SMR/643 - 13

SECOND COLLEGE ON
MICROPROCESSOR-BASED REAL-TIME CONTROL -
PRINCIPLES AND APPLICATIONS IN PHYSICS
5 - 30 October 1992

*FULLY PIPELINED AND PROGRAMMABLE
LEVEL 1 TRIGGER*

D. CROSETTO
Superconducting Super Collider Laboratory
2550 Beckleymeade Avenue
Dallas, TX 75237
USA

These are preliminary lecture notes, intended only for distribution to participants.

SSCL-576

Superconducting Super Collider Laboratory

Fully Pipelined and Programmable
Level 1 Trigger

D. Crosetto and L. Love

July 1992

Fully Pipelined and Programmable Level 1 Trigger

D. Crosetto and L. Love

Superconducting Super Collider Laboratory*
2550 Beckleymeade Ave.
Dallas, TX 75237

July 1992

CONTENTS

FIGURES

Fully Pipelined and Programmable Level 1 Trigger

D. Crosetto and L. Love

Abstract

The types of detectors and the physics involved in present experiments are reaching a level of cost and complexity so great that it is preferable to implement a programmable trigger solution at all levels rather than a system realized with cabled logic. Experience demonstrates that the fine tuning on the trigger is often only achieved after running an experiment and analyzing the first data acquired. Recent advances in technology made real-time programmable algorithms down to the Level 1 trigger feasible. In this report a number of algorithms for the first level trigger have been simulated using one of the most advanced chips available. A fully-pipelined and programmable Level 1 trigger system sustaining a clock rate of 16 ns has been designed based on a modified version of the DataWave chip.

1.0 INTRODUCTION

The Superconducting Super Collider (SSC) is being built to study high-energy physics. Every 16 ns, proton beams will collide and the particles produced by the collision must be identified and studied.

Many detectors will be used to detect and identify the particles. The calorimeter (shown in Figure 1) is one of the sub-detectors to be used at the SSC. Two proton beams will collide in the center of the calorimeter sending particles to the calorimeter towers in the barrel and end caps. The amount of energy released in the collision is detected and then transferred through channels to digital processors, where the identification of particles is begun in the Level 1 triggering.

Recent advances in processor technology have made real-time programmable algorithms, down to the Level 1 trigger, feasible. This study takes already developed off-line algorithms and modifies them for on-line use with a suitable chip available today, the DataWave processing chip. The DataWave is a data-controlled RISC processor with high-bandwidth communication capability developed for video signal processing by International Telephone and Telegraph (ITT). Using a list of physics requirements (described fully in Section 4) and the DataWave architecture (described in Section 6), we have simulated the real-time algorithms of filters (Section 7), electromagnetic cluster finding (Sections 8 and 9), isolated electron finding (Sections 10 and 11), and jet-finding (Section 12) as they relate to the calorimeter.

Readers interested in the flexibility of defining the trigger tower segmentation will find an overview of examples of possible segmentation in Section 5, and an algorithm implementation using the DataWave (or Front-end processor, FEP) to sum the digital values coming from the calorimeter in Sub-section 14.2.2.

Readers who are interested in how this state-of-the-art processor technology is suitable for this type of application can find a brief overview of each algorithm as it pertains to the DataWave architecture in the first section of each section. Results of the simulation, complete with detailed timing results, are found in the last part of each section. A combined test of both isolated electrons and jets is found in Section 13. A suggested modification of the DataWave processor to a front-end processor (FEP) for a fully pipelined and programmable first level trigger sustaining 16 ns clocking is described in Section 14. Examples of programmable first level trigger algorithms, digital filters, and implementing segmentation in the FEP system (all sustaining the 16 ns clocking) are also given in Section 14. An evaluation of the overall performance of the DataWave and FEP processor array as applied to these algorithms is given in Section 15, Table 30.
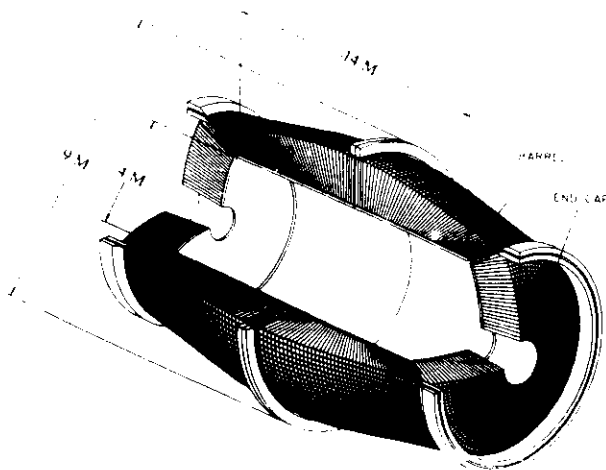
Figure 1. Conceptual View of a Calorimeter.

2.0 PURPOSE OF THE SIMULATION

The purpose of the simulation is to try and solve the rate requirements of the Level 1 trigger with a programmable chip, to determine the suitability of an advanced component available for this type of application, and then to suggest modifications to the chip to better satisfy the requirements of the application.

The types of detectors and physics involved in present experiments are reaching such a high level of cost and complexity that if the technology could meet the requirements, a programmable trigger solution at all levels is preferable rather than staying with a fixed algorithm implemented with cabled logic.

Experience demonstrates that the fine tuning of the trigger is often achieved after running an experiment and analyzing the first data acquired. With a programmable solution, it is possible to use the same electronic (commonality) chain for several experiments. For this reason, and because all physicists do not accept a specific type of trigger algorithm, a programmable solution is highly desirable.

A market survey has been conducted to identify the component best suited to fulfill the requirements of the Level 1 trigger algorithms. Presently, there does not exist a component that meets 100% of the requirements. Because of its features, the DataWave processor is considered one of the best. In order to verify its suitability, a series of typical algorithms for the first-level trigger were selected and tested on the DataWave. The DataWave component (or a modified version of it) may be used as a preprocessor of the Level 2 trigger or as pipelined processors sustaining the rate of the Level 1 trigger.

As an example of programmability, many trigger algorithms have been implemented. Among these are finding local maximum, calculating cluster and transverse energy, comparing cluster and partial sums to different thresholds, determining if an electromagnetic cluster is isolated from nearby energy deposition, and determining if a 4 × 4 or 8 × 8 matrix is a possible jet.

The algorithms proposed and tested in this report are not the only ones or necessarily the best applicable to the Level 1 trigger. They are examples of operations and correlation of data that need to be done for a Level 1 trigger decision. Is is not necessary to execute all of them, because only one algorithm is needed for each type of information (identifying electrons, jets, etc.).

Some have been simulated on a platform of a DataWave array made of DataWave chips, each containing one DataWave processor cell. Since the packaging (printed circuit board or Multi-Chip-Module) is also an important issue in realizing these types of systems, some algorithms have also been simulated on a second platform of DataWave array processors which assumed to use chips containing 16 DataWave cells each.

The system is scalable to different sizes of parallel processor arrays thus making it possible to apply the system to different calorimeter sizes and to execute algorithms of different complexities.

The importance of this simulation and study lies in the programmability of the system and the "real-time" algorithm. The starting point is always the off-line trigger algorithms that require milliseconds (ms) for execution. The challenge is to find a given "processor architecture" and "system architecture" which provide the best and most suitable (to the component) conversion of the off-line algorithm to a fast and simple "real-time" algorithm that will still have high particle-identifying efficiency. Ratios, trigonometric functions, and other time-consuming operations cannot be performed during "real-time." As a result, speed optimization techniques for real-time computations, such as precalculated look-up tables and multiplication, comparisons are used in place of ratios. Finally, a design based on a modified version of the DataWave ITT processor aimed to efficiently execute Level 1 trigger algorithms on pipelined and programmable mode has been achieved.

3.0 TRIGGER AND DATA ACQUISITION SYSTEM OVERVIEW

A complete overview of a typical trigger and data acquisition (DAQ) system is shown in Figure 2. The figure shows the relationship between the requirements (top part), the hardwired or programmable principles to realize it (middle), and the technology and communication techniques suitable to build each part (bottom). The figure points out the advantages, disadvantages, and limits of technology versus the specific requirements for the best performance and programmability.

At the top of the figure, the layout of a trigger and data acquisition (DAQ) system is shown. Millions of signals are electronically sent for all subdetectors to the trigger and DAQ system.

For the fast decision required of the Level 1 trigger, only a few subdetectors (calorimeters, shower maximum detectors, central tracking, and muon systems) send information in a macro-granularity form (signals are grouped together and the analog sum of the group is sent instead of each individual value). The input to the Level 1 trigger occurs at a rate of 16 ns and the Level 1 decision must be made in few μsec. The Level 1 trigger could be implemented on existing technology consisting of Programmable Array Logic (PAL), Field Programmable Gate Array (FPGA or XILINX), Application Specific Integrated Circuit (ASIC), DataWave-ITT or Front-End-Processor (FEP). The type of communication at this level is very simple. Signals are transported in point-to-point connections assembled on Printed Circuit Board (PCB) boards, Multi-Chip Module (MCM), etc.

The Level 1 trigger has been realized with cabled logic or with a very limited capability of programmability in past experiments. With the rapid advance in technology today, it is feasible to have a Level 1 trigger device with rather wide programmability features such as DataWave or something similar (FEP).

The top middle part of the figure illustrates the flow of the data from the detectors to the Level 2 trigger and DAQ. The full granularity information is transferred from only a subset of the subdetectors to the Level 2 triggers, while the full information from all detectors is sent to the DAQ system. The timing involved in this stage ranges from tens to hundreds of μsec. Parallel processing systems with an instruction set that gives full programmability for the Level 2 trigger decision have been used in recent experiments. The connectivity of the Level 2 trigger, which is an important and challenging issue, should be more flexible than the one used in the Level 1 trigger but less general purpose than the Scalable Coherent Interface (SCI), High Performance Parallel Interface (HPPI), or Fiber Distributed Data Interface (FDDI) used in the Level 3 trigger. The challenge comes from the need to make specific transfers efficiently and to allow easy access to each processing node.

