

        Y_SC = PR_GAP_Y + (SC_num +Ø.5) * PR_cell_Y
        Y_DC = Y_SC * ZDC / ZPR
        central_addr = INT ((Y_DC - DC_gap_Y) / DC_cell_Y)   !wire # from Ø..

        if (central_addr .lt. Ø)  central_addr  = Ø
        if (central_addr .ge. DC_wires_Y)  central_addr  = DC_wires_Y-1
c
c-------- add offset according to loop counter and DA status --------------
c

        if ((YN_DA .eq. Ø) .and.                ! normal search case
        1   (Loop_Counter .le. 4) .and. (Loop_Counter .ge. Ø)) then

           addr = central_addr - search_offset (loop_counter)  ! check 2 above
                                                               ! & 2 below

        else if ((YN_DA .ne. Ø) .and.           ! "found hit" case
        1        (Loop_Counter .le. 8) .and. (Loop_Counter .ge. 2)) then

           addr = central_addr - search_offset (loop_counter-2)! go 1 srch addr
        else                                                   ! backwards

           addr = Ø                             ! no need for search adrs
        end if


        if (addr .ge. DC_wires_Y)  addr = DC_wires_Y-1  ! unless reached
        if (addr .lt. Ø)   addr = Ø                    ! limits (value Ø-33)