The top right part of the figure shows the Level 3 trigger. Since the timing in this level (1 μs to 0.1 sec) is not as important as the timings in the Level 1 and Level 2 triggers; commercially available workstations can be used. The Level 3 trigger is meant to be a FARM of workstations connected by a standard hardware link and software protocol (SCI, HIPPI, FDDI, ETHERNET, etc.). The purpose of the FARM workstations is to fully, or partially, reconstruct the event in order to make the decision to store the event on tape.
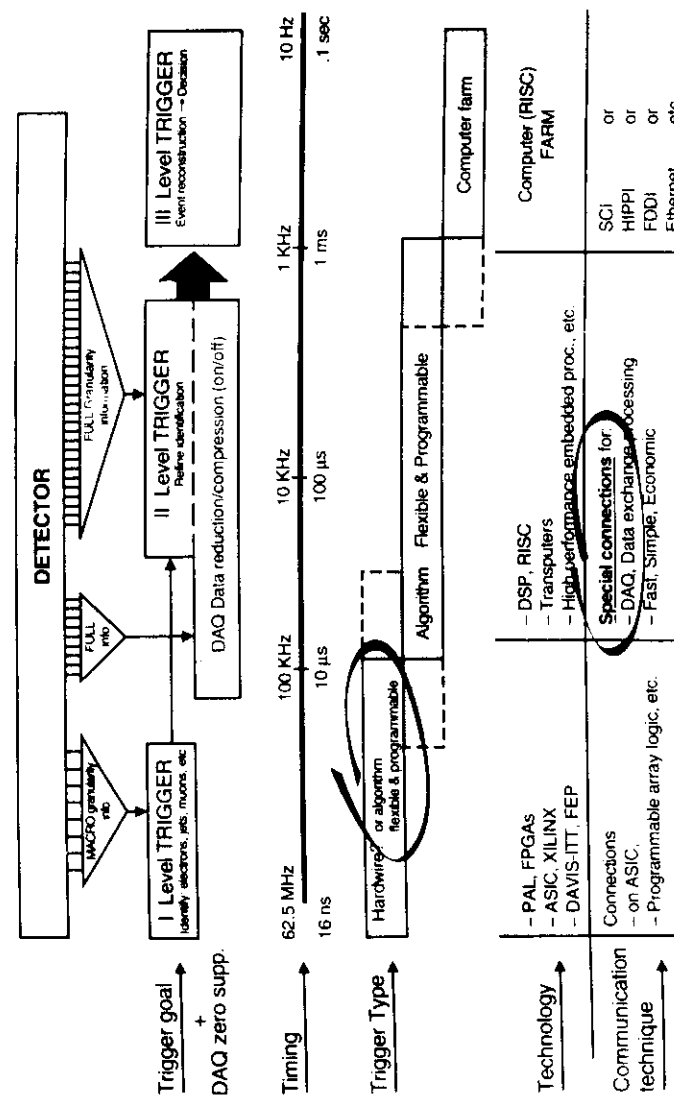
Figure 2. Technology versus Requirements/Performance, versus Programmability.

4.0 PHYSICS REQUIREMENTS

Experiments at the SSC will have on the average of 1.6 interactions during a beam crossing which occurs every 16 ns. The triggering mechanism must be able to rapidly reduce the amount of data by discarding unimportant data. Every 16 ns, the sub-detectors (including the calorimeter) send data to the Level 1 trigger, which then must be able to distinguish between events of interest and background events.

4.1 Level 1 Trigger

The Level 1 trigger will consider single objects (muons, photons and electrons) and combined objects (dileptons, and jets). Any of the above may be combined with other trigger informations (minimum bias and E_t sums). Only a few subdetectors (calorimeter, shower maximum detector, central tracking and muon system) send information to the Level 1 trigger. The calorimeter will provide Level 1 trigger information regarding electrons, photons, jets, and missing E_t (such as neutrino).

4.2 Calorimeter Trigger Information at the Level 1 Trigger

There are many conditions to test when making the Level 1 decision. To distinguishing electrons and photons, the electromagnetic ("em") trigger tower energy must be greater than a threshold, the hadronic ("had") to "em" ratio must be very small, and if isolation is to be achieved in Level 1, the surrounding towers must contain only small amounts of energy. For jet identification, the sum of a tower matrix must be tested against a threshold. To distinguish neutrinos, the E_t sum must be compared with a threshold.

There exist several methods to verify the existence of such conditions. As an example of a programmable system, a few methods that verify these conditions have been implemented using the DataWave and FEP parallel processing system array.

We have implemented two methods for cluster finding based on these algorithms. The first method requires a cluster be distinguished by a "hit" in a single tower with all of the energy of the cluster deposited in the surrounding 3×3 tower matrix. The "center" of the cluster is found by determining the tower in the cluster which contains most of the energy, which is called the local maximum. Further investigation will help to identify the type of cluster (jet, electron, etc.)[1, 2, 3]

The second method not only recognizes clusters, but also tries to distinguish between an isolated electron and a jet. An isolated electron is recognized by a large amount of energy deposited in a small area (about 1 tower wide), whereas a jet's energy is spread throughout a large matrix of calorimeter towers.

This method of electron finding also takes into consideration the possibility of a "hit" occurring between two towers. In that case, the energy of the electron would be divided between the two towers. Therefore, an electron is distinguished from other particles by a 1×2 or 2×1 region containing most of the energy, while the surrounding towers receive almost none.[2, 3, 4] Because this algorithm must be run in "real time," there is not enough time to decide whether the region is 1×2 or 2×1, and then sum the ten surrounding tower energies; this operation must be done in parallel. Therefore, an electron is considered to be isolated if the 2×2 "em" matrix contains most of the energy while the surrounding twelve "em" towers (in a 4×4 matrix) and the 16 "had" towers contain little energy.

There exist several jet-finding algorithms. A Monte-Carlo simulation executed at the Solenoidal Detector Collaboration (SDC) showed that for high energy particles, the 8×8 matrix was more efficient, while for lower energy particles, the 4×4 matrix was more reliable.[5] For this reason, both techniques are included in our algorithms.

Other information regarding the Level 1 trigger rate requirements have been learned from additional references.[6, 7, 8, 9, and 10]

4.2.1 Electronic Channel Information

The electronic channel information can be a single value preprocessed in analog form or a series of samples at high rate converted into digital form to which a digital filter will be applied.

Regardless of how the basic information generated from the calorimeter element is treated (by analog circuit or with digital filter algorithms applied upon several digitized samples per signals), the information obtained will be a value proportional to the energy deposited by the particle in that particular element.

4.2.1.1 Single Value Derived from Analog Filter

As has been done in the previous experiments, analog filters, charge preamplifiers, shapers, etc., were used to analyze the signal produced by particle interactions in calorimeter elements and generate a digitized single value proportional to the energy deposited in the calorimeter element.

All the algorithms simulated in the DataWave or FEP parallel processing array in this report assume to receive a single digitized value of this type from the calorimeter.

In past experiments, the digitized single value from each calorimeter element was sent to a look-up table that was exploiting the function of linearization (from 8-bit logarithmic to 12-bit linear value), pedestal subtraction and calibration constant. By using the DataWave or FEP parallel processing array system, (besides the on-line pattern recognition to identify the particles), one can decide to have that look-up table inside the processor cell. This allows the use of the same electronics as those used to store the program in the processor cell, to store also the precalculated values in the memory look-up table in the processor cell, thus saving the cost of building new electronics. The feasibility of using the processor cell calculation capability (multiplication by calibration constants and pedestal substractions) combined with a smaller look-up table (for operations that are time consuming in real time) for conversion algorithms (from calorimeter channel raw data to corrected values) should be studied. If this conversion is feasible by reducing the lookup table size and substituting with mathematical calculations, the cost will be reduced.

4.2.1.2 Digital Filter Upon Receiving Several Sampling Values from the Input

In the case that several digitized samples per signal are received from the calorimeter element, a digital filter program (Table 7, Table 26, Table 27) can be executed in front of the DataWave or FEP parallel processing pipelined array. A graphical representation of this is shown in Figure 3. (Section 4.2).

Furthermore, the analysis of the signal with a digital filter can be used to compute shape variables.

4.2.2 Total Energy

The total energy ("em" + "had") is defined as:

$$E_{TOT} = \sum_{i=1}^{n} c_i \cdot E_i \qquad (1)$$

where E_i is the energy of the calorimeter tower i and c_i is the calibration constant for calorimeter tower i and n = number of trigger towers. This is the case when the information is provided by an analog filter. In the case where a basic information is obtained by a series of digitized sampling at high rate, for each calorimeter signal, there will be an output result from the digital filter (e.g., as reported in Sub-section 7.3). See Sub-section 14.2.4 and Table 30 for real-time calculation performance.

4.2.3 Transverse Energy

Transverse energy is calculated by converting the 8-bit logarithmic "em" and "had" values to a linear 12-bit scale and multiplying by the sine of the tower angle of incidence found in the lookup table. Where θ_i is the angle of incidence for the calorimeter tower i and n is the number of trigger towers. See Table 30 for the real-time calculation performance.

$$E_T = \sum_{i=1}^{n} c_i E_i \sin \theta_i \quad . \tag{2}$$

4.2.4 Local Maximum Identification in a 3 × 3 Matrix

A local maximum is found when a cell's total energy is greater than or equal to all eight of its surrounding neighbor's total energy in a 3 × 3 matrix (see Figure 3).

Figure 3. Local Maximum, a Tower Value Greater Than or Equal to its Neighbors.

$$C > I_i \quad \text{for } i = 1, \dots, 8. \tag{3}$$

The sum of the energy of a tower and its eight neighbors (in a 3 × 3 matrix) must be greater than a threshold in order to be considered as a possible physics interaction.

$$Threshold < \sum_{i=1}^{8} I_i + C. \tag{4}$$

See Sections 8 and 9 for real-time calculations.

4.2.5 "Em" Cluster Finding in a 4 × 4 Matrix

This algorithm, aimed at identifying electrons, compares the sum of two adjacent towers (1 × 2 or 2 × 1 region) with a threshold. See Figure 4.
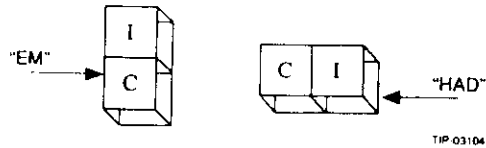
Figure 4. Front-to-back Algorithm in Cluster Finding.

$$Threshold < C_{em} + I_{em} \quad . \tag{5}$$

Options to this method include vetoing the candidate electromagnetic clusters if there is measurable energy in the hadron trigger channels behind the electromagnetic section of the calorimeter.
See Section 10 for real-time calculations.

Another option is vetoing the candidate if the electromagnetic cluster is not isolated from nearby energy channels (Figure 5). See Section 11 for real-time calculations.

Figure 5. Isolation Algorithm in Cluster Finding.

$$Threshold > \frac{(C_H + I_H)}{(C_{EM} + I_{EM})} \quad . \tag{6}$$

$$Threshold < C_H + \sum_{i=1}^{3} I_H + \sum_{i=1}^{12} O_H + \sum_{i=1}^{12} O_{EM} \quad . \tag{7}$$

4.2.6 Jet Finding

The basic granularity used to find jets is four times greater than that for the electromagnetic clusters. Thus it will be 0.64 $\Delta\eta$ × 0.64 $\Delta\Phi$ for Gammas Electrons and Muons (GEM) experiment and 0.4 $\Delta\eta$ × 0.4 $\Delta\Phi$ for Solenoidal Detector Collaboration (SDC). The SDC experiment is still investigating the basic granularity with 0.2 $\Delta\eta$ × 0.2 $\Delta\Phi$, 0.4 $\Delta\eta$ × 0.4 $\Delta\Phi$ and 0.8 $\Delta\eta$ × 0.8 $\Delta\Phi$. For the purpose of this simulation, granularities of 0.4 $\Delta\eta$ × 0.4 $\Delta\Phi$ and 0.8 $\Delta\eta$ × 0.8 $\Delta\Phi$ are assumed. See Figure 6.
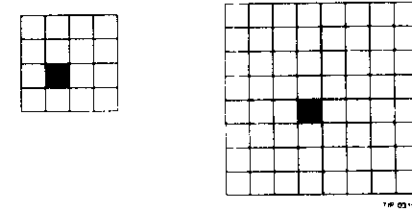
Figure 6. Jet Finding in a 4 × 4 and an 8 × 8 Region.

$$Threshold < \sum_{i=1}^{n} E_{em} + \sum_{i=1}^{n} E_H \tag{8}$$

Where n in the 4 × 4 algorithm is 16, while in the 8 × 8 algorithm is equal to 64. See Section 12 for real-time calculations.

5.0 PROCESSOR ARRAY VERSUS CALORIMETER ARRAY

5.1 Present Calorimeter Segmentation for SDC and GEM Experiments

A length-wise cross section and a side view of the end caps of the calorimeter (illustrated in Figure 1), is shown in Figure 7. In the experiments within GEM and SDC at the SSC, there is a varying calorimeter type, segmentation, and granularity of the digitized information for the Level 1 trigger. While GEM is experimenting with a $0.16 \eta \times 0.16 \Phi$ calorimeter, SDC is developing a $0.1 \eta \times 0.1 \Phi$ calorimeter. Although, in the SDC, each individual tower of the calorimeter is divided into either four (barrel) or eight (end cap) "em" sections and two "had" sections (see center right of Figure 7), for the purpose of the Level 1 trigger, the "em" sections are combined into one value and also the "had" sections are combined (see below).

The geographical representation of the calorimeter can be related to a processor array. Each calorimeter tower (consisting of an "em" part and a "had" part) has a one-to-one correspondence with a processor cell in the processor array (see bottom left of Figure 7). A description of both GEM and SDC towers as they relate to the simplified towers is shown on the right of Figure 7. The size of the processor array depends on the segmentation and granularity of the calorimeter (see Table 1).

In bold on the tower matrix array of Figure 7 is shown the types of possible investigations that can be done on such a processor array in order to identify particles and obtain the relevant information. A listing to the right of the matrix is provided.

TABLE 1. PRESENT TRIGGER TOWER SEGMENTATION FOR GEM AND SDC EXPERIMENTS.

Experiment	Subsystem	$\Delta\eta \times \Delta\Phi$	Total number of channels at full granularity	Macro-granularity for Level 1 total number of towers = total number of processors
SDC	Em Had	0.05 × 0.05 0.1 × 0.1	21 504 7 168	3584
GEM	Em Had	0.032 × 0.032 0.8 × 0.8	30 000 × 2 5000 × 4	1250

Figure 7. Processor Array Versus Calorimeter Array.

5.2 Flexibility in Defining the Trigger Tower with a Programmable Chip as DataWave or FEP

5.2.1 Trigger Analog Sums and Digital Sums

Since it may be convenient to have the possibility of flexibly defining the number of analog sums and also the trigger tower segmentation, a few examples aimed at showing the flexibility of the DataWave (or FEP) to handle various sums and segmentation are shown in Sub-section 5.2.2. As the number of analog sums increases so does the noise, which limits the reliability of the analog sum. If this noise increases to a point where the analog sum is not reliable, digital processors may be used to add the digitalized partial sums from the calorimeter, thus allowing for digital error correction.

5.2.2 Example of Other Trigger Tower Segmentation

Although present segmentation (SDC) is the one reported i⊓ Figure 7, other examples of segmentation exist. Table 2 and Figure 8 show and describe five segmentation examples.

From an electronic viewpoint, towers having the "em" and "had" section aligned are much simpler and easier to handle. Therefore, the examples that are listed show the possibility of a 9:1 or a 4:1 alignment between the "em" and "had," as well as different depths of "had" (either 2 or 4 levels).

The first three examples (0.2 $\Delta\eta$ × 0.2 $\Delta\Phi$ and 0.16 $\Delta\eta$ × 0.16 $\Delta\Phi$) can be simplified into the trigger tower (see top right Figure 8) by 12 to 26 analog sums. The other 0.1 $\Delta\eta$ × 0.1 $\Delta\Phi$ and 0.08 $\Delta\eta$ × 0.08 $\Delta\Phi$ can be simplified by 10–20 analog sums resulting in the trigger tower on the bottom right of Figure 8.

The resulting trigger tower values (4 "em" + 2 "had" or 1 "em" + 1 "had") can then be sent to a digital processor (by signals over optical fibers) which can add the values together digitally without summing the error as in analog sums.

TABLE 2. EXAMPLES OF DIFFERENT TRIGGER TOWER SEGMENTATION.

$\Delta\eta \times \Delta\Phi$	Calorimeter Elements per trigger tower at full granularity	Analog Sums	Signals Per Tower	Digital Sums	Number of trigger towers = number of processors at each pipelined stage
0.16 x 0.16	36 "em1" + 36 "em2" + 4 "had1" + 4 "had2" + 4 "had3" + 4 "had4"	18 "em" = 1 "em" 8 "had" = 1 "had"	4 "em" 2 "had"	6	1250
0.16 x 0.16	16 "em1" + 16 "em2" + 4 "had1" + 4 "had2" + 4 "had3" + 4 "had4"	8 "em" = 1 "em" 8 "had" = 1 "had"	4 "em" 2 "had"	6	1250
0.2 x 0.2	16 "em1" + 16 "em2" + 4 "had1" + 4 "had2" +	8 "em" = 1 "em" 4 "had" = 1 "had"	4 "em" 2 "had"	6	896
0.08 x 0.08	8 "em1" + 8 "em2" + 1 "had1" + 1 "had2" + 1 "had3" + 1 "had4"	16 "em" = 1 "em" 4 "hao" = 1 "had"	1 "em" 1 "had"	2	5000
0.1 x 0.1	4 "em1" + 4 "em2" + 1 "had1" + 1 "had2" +	8 "em" = 1 "em" 2 "had" = 1 "had"	1 "em" 2 "had"	2	3584
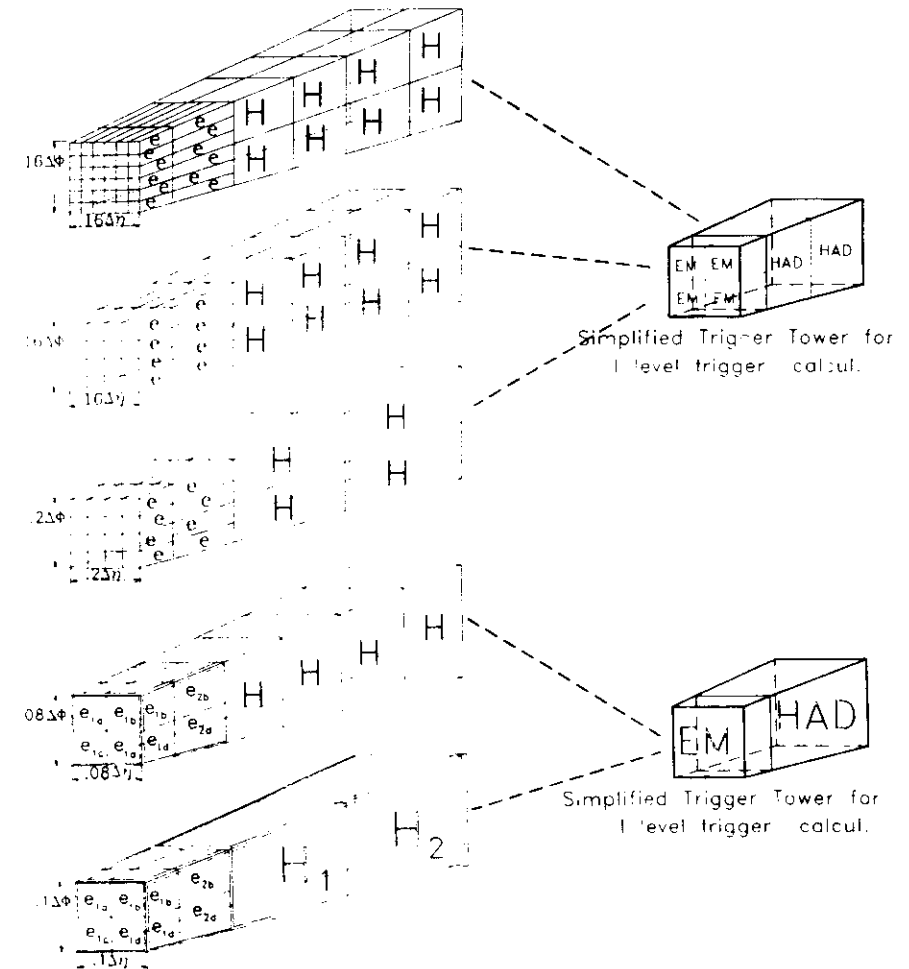
Figure 8. Examples of Different Trigger Tower Segmentation.

6.0 DATAWAVE ARCHITECTURE DESCRIPTION

The architecture of a DataWave cell is shown in Figure 9 (courtesy of ITT). The feature that differentiates the DataWave[11] chip from other processors used in filtering or image processing is the high bandwidth communication. Its three-ring bus makes it possible to receive from two ports and send to all four ports during the same clock cycle. This architecture should be taken advantage of whenever high parallelism is necessary.

The processor cell is based on a data-driven principle. The name "DataWave" was given to this processor (originally the DAVIS processor) due to data flows controlling the parallelism instead of a non-local clock timing.

A clock running at a frequency of 125 MHz synchronizes the operation of the cells. Each cell consists of a multiplier accumulating cell (MAC), arithmetic and logic unit (ALU), shift unit, register block, and program storage surrounded by a system of three-ring buses. The program can store up to 64 instructions of 48 bits each. A "deep" pipeline structure allows new instructions to be started at every clock cycle, and internal operations allow alues in the MAC and ALU to be used in the next clock cycle. An example of the use of the DataWave in a parallel processing system for calorimeter triggers is described in Reference 12.
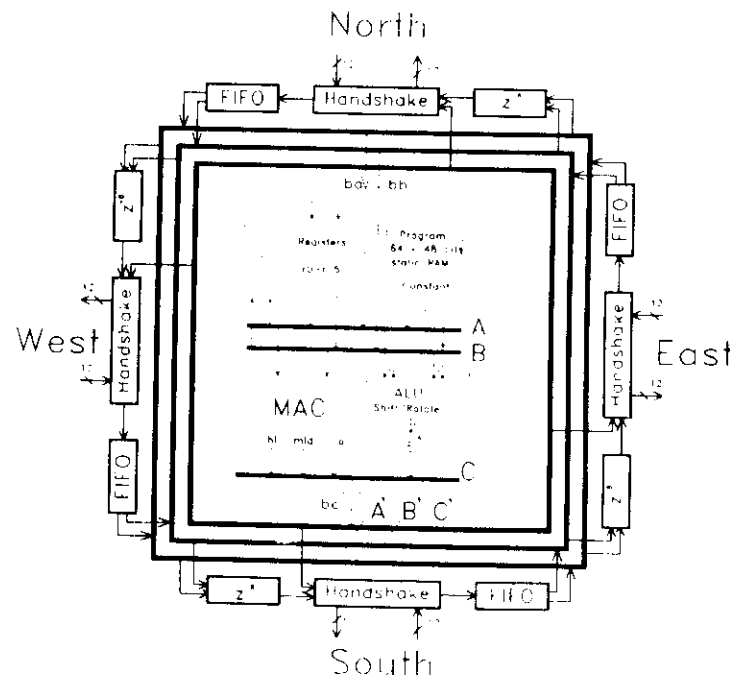
Figure 9. DataWave Cell Architecture.

6.1 DataWave Instructions

Although new instructions can be started at every clock cycle, not all instructions require the same amount of clock cycles to complete. Some instructions (involving storing a MAC operation in a register, or sending a result to another cell) require more clock cycles than simple instructions (register transfer, MAC or ALU internal operations). The difference in completion steps does not affect the pipelining of the operations.