        SEARCH_ADRS = addr

        return
        end
```

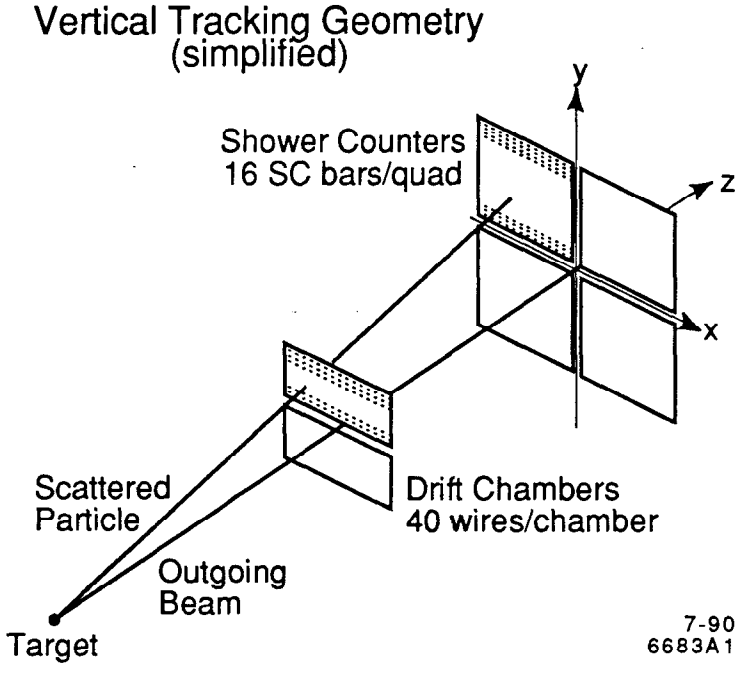
Vertical Tracking Geometry
(simplified)
y
Shower Counters
16 SC bars/quad
z
x
Scattered
Particle
Drift Chambers
40 wires/chamber
Outgoing
Beam
Target
7-90
6683A1

Fig. 1

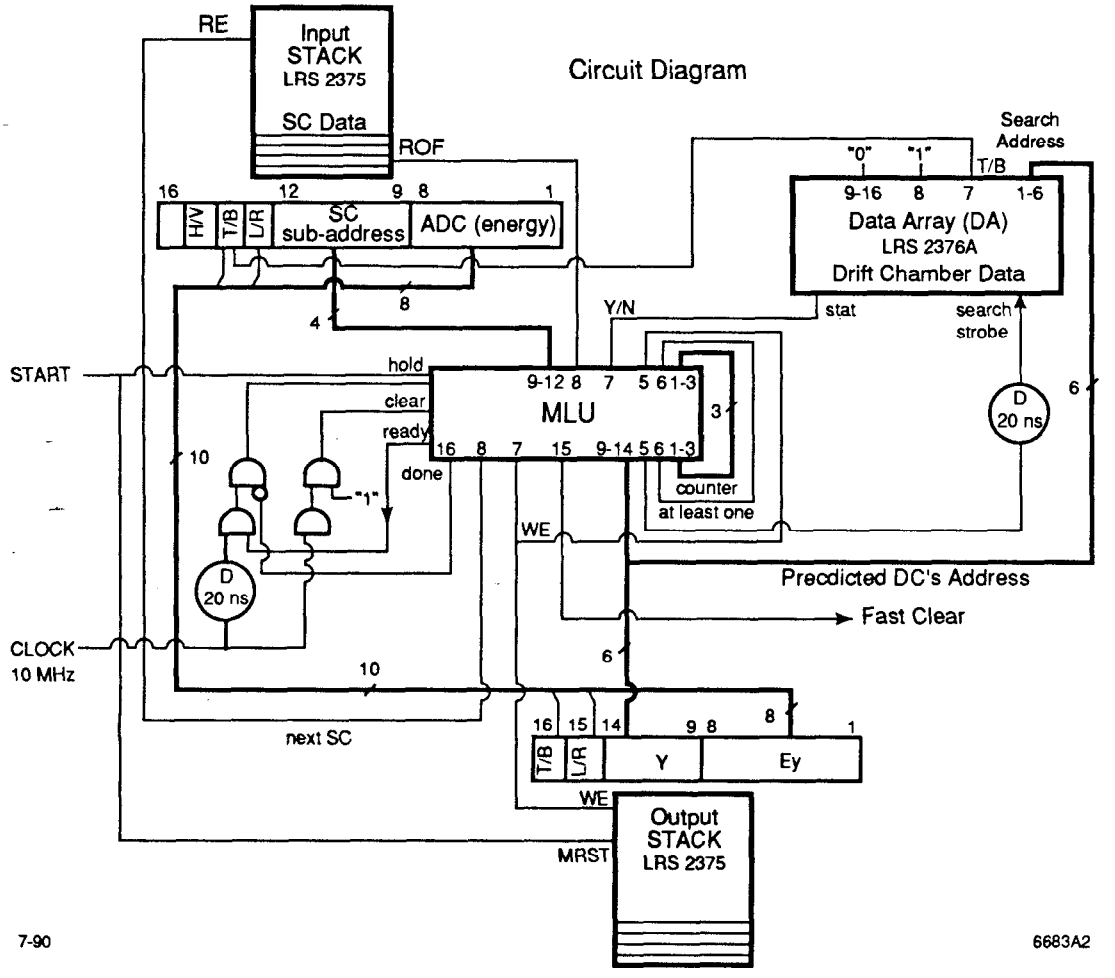
Circuit Diagram
RE
Input
STACK
LRS 2375
SC Data
ROF
16
12
9 8
1
H/V
T/B
L/R
SC
sub-address
ADC (energy)
Search
Address
"0"
"1"
T/B
9-16
8
7
1-6
Data Array (DA)
LRS 2376A
Drift Chamber Data
stat
search
strobe
Y/N
4
8
START
hold
clear
ready
done
9-12 8 7 5 6 1-3
MLU
16 8 7 15 9-14 5 6 1-3
3
counter
at least one
"1"
10
D
20 ns
6
WE
Predicted DC's Address
Fast Clear
CLOCK
10 MHz
10
6
next SC
16 15 14
9 8
8
1
T/B
L/R
Y
Ey
WE
Output
STACK
LRS 2375
MRST
7-90
6683A2

Fig. 2

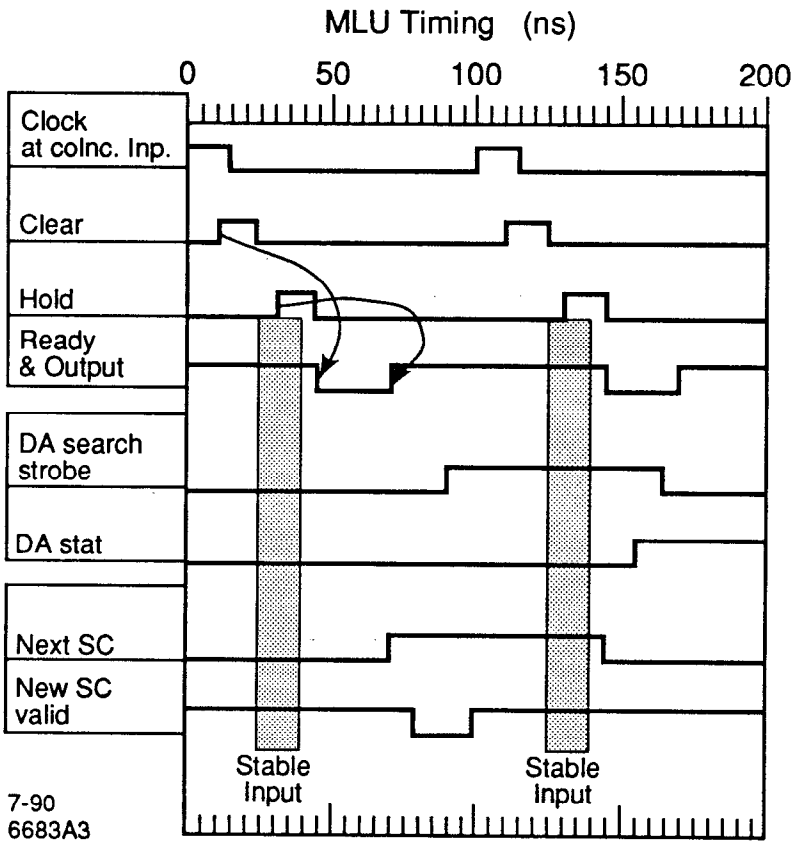
MLU Timing (ns)
0
50
100
150
200
Clock
at colnc. Inp.
Clear
Hold
Ready
& Output
DA search
strobe
DA stat
Next SC
New SC
valid
Stable
Input
Stable
Input
7-90
6683A3

Fig. 3