The DataWave instruction set relevant to the cluster algorithms is as follows:

MAC operations, 6 clock cycles before result may be used

$$r5 = n * r15$$
$$r10 = acc + r5$$

Register Load, 2 clock cycles before result may be used

$$r6 = w$$
$$r0 = 256$$

Port Operations, 12 clock cycles before result may be used

$$n = w$$
$$w = n * r15$$

MAC/ALU internal operations, result may be used at the next clock cycle

$$acc = acc + r2$$

Branch on ALU, 6 clock cycles before the result may be used

$$alu = r5 - r1, bmi notamax*$$

*NOTE: Due to the pipeline structure of the cell, the cell will execute the three consecutive operations after a branch whether the cell branches or not.

6.1.1 Multiple Operations per Instruction

The 48-bit instruction word allows for multiple instructions per clock cycle. Although the DataWave processor is capable of many types of multiple operations in one clock cycle, only a few of its capabilities are relevant to this algorithm. The DataWave architecture allows each cell to receive a value from one cell and store it in a register while sending the value to all four neighboring processors. The architecture also allows the cell to use the ALU and the MAC simultaneously. For example, the cell can send a value from a register to a port and at the same time store a value into the MAC. The multiple operations per instruction also result in being able to send or load and at the same time branch (conditionally or unconditionally).

6.1.2 Waiting for an Input from a Port

If an instruction is not allowed to proceed due to lack of input at a port, the whole pipeline is stopped. If two neighboring cells send values to each other and both issue the instruction to receive the value from the other cell before the send instruction finishes the pipeline, deadlock can occur. Therefore, it is necessary to finish sending values before issuing an instruction to receive from the same cell. This results in many "nops" in a program that primarily sends and receives from all four of its neighbors. A hardware optimization to remove the pipeline between adjoining cells will increase the efficiency and timing of this algorithm.

6.1.3 Branching

Due to the pipeline structure, the next three lines after a branch will always be executed. However, instead of wasting program code and clock cycles, in some cases the branch can be placed three instruction lines before the branch needs to take place.

TABLE 3. PROGRAM EXAMPLE OF OPTIMIZING "BRANCHES".

60	sendn:	bra loop
61		n = 1
62		n = 23
63		n = r10

The sample program in Table 3 sends the three values and executes the "branch" even though the branch is written before the send statements.

Another method of using the branching delay to an advantage is by using the delay to "pass a parameter". In Table 4 the code needs to set a flag according to the reading in the ALU and then branch.

TABLE 4. PROGRAM EXAMPLE OF USING "BRANCH" TO PASS A PARAMETER.

60	r0 = 0
61	alu = r10 – r11
62	nop
63	nop
64	nop
65	nop
66	bpl check
67	bmi check
68	nop
69	nop
70	r0 = 1

The ALU is set and at the appropriate time the result is checked. If the ALU is greater than zero, the program branches to "check" executing all shown lines except for line 70; hence, r0 remains 0. If the ALU is less than zero, the program executes statement 70 before branching to "check" and sets r0 to 1.

6.1.4   MAC/ALU

Most of the statements using the MAC or ALU can be written using the other unit. The notable exceptions (to this set of algorithms) are the multiplication of two registers (acc = r5 * r1) which must use the MAC and the summing of two registers (alu = r5 + r1) which must use the ALU. Though the other instructions may interchange the ALU and MAC, the number of clock cycles before a MAC result may be used is greater than the clock cycles for the same ALU instruction. However, the MAC has greater precision.

6.2   Optimization Techniques for Program Execution Speed in Real-time Computations

Due to the time factor in a Level 1 trigger, algorithms used in the Level 3 trigger must be modified to achieve reasonable throughput.

6.2.1   Threshold Comparison and Ratio Calculation

Although threshold comparisons and ratio calculation use division off-line, division is too time consuming for "real-time" calculations. The following is the substitution for those equations.

$$\frac{(C_H + I_H)}{(C_{EM} + I_{EM})} < Threshold \Rightarrow (C_{EM} + I_{EM}) \times Threshold > (C_H + I_H) \tag{9}$$

6.2.2   Precalculated Constants

Trigonometric functions cannot be calculated in "real-time". Due to all cells always having the same Φ and η, the result of a trigonometric function can be calculated outside the algorithm and the result stored in a cell's register to be used as a constant. The following is an example of this substitution.

$$E_t = b_i \times E_i \times \sin\theta_i \Rightarrow E_T = c_i \times E_i \tag{10}$$

where c_i is the calibration constant multiplied by $\sin\theta_i$

6.3   The Davis Simulation Package

ITT has provided the Davis (original name of what is called today "DataWave") simulation package that is not only easy to use, but is also very helpful in tracing through the different algorithms. The package allows for "looking" at the data flow between cells and also at "looking" into the contents of the memory of each cell.

Figures 10 through 13 show an example of a Davis simulator data flow and a three-step example of the contents inside the Davis cell. Both these examples are taken from the code in Table 8 (Sub-section 8.2). The timings in the simulation will be different than the timings in the code, because in the code we assume that the line "s = n = e = w = r5 = n * r15" is decoded at time t = 0, although in the simulation it is not decoded until time t = 4. The reason behind this was to start all the algorithm timings at t = 0 when the cell first fetches from the calorimeter. The code preceding the fetch from the calorimeter is considered an "initialization phase" executed only once before the initial Trigger is sent. All other timings in the simulation are exactly 4 clock cycles more than the timings in the code.

The data flow window (Figure 10) shows the dataflow from all cells in the simulation at clock time t = 18 (shown in the bottom right corner). The line number of code that each cell is executing is at the bottom of each cell, with a STOP sign inside the cell if the cell is waiting for data, and a magnifying glass if the cell's memory is being displayed.

Figure 11 shows the contents of cell 0,2,1 at time t = 18. It is currently decoding the instruction, "n = r6 = e", and at time t = 17 the instruction "acc = r5, s = r4 = w" was decoded, which implies to store the content of register 5 in the MAC and at the same time store the input from West port into r4 and output the same value to the South port. As one can see, the value in the West input (top right) is "$00a" and the contents of Register 5 is "$000".
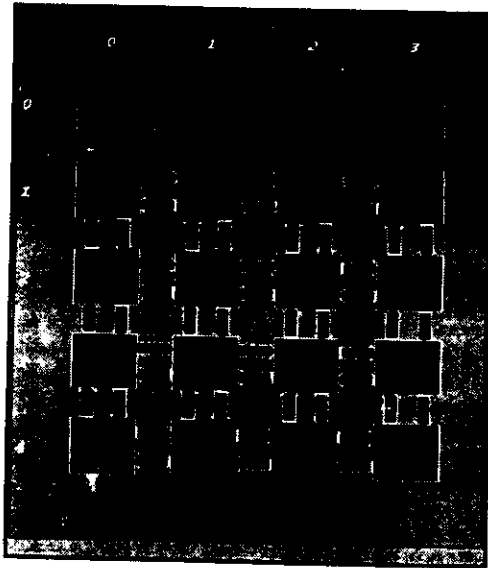
Figure 10.   Example of a Data Flow between DataWave Cells in a Step-by-step Simulation.
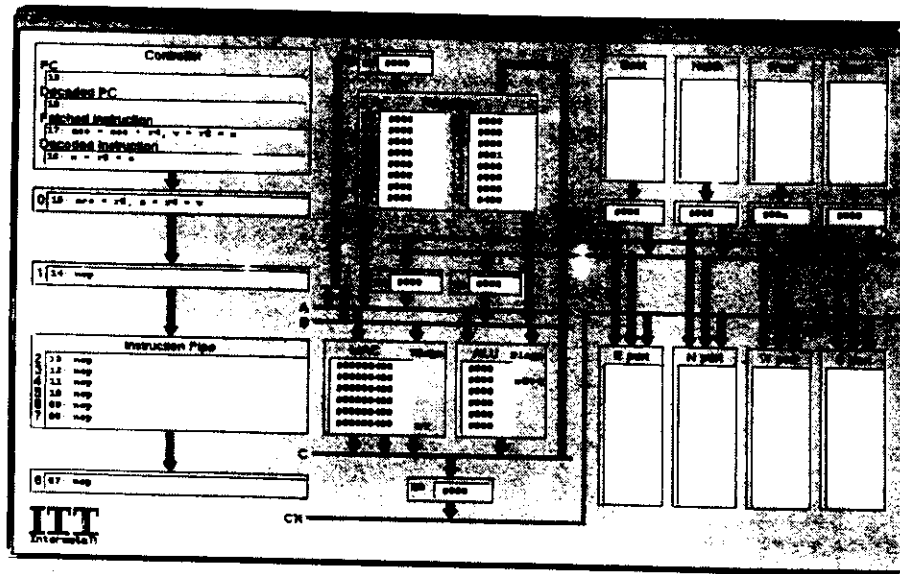
Figure 11.   Registers, ALU, MAC and Ports Content of a DataWave Cell in a Step-by-step Simulation.

Figure 12 shows the same cells memory a clock cycle later. The input from West port has moved along the AX bus and is shown in the BA box (middle of Figure 12).
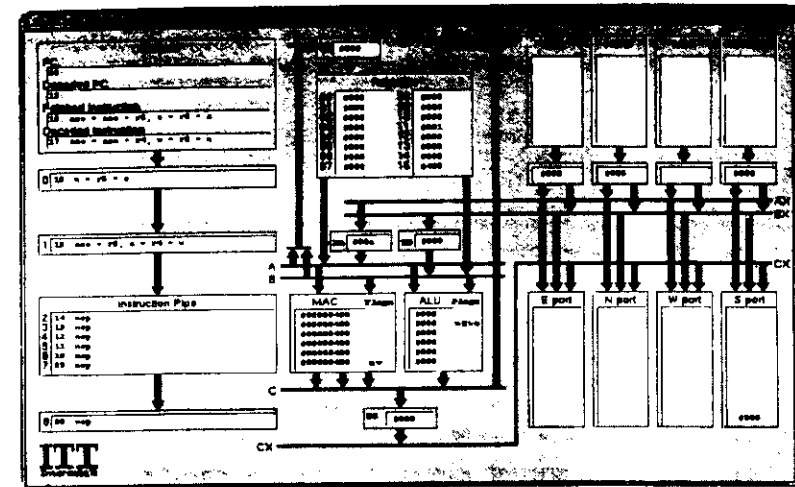
Figure 12.   Registers, ALU, MAC and Ports Content of a DataWave in the Next Step Simulation.

Figure 13 shows the cell at clock time t = 20, another step later. At this time the contents of Register 5 "$000" have been stored inside the MAC, while the input from the West has moved along the A bus to the QD box. At the next clock cycle (t = 21), the input from the West will be loaded into Register 4. The value from the West is not sent out the South port until time (t = 28).
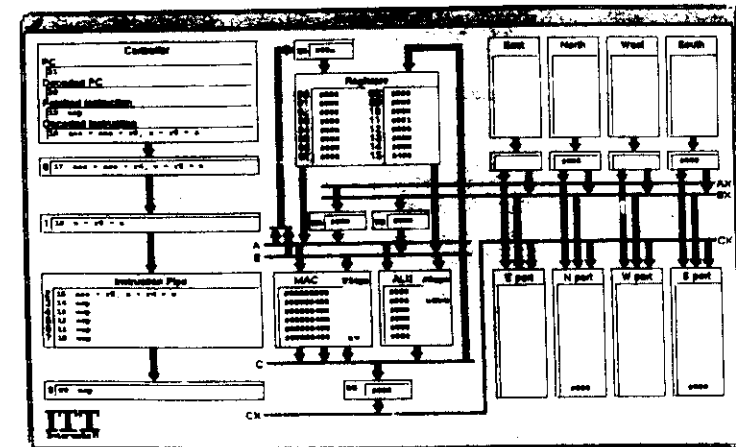
Figure 13.   Registers, ALU, MAC and Ports Content of a DataWave Cell in the Next Step Simulation.

7.0 DIGITAL FILTER EXAMPLES

Several digital filter algorithms can be applied to the trigger tower signal. The analog signal is sampled and digitized at the rate of 60 MHz and is sent to the DataWave processor.

The programmable filter capability of the DataWave processor allows physical information to be extracted. Typical filters that should be performed on the digitized samples are of the type:

$$output = \sum_{i=1}^{n} (input_i \times W_i) \tag{11}$$

where: n can vary from 5 to 8 and W_i are precalculated coefficients stored in lookup tables.

In order to give an idea of the time required to realize a digital filter with the DataWave, the following three examples are given.

7.1 Example of a Transverse Filter

A five-tap Finite Impulse Response (FIR) will input from East a value every 5 clock cycles and will output a result to the West with a latency of five clock cycles. (1 clock cycle = 8 ns in the present version and 4 ns in the future DataWave version). This 5-tap filter will sustain an input frequency of 12.5 MHz on the present version and 25 MHz in the future version. Reference to Table 5.

TABLE 5. DATAWAVE ASSEMBLER CODE EXAMPLE OF A NON-RECURSIVE FILTER.

1	FIR:	acc = r1 * w,	r12 = w	
2		acc = acc + r2 * r12,	r13 = r12	bra FIR
3		acc = acc + r3 * r13,	r14 = r13	
4		acc + acc + r4 * r14,	r15 = r14	
5		e = acc + r5 * r15		
		courtesy of ITT		

7.2 Example of Recursive Filter

In the following code, due to internal pipelines, a new value can be input from the West every 7 clock cycles. Reference to Table 6.

TABLE 6. DATAWAVE ASSEMBLER CODE EXAMPLE OF A RECURSIVE FILTER.

1	IIR:	acc = w	
2		acc = acc + r2 * r12	
3		e = r11 = acc + r1 * r11	
4		r12 = r11,	bra IIR
5		nop	
6		nop	
7		nop	
		courtesy of ITT	

7.3 Example of a Digital Filter Applied to Calorimeter Signals
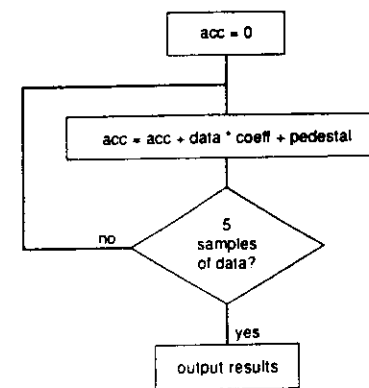
Figure 14. Flow Chart of a Digital Filter Applied to Calorimeter Signals.

An implementation with the DataWave of the filter flow-chart described in Figure 14 will imply the following code (Table 7):

TABLE 7. DATAWAVE ASSEMBLER CODE EXAMPLE OF A DIGITAL FILTER APPLIED TO CALORIMETER SIGNALS.

1	CIR:	acc = acc + w * r11	
2		acc = acc + r1	
3		acc = acc + w * r12	
4		acc = acc + r2	
5		acc = acc + w * r13	
6		acc = acc + r3	
7		acc = acc + w * r14,	bra CIR
8		acc = acc + r4	
9		acc = acc + w * r15	
10		e = acc + r5	

r11, r12, r13, r14, r15 are different coefficients and r1, r2, r3, r4, r5 are pedestal values.

Figure 15. Sampling the Calorimeter Signal for Digital Filter Computation.

8.0 ELECTRON IDENTIFICATION IN A 3 × 3 MATRIX (1-CELL PER CHIP)

8.1 Loading "Em" Data into 1-cell Per Chip Assembly

The purpose of this algorithm is to determine whether or not the calorimeter trigger tower corresponding to a cell is a local maximum of a cluster.

Each cell on the DataWave array corresponds directly with a "tower" in the calorimeter array. It is not only necessary for each cell to receive the energy of its corresponding trigger tower, but also the energies relating to the surrounding trigger towers (see Figure 16) while routing these values to other cells needing the data (see Figure 17). Once each cell contains all eight energies of the surrounding cells, the cell begins to determine whether or not it is a local maximum. Because of the 1-cell per chip packaging, all cells are loaded with the same code for routing data and finding the local maximum.

8.1.1 Receiving Data from the Calorimeter

Before the cell receives Trigger 1, the cell is connected to the calorimeter by its North port. Once the trigger is sent, the cell receives from the calorimeter the energy of the calorimeter tower corresponding to this cell. The cell then disconnects from the calorimeter and connects to its North neighbor.

"*" = fetching time from program in Cell 0,2,1

all other timing is related to time sent

Figure 16. Routing of Data to One Cell in a 3 × 3 Matrix (1-cell Per Chip Assembly).

8.1.2 Receiving and Routing of Data

At the time the cell (cell 0,2,1) receives the value from the calorimeter (t = 0), the cell multiplies the value by the calibration constant for that calorimeter tower, sending the calibrated value to the cell's four immediate neighbors (cells 0,1,1; 0,2,0; 0,2,2; 0,3,1). At the same time, all four of the cell's neighbors send the value of their calorimeter tower to the cell (cell 0,2,1) (see Figure 16). A delay of twelve cycles is required between the communication ports of two neighboring cells; hence, a cell that is sent a value from its neighbor at time t = 0 will receive the value at time t = 13.

At time t = 13 through time t = 16 the cell receives a value from its immediate neighbor and routes the value to the neighbor counterclockwise from the sending neighbor; hence, the value received from cell 0,2,2 is sent to cell 0,1,1 (see Figure 17). Since the cell's neighboring cells are executing the same algorithm, after twelve delay cycles (t = 26 through t = 29) the cell receives the values relating to its four corner neighbors' cells 0,1,0; 0,1,2; 0,3,0; 0,3,2) calorimeter trigger towers. At time t = 33 (four clock cycles for the register load) the cell is finished routing data between cells.

Figure 17. Routing of Data from One Cell in a 3 × 3 Matrix (1-cell Per Chip Assembly).

8.1.3 Finding Local Maximum in a 3 × 3 Matrix

At time t = 32, each cell begins comparing itself with all eight surrounding cells and also compares the total energy in the 3 × 3 matrix with the threshold energy for an electron. If the value of the cell is greater than all of these values and the total 3 × 3 matrix energy is greater than the threshold, the cell sends its id number and the value of its energy to the North. Otherwise the cell is not a local maximum and it sends null values to the North. All programs in all cells are finished by time t = 61.

8.2 DataWave Assembler Code and Detailed Timing Description

Each processor is loaded with the same program code for receiving, routing, and determining if the cell is a local maximum (see Table 8). In the case of routing the result of the local maximum finding, the result should be routed to a common exit point, and therefore, in order to implement this additional feature, the code should be changed. Due to the limitations of the program storage of the processor, the routing of the results of the local maximum search could not be implemented in this program. Increasing the program storage area will allow the addition of this feature.

The program shown in Table 8 has been verified by the simulator as to the correct flow of the data and to the correct timing of the instructions. All timings are shown in the program code. A "d" refers to the time that the instruction was decoded. The "u" refers to the time that the result of the operation can be used by another instruction. The "f" refers to the time that the operation is fully completed.

Register 15 is used as the threshold constant for determining whether or not the cell contains enough energy to be an electron. Register 11 is used as a calibration constant for the individual calorimeter trigger tower.

Line 4 of the program initializes r0. Since the processor cannot use a constant and a branch statement in the same instruction, the null value 0, meaning the cell is not a local maximum, is loaded into a register during a "nop" cycle. This allows line 59 to be executed as one clock cycle instead of two.

TABLE 8. DATAWAVE ASSEMBLER CODE FOR ONE CELL IN A 3 × 3 MATRIX ALGORITHM (1-CELL PER CHIP ASSEMBLY).

```
      .cell 0,2,1
1         r15 = 1024                  ;          THRESHOLD
2         R11 = 1                     ;          Calibration Constant for Cell

          ;RECEIVE FROM CALORIMETER/INITIAL SEND
3  loop:  s = n = e = w = r5 = n * r15;   d=0    u=7   f=9,11
4         r0 = 0                      ; d=1   u=3   f=5   uses "nop" to initialize r0
5         nop                         ; d=2
6         nop                         ; d=3
7         nop                         ; d=4
8         nop                         ; d=5
9         nop                         ; d=6
10        nop                         ; d=7
11        nop                         ; d=8
12        nop                         ; d=9
13        nop                         ; d=10
14        nop                         ; d=11
15        nop                         ; d=12

          ; RECEIVING/ROUTING DATA
16        s = r4 = w, acc = r5        ; d=13  u=15  f=17,24
17        n = r6 = e                  ; d=14  u=16  f=18,25
18        w = r2 = n, acc = acc   r4; d=15   u=17  f=19,26
19        e = r8 = s, acc = acc + r6; d=16   u=18  f=20,27
20        nop                         ; d=17
21        nop                         ; d=18
22        nop                         ; d=19
23        nop                         ; d=20
24        nop                         ; d=21
25        nop                         ; d=22
26        nop                         ; d=23
27        nop                         ; d=24
28        nop                         ; d=25
29        r1 = n, acc = acc + r2      ; d=26  u=28  f=30
30        r9 = s, acc = acc + r8      ; d=27  u=29  f=31
31        r3 = e, acc = acc + r1      ; d=28  u=30  f=32
32        r7 = w, acc = acc + r9      ; d=29  u=31  f=33
33        acc = acc + r3              ; d=30
          ;Sum of Cell + Surrounding Cells
34        r10 = acc + r7              ; d=31        f=34,40

          ; DETERMINING IF THE CELL IS A LOCAL MAXIMUM
35        alu = r5 - r1               ; d=32        f=37
36        alu = r5 - r2               ; d=33        f=38
37        alu = r5 - r3               ; d=34        f=39
38        alu = r5 - r4               ; d=35        f=40
39        alu = r5 - r6               ; d=36        f=41
40        alu = r5 - r7,bmi notamax   ; d=37        f=42
41        alu = r5 - r8,bmi notamax   ; d=38        f=43
42        alu = r5 - r9,bmi notamax   ; d=39        f=44
43        alu = r10 - r11,bmi notamax ; d=40        f=45
44        bmi notamax                 ; d=41
45        bmi notamax                 ; d=42
46        bmi notamax                 ; d=43
47        bmi notamax                 ; d=44
48        bmi notamax                 ; d=45
49        nop                         ; d=46
50        nop                         ; d=47  neccesary for the bmi
51        nop                         ; d=48

          ; CELL IS A LOCAL MAX
          ; send cell id to north
52  max:  n = 021                     ; d=49        f=60
          ; send cell energy to north
53        n = r10, bra loop           ; d=50        f=61
54        nop                         ; d=51
55        nop                         ; d=52  necessary for the bra
56        nop                         ; d=53

          , CELL IS NOT A LOCAL MAX
57  notamax:nop                       ;             necessary for the bmi
          ; send no cell id to north
58        n = r0
          ; send no cell energy to north
59        n = r0, bra loop            ;             send no id or energy
60        nop
61        nop                         ;             necessary for the bra
62        nop
      end
```

8.3 Result of Analysis on Electron Identification in a 3 × 3 Matrix (1-cell Per Chip Assembly)

The total time required in all the arrays (considering also the dependency of data that must be exchanged between processors) in "T" cycles (1 cycle = 8 ns in the present DataWave version and 4 ns in the future version), is shown below (See Table 9).

TABLE 9. TOTAL 3 × 3 MATRIX ALGORITHM EXECUTION TIME ON DATAWAVE
(1-CELL PER CHIP ASSEMBLY).

OPERATION	LINE NO.	TIME (CLOCK CYCLE)	TIME (NS)
Finished routing data	32	33	132
Finished summing energies*	34	34	136
Finished finding local maximum	43	45	180
Send tower id and energy	52; 53	60; 61	240; 244

Hardware optimizations (i.e., to improve the pipelining between adjacent cells and increase the storage area) might improve the timing of each processor.

9.0 ELECTRON IDENTIFICATION IN A 3 × 3 MATRIX (16-CELLS PER CHIP)

9.1 DataWave Chip Assembly

The purpose of this algorithm is to determine whether or not a cell corresponding to a calorimeter trigger tower is a local maximum of a cluster. The algorithm is implemented on DataWave chips assembled with 16-cells per chip (see Figure 18). A DataWave inter-chip bus provides the parallel I/O ports of the DataWave chip with access to the calorimeter and adjacent DataWave cells. Two bits allow each cell to connect to the inter-chip bus. The algorithm assumes that only one cell will be linked to a given bus switch during a clock cycle and allows for one cycle to disconnect a cell from the bus switch and connect another cell.
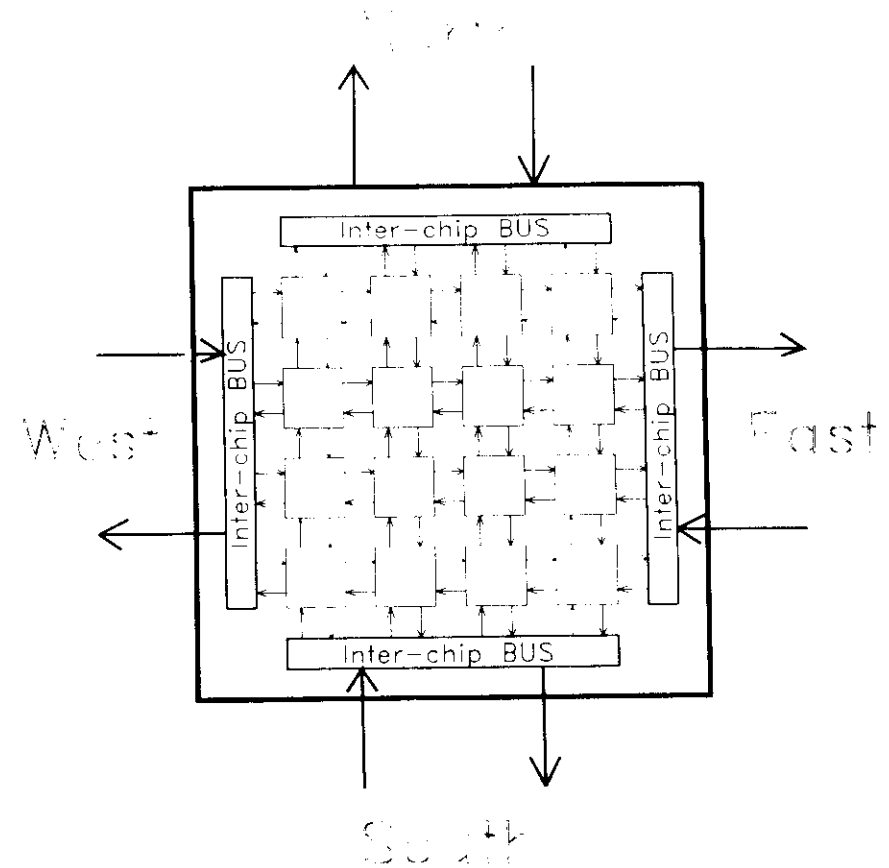
Figure 18. DataWave Chip Assembly with 16-cells Per Chip.

9.2 Loading "Em" Data into 16-cells Per Chip Assembly

The purpose of the algorithm is to find the local maxima in an array of calorimeter towers. In order to receive the values of the surrounding towers, data must be transferred between chips on an inter-chip bus. (see Figure 19). Because only one chip may be hooked up to specific bus switch at a time, this algorithm tries to use the inter-chip bus as infrequently as possible while taking advantage of the ease of communication between cells on the same chip. Due to the differences in each cell's location in relation to the four inter-chip buses, each cell in a chip is loaded with a different code.
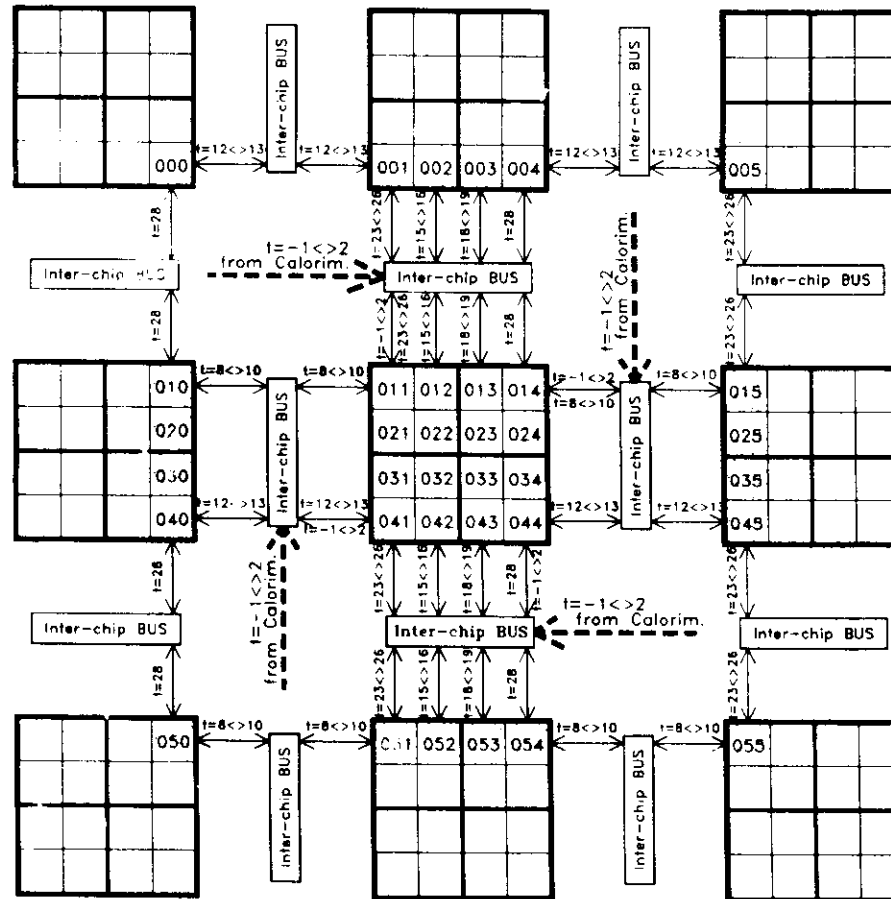
Figure 19. Inter-chip Data Flow in a 3 × 3 Matrix (16-Cells Per Chip).

9.2.1 Receiving Data from the Calorimeter

The 16 cells on the chip are divided into four groups of four cells (see Figure 19). Each group contains a "loader" (cells 0,1,1; 01,4; 0,4,1; and 0,4,4) which receives from the calorimeter all the values relating to the four cells in the group. Before Trigger 1 is sent, the "loader" cells are connected to the inter-chip bus which is connected to the calorimeter. Once the trigger is sent, all four cells receive the values of their group of four cells. For example cell 0,1,1 receives values for cell 0,1,1; 0,1,2; 0,2,1; and 0,2,2. Immediately following, the inter-chip bus disconnects from the calorimeter and connects to the adjacent DataWave chip.

9.2.2 Receiving and Routing of Data for Cell 0,1,1

Each group of four cells behave similarly except for time fluctuations due to waiting for a connection to the inter-chip bus. At time t = 0 through t = 3, cell 0,1,1 receives the data from the calorimeter and routes the data to its South and East neighbors where the values will continue to be routed to the internal neighbors (see Figures 20 and 21).

At time t = 7, cell 0,1,1 connects to the West bus switch and passes the values of cell 0,1,1 and cell 0,2,1 through the inter-chip bus to the cell 0,1,0. At the same time, cell 0,1,0 connects to its East bus switch and passes the values of cell 0,1,0 and 0,2,0 through the inter-chip bus to cell 0,1,1(see Figure 20). At time t = 13, cell 0,0,0 sends its value through the inter-chip bus to cell 0,0,1 which routes the value to cell 0,1,1 at time t = 26.

After cell 0,0,1 loads the values from the calorimeter, it sends the values of cell 0,0,1 and 0,0,2 to cell 0,0,2 (at time t = 2 through t = 3) which then routes the values through the inter-chip bus to cell 0,1,2. Cell 0,1,2 (at time t = 29 through t = 30) then sends the two values to cell 0,1,1. Cell 0,1,1 finishes routing data between cells at time t = 41 (see Figure 20).
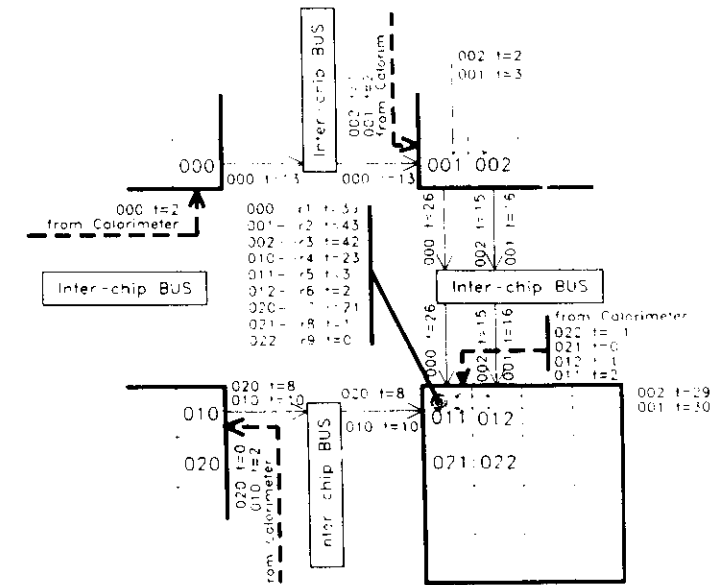
Figure 20. Routing of Data to One Cell In a 3 × 3 Matrix (16-Cells Per Chip Assembly).

Figure 21. Routing of Data from One Cell in a 3 × 3 Matrix (16—Cells Per Chip Assembly).

9.3 DataWave Assembler Code and Detailed Timing Description

Each cell in a chip contains a different routing code. However, cells in the same location in different chips contain the same code. Therefore, any amount of chips when connected by inter-chip buses can be loaded with the same set of 16 programs.

The total lines of code of eight of the cells (maximum number of lines is 78) cannot fit on the 64-word DataWave processors. The simulator has verified the routing algorithm according to the assumptions made in Sub-section 9.1. The determination of the local maximum is identical to the algorithm in the 1-cell per chip program and was verified during that simulation.

All symbols used in the program are defined in Sub-section 8.2. Registers 12 through 15 are used to store the calibration constants for each calorimeter tower that is loaded through the cell. All connections to the inter-chip bus through a bus switch on the chip are described in the comments of the program and in Figure 19.

The program example, (See Table 10) cell 0,1,1 was chosen because of its location. As shown in Figure 19, this cell's eight neighbors are contained on four chips. Cell 0,1,1 must receive information from all of these chips through the inter-chip bus.

TABLE 10. DATAWAVE ASSEMBLER CODE FOR ONE CELL IN A 3 × 3 MATRIX (16-CELLS PER CHIP ASSEMBLY).

.cell 0,1,1

1	r12 = 1	;	cal constant for cell 0,2,2			
2	r13 = 1	;	cal constant for cell 0,2,1			
3	r14 = 1	;	cal constant for cell 0,1,2			
4	r15 = 1	;	cal constant for cell 01,1			
	; RECEIVE FROM CALORIMETER					
5	loop: e = s = r9 = n * r12	; d=0	u=7	f=9,11		
6	e = s = r8 = n * r13	; d=1	u=8	f=10,12		
7	e = s = r6 = n * r14	; d=2	u=9	f=11,13		
8	e = s = r5 = n * r15	; d=3	u=10	f=12,14		
9	r11 = 1	; d=4	u=6	f=8	THRESHOLD	
10	r0 = 0	; d=5	u=7	f=9	uses "nop" to initialize	
	; RECEIVING/ROUTING DATA					
11	nop	; d=6				
12	nop	; d=7	connect to West Chip BUS			
13	w = r8, acc = r9	; d=8	f=19			
14	nop	; d=9				
15	w = r5, acc = acc + r8	; d=10	f=21			
16	nop	; d=11	disconnect from West Chip BUS			
17	nop	; d=12				
18	nop	; d=13				
19	nop	; d=14				
20	nop	; d=15				
21	nop	; d=16				
22	nop	; d=17				
23	nop	; d=18				
24	nop	; d=19				
25	nop	; d=20				
26	s = r7 = w, acc = acc + r6	; d=21	u=23	f=25,32		
27	nop	; d=22	connect to North Chip BUS			
28	n = s = r4 = w, acc = acc + r5	; d=23	u=25	f=27,34		
29	nop	; d=24				
30	nop	; d=25				
31	nop	; d=26				
32	nop	; d=27	disconnect from North Chip BUS			
33	nop	; d=28				
34	nop	; d=29				
35	nop	; d=30				
36	nop	; d=31				
37	nop	; d=32				
38	nop	; d=33				
39	nop	; d=34				
40	nop	; d=35				
	;stop	; d=36				
	;stop	; d=37				
	;stop	; d=38				
41	r1 = n, acc = acc + r7	; d=39	u=41	f=43		
	;stop	; d=40				
	;stop	; d=41				
42	r3 = e, acc = acc + r4	; d=42	u=43	f=45		
43	r2 = e, acc = acc + r1	; d=43	u=44	f=46		
44	acc = acc + r3	; d=44				

```
45        r10 = acc + r2                    ; d=45      f=48,54

          ; DETERMINING IF THE CELL IS A LOCAL MAXIMUM
46        alu = r5 - r1                     ; d=46      f=51
47        alu = r5 - r2                     ; d=47      f=52
48        alu = r5 - r3                     ; d=48      f=53
49        alu = r5 - r4                     ; d=49      f=54
50        alu = r5 - r6                     ; d=50      f=55
51        alu = r5 - r7,bmi notamax         ; d=51      f=56
52        alu = r5 - r8,bmi notamax         ; d=52      f=57
53        alu = r5 - r9,bmi notamax         ; d=53      f=58
54        alu = r10-r11,bmi notamax         ; d=54      f=59
55        bmi notamax                       ; d=55
56        bmi notamax                       ; d=56
57        bmi notamax                       ; d=57
58        bmi notamax                       ; d=58
59        bmi notamax                       ; d=59
60        nop                               ; d=60
61        nop                               ; d=61
62        nop                               ; d=62

          ;CELL IS A LOCAL MAX
          ; send cell id to north
63   max: n = 011                           ; d=63
          ; send cell energy to north
64        n = r10,bra loop                  ; d=64

          ;limits of existing chip

          ;---------------------------------------
65        nop                               ; d=65
66        nop                               ; d=66  necessary for bra
67        nop                               ; d=67

          ;CELL IS NOT A LOCAL MAX
68   notamax:nop                            ;        necessary for bmi
          ; send no cell id to north
69        n = r0
          ; send no cell energy to north
70        n = r0,         bra loop
71        nop
72        nop                                       necessary for bra
73        nop
     .end
```

9.4   Code Differences Between the Cells Within a Chip

All "loader" cells (cells 0,1,1; 0,1,4; 0,4,1; and 0,4,4) contain roughly the same code except for the lines relating to the scheduling of the inter-chip bus. The "loader" cells route data to their immediate neighbors on the same chip as well as to their immediate neighbors on adjacent chips (see Figure 19).

The cells to the left or right of the "loader" cells (cells 0,1,2; 0,1,3; 0,4,2; 0,4,3) receive data from the "loader" cells at time t = 13 <> 16 and are responsible for sending the loaded data to their North or South neighbors (cells 0,2,2; 0,2,3; 0,3,2; and 0,3,3) and for sending information needed by the adjacent chip through the inter-chip bus (see Figure 19).

The cells North or South of the "loader" cells (cells 0,2,1; 0,2,4; 0,3,1; and 0,3,4) also receive data from the "loader" cells at time t = 13 <> 16. They then send the information needed by the adjacent cells to their North or South. Hence, cell 0,2,1 sends data to cell 0,3,1 and cell 0,3,1 sends information to cell 0,2,1.

The inner four cells (cells 0,2,2; 0,2,3; 0,3,2; and 0,3,3) contain almost identical code except for the direction from which a value is sent or received. Primarily the inner cells only receive information. This is due to the fact that the cells do not begin receiving data until time t = 26. (Information sent from the "loader" cell 0,1,1 at time t = 0 will arrive at cell 0,1,2 at time t = 13 and will be sent to cell 0,2,2, arriving at time t = 26.) The only sending required of the inner cells is the exchange of data with one of its adjacent inner cells.

9.5   Result of Analysis on Electron Identification in a 3 × 3 Matrix (16-cells Per Chip Assembly)

The total time required in all the arrays (considering also the dependency of data that must be exchanged between processors) in "T" cycles (1 cycle = 4 ns), is shown in Table 11.

TABLE 11.   TOTAL 3 × 3 MATRIX ALGORITHM EXECUTION TIME ON DATAWAVE (16-CELL PER CHIP ASSEMBLY).

CELL ID	0.1.1	0.1.2	0.1.3	0.1.4	0.2.1	0.2.2	0.2.3	0.2.4	0.3.1	0.3.2	0.3.3	0.3.4	0.4.1	0.4.2	0.4.3	0.4.4
Lines of code	73	61	60	78	66	57	57	66	65	57	57	65	75	61	61	78
Routing data (in clock cycles)	47	48	46	49	55	56	56	55	51	56	56	51	46	48	46	47
Summing energies (in clock cycles)	48	49	47	50	56	57	57	56	52	57	57	52	47	49	47	48
Finding local max (in clock cycles)	59	60	58	61	67	68	68	67	63	68	68	63	58	60	58	59
Sending id & energy (in clock cycles)	75	76	74	77	83	84	84	83	79	84	84	79	74	76	74	75

The maximum time for a cell to finish routing the data is time t = 56 (which corresponds to all four inner cells). Due to the fact that all cells use the same algorithm for finding the local maximum and sending out the result, all cells finish their algorithms 28 clock cycles after finishing routing the data.

Due to the time that it takes for a cell to send a value to an adjacent cell and for that cell to receive it (13 cycles), with the existing chip the algorithm cannot increase in speed. The value for cell 0,2,2 is loaded from the calorimeter to cell 0,1,1 at time t = 0. This cell immediately sends the value to cell 0,1,2, which receives the value at time t = 13 and promptly sends the value to cell 0,2,2 which receives it at time t = 26. Immediately cell 0,2,2 sends the value to 0,2,3, which in return sends the value to 0,3,3 at time t = 39. Cell 0,3,3 receives the value at time t = 52 and uses four clock cycles to load the value into a register, which ends its routing algorithm at time t = 56. Since the value of 0,2,2 must be sent to cell 0,3,3 and there is no faster path between cell 0,1,1 (where the value is loaded) and cell 0,3,3 (where the value must be sent), the timing of the algorithm will not decrease, unless the number of clock cycles necessary to send information between two adjacent cells is decreased.

10.0 "EM" CLUSTER FINDING (TWO "EM" SUMS + FRONT-TO-BACK)

10.1 Real-time Algorithm Description for Two "Em" Sums + Front-to-back Veto

The purpose of this algorithm is to find possible electrons by searching 1×2 and 2×1 regions. Every cell checks the 1×2 region to the North and the 2×1 region to the East (see Figure 22). If the sum of the "em" energy of one of the regions is greater than a threshold, the ratio of the "had" to the "em" energy is compared. Although the ratio equation is

$$\frac{(HAD)}{(EM)} < THRESHOLD \tag{12}$$

since the program is in "real-time" the equation was rearranged into

$$(EM \times THRESHOLD) - HAD > 0 \tag{13}$$

If the comparison is greater than zero the cell is classified as a possible electron.
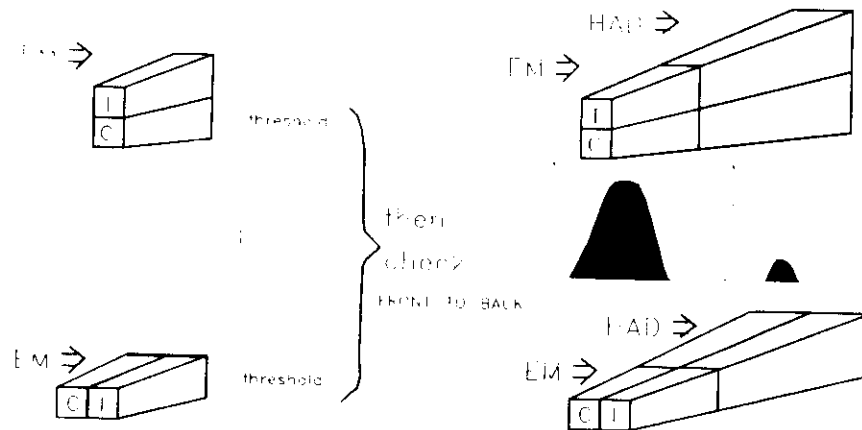
Figure 22. Electromagnetic Cluster Algorithm in a 1 × 2, 2 × 1 Region (Front-to-back).

10.2 Loading "Em" and "Had" Data to Check "Em" Sums + Front-to-back

After loading the two values ("em" and "had") from the calorimeter tower, each cell multiplies the values by the calibration constant for the individual tower and distributes the adjusted values to its South and West neighbors. Each cell then disconnects with the calorimeter and connects to its North neighbor.

The "em" value of the North cell arrives at a cell at time $t = 13$ (see Figure 23). The value is added in the Arithmetic Logic Unit (ALU) to the cell's "em" value. Due to the internal operations in the ALU, the result of the addition can be used by the ALU in the next clock cycle, even though the result will not yet be in the register. At time $t = 14$, the "em" 1×2 sum in the ALU is compared with the threshold. Because the result of this comparison cannot be used until time $t = 19$, the cell uses the next four instruction cycles for other calculations.

At time $t = 15$, the cell receives the "had" value from the North, adds it to its own "had" value, and stores the sum in a register. At time $t = 17$ the "em" value from the East is received and added to the cell's "em" value. Like the 1×2 sum, this 2×1 sum is compared with the threshold at time $t = 18$.

At time $t = 19$, the result from the ALU can be tested. If the "em" sum is greater than the threshold, the program branches to check the ratio of "had" to "em". In the three lines of code after the "branch" to the North, the program sets the value of the ACC to be equal to the ("em" * threshold – "had") value. Although the ACC will be set regardless of whether or not the 1×2 North region is a possible electron, if the North region is not a possible electron, but the 2×1 East region is a possible electron, the code will store the East ("em" * threshold – "had") value in the ACC before the ACC is tested.
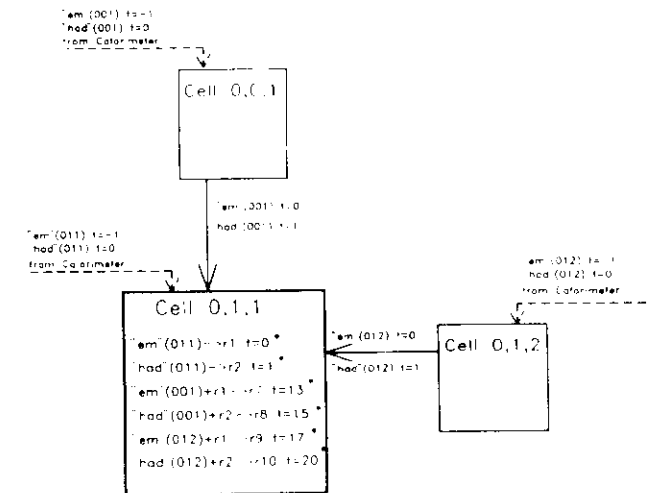
"*" = fetching time from program in Cell 0,2,1
all other timing is related to time sent

Figure 23. Routing Data to Two "Em" Cells in a 1 × 2, 2 × 1 Region.

At time $t = 19$, the "had" value from the East cell is received, it is added to the cell's "had" value, and the sum is stored in a register. The cell then waits until the result from the comparison of the 2×1 "em" value is ready to be tested. At time $t = 23$, the result is tested; if the East "em" is greater than the threshold, the program branches to check the "had" to "em" ratio, while issuing statements to place the East "had" to "em" "ratio" in the ACC.

If the 1×2 (North) "em" region was greater than the threshold, at time $t = 31$, the ACC result is compared with zero. If it is greater than zero (hence, the ratio of "had" to "em" is small) the cell is identified as a possible electron and the program branches to the code that sends the tower id, the "em" sum, and the "had" sum to the North (time $t = 36 <> 38$). Otherwise the program branches to code that sends null values to the North.

34

35

10.3 DataWave Assembler Code and Detailed Description

Each processor is loaded with the same source code that performs the operations of receiving the "em" and "had" values from the calorimeter tower, receiving the "em" and "had" values of the neighboring cells, and comparing both the 1 × 2 and 2 × 1 regions with the thresholds. The ratio of "em" to "had" is only checked if the "em" values are greater than the threshold. Each cell connects to the calorimeter through its North port before the program begins. After it finishes loading from the calorimeter the values of the "em" and "had", the cell disconnects from the calorimeter and connects to the cell to the North.

Registers 14 and 15 are used for the "em" and "had" calibration constants for the corresponding tower. Registers 5 and 6 contain the thresholds for the "em" and ("em" + threshold − "had") results. For a code listing of one cell see Table 12.

TABLE 12. DATAWAVE ALGORITHM FOR "EM" CLUSTER FINDING (TWO "EM" SUMS + FRONT-TO-BACK).

```
      .cell 0,1,1
  :           connect the North port to the calorimeter
  1           r14 = 1                        ;   calorimeter constant for "em"
  2           r15 = 1                        ;   calorimeter constant for "had"
  3           r5 = 16                        ;   "em" threshold for 1 x 2 cell region
  4           r6 = 16                        ;   ("em – "had") threshold for 1 x 2 cell region
  5    loop:  r1 = s = w = n * r14    ;d=0   load "em" value from the calorimeter,
                                      ;      multiply it by the calorimeter constant
                                      ;      and send it to West and South neighbors
  6           r2 = s = w = n * r15    ;d=1   load "had" value from the calorimeter,
                                      ;      multiply it by the calorimeter constant
                                      ;      and send it to West and South neighbors
  :           disconnect the North port from the calorimeter
  7           nop                     ;d=2
  8           nop                     ;d=3
  9           nop                     ;d=4
 10           nop                     ;d=5
 11           nop                     ;d=6
 12           nop                     ;d=7
 13           nop                     ;d=8
 14           nop                     ;d=9
 15           nop                     ;d=10
 16           nop                     ;d=11
 17           nop                     ;d=12
 18           r7 = r1 + n             ;d=13   r7 = "em" sum of cells 0,1 and 1,1
 19           alu = alu – r5          ;d=14   compare "em" sum with threshold
 20           r8 = r2 + n             ;d=15   r8 = "had" sum of cells 0,1 and 1,1
 21           nop                     ;d=16   required for 3–"nops" after "branch" in line24
 22           r9 = r1 + e             ;d=17   r9 = "em" sum of cells 1,1 and 1,2
 23           alu = alu – r5          ;d=18   compare "em" sum with threshold
 24           r10 = r2 + e,  bpl  north  ;d=19  if "em" sum (cells 0,1 & 1,1) > thrshld goto north
 25           nop                     ;d=20   necessary for the 2nd branch instr.
 26           acc = r7 * r5           ;d=21   "em" * "threshold" (1 x 2)
 27           acc = acc – r8          ;d=22   "em" * "threshold" – "had"— tested in line 41,42
 28           bpl  east               ;d=23   if "em" sum (cells 1,1 & 1,2) > thrshld goto east
 29           bmi  nosend             ;d=24   if the "em" sum is not > thrshld send null values
 30           acc = r9 * r5           ;d=25   "em" * "threshold" (2 x 1)
 31           acc = acc – r8          ;d=26   "em" * threshold – "had" — tested in line 54,55
 32           nop                     ;d=27
 33    north: nop                     ;d=23
 34           nop                     ;d=24
 34           nop                     ;d=25
 36           nop                     ;d=26
 37           nop                     ;d=27
 38           nop                     ;d=28
 39           nop                     ;d=29
 40           nop                     ;d=30
 41           bpl sendn               ;d=31   if "em" * threshold — "had"goto sendn
 42           bmi nosend              ;d=32   else goto nosend
 43           nop                     ;d=33   3– nops" after a branch
 44           nop                     ;d=34
 45           nop                     ;d=35
 46    east:  nop                     ;d=27
 47           nop                     ;d=28
 48           nop                     ;d=29
 49           nop                     ;d=30
 50           nop                     ;d=31
 51           nop                     ;d=32
 52           nop                     ;d=33
 53           nop                     ;d=34
 54           bpl sende               ;d=35   if "em" * threshold – "had" > 0 goto send
 55           bmi nc end              ;d=36   else goto nosend
 56           nop                     ;d=37
 57           nop                     ;d=38
 58           nop                     ;d=39
 59    sendn: bra loop               ;d=35   branch to loop but do next 3 lines
 60           n = 10                  ;d=36   f=47  send out tower id
 61           n = r7                  ;d=37   f=48  send out "em" energy
 62           n = r8                  ;d=38   f=49  send out "had" energy
 63    sende: bra loop               ;d=39   branch to loop but do next 3 lines
 64           n = 10                  ;d=40   f=51  send out tower id
 65           n = r9                  ;d=41   f=52  send out "em" energy
 66           n = r10                 ;d=42   f=53  send out "had" energy
 67    nosend:bra loop               ;d=28,36,or 40  branch to loop but do next 3 lines
 68           n = 0                   ;d=29,37,or 41  f=40,48,or 52  send out null tower id
 69           n = 0                   ;d=30,38,or 42  f=41,49,or 53  send out null em"
 70           n = 0                   ;d=31,39,or 43  f=42 50,or 54  send out null "had"
```

10.4 Result of Analysis on Two "Em" Sums + Front-to-back in 1-cell Per Chip Assembly (not Pipelinable)

The total time required in all the arrays (considering also the dependency of data that must be exchanged between processors) in "T"cycles (1 cycle = 8 ns in the present DataWave version and 4 ns in the future version), is shown in Table 13.

TABLE 13. TOTAL DATAWAVE ALGORITHM EXECUTION TIME FOR "EM" CLUSTERING (TWO "EM" SUMS + FRONT-TO-BACK).

Number of lines code	70	
Minimum time for 1 x 2 decision (in clock cycles)	sending tower id sending em" value sending "had' value	29 30 31
Maximum time for 1 x 2 decision (in clock cycles)	sending tower id sending em" value sending "had' value	41 42 43

If the northern 1 × 2 region is a possible electron, at time t = 36<>38 the values of the trigger tower id, the "em" energy 1 × 2 sum, and the "had" energy 1 × 2 sum will be sent out to be received (by the cell at the North) at time t = 47 <>49.

10.5 Result of Analysis on Two "Em" Sums + Front-to-back in 1-cell Per Chip Array (Pipelinable)

In order to achieve the 16 ns input rate, the DataWave processors can be arranged in several processor array stages connected in pipelined mode. Each processor must not only execute its own trigger algorithm, but also pass both input and output values through the array pipelined stages.

The pipeline begins by the first processor fetching the first two values ("em" and "had") from the calorimeter and starting its trigger algorithm on that data set. After 16 ns, the processor belonging to the first stage fetches the next values from the calorimeter and immediately sends these values to the processor of the second stage array. The processor of the second stage begins its trigger algorithm on the data set, and the rest of the data is pipelined in the same fashion. After each processor finishes its algorithm, it sends its output down the pipelined stages which the last processor array stage outputs at a rate of 16 ns (although the time to execute the trigger algorithm is much longer than 16 ns).

Due to the fact that there are only four ports on the existing DataWave processor, two ports must be able to connect to more than one cell. In the example in Table 14, every cell's North port connects to both the northern cell and the processor that is before the cell in the pipelined processor array stage (or to the calorimeter if the cell is the first processor in the pipeline). The South port, likewise, connects to both its southern cell and to the processor that is behind the cell in the pipeline processor array stage.

Table 14 shows the code for two pipelined stages similar to the code in the non-pipelined application shown in Table 12. All pipelined code for each stage is shown to the left of the trigger algorithm. The pipeline symbol (in parenthesis) is the number of the pipelined stage that will use the inputted value or that has sent the output value. Different values are marked by the four symbols:

a   inputted "em" value
b   inputted "had" value
c   outputted cell tower value
d   outputted "em" value

The line number of code is shown on the left while the timing (clock cycle) is shown at the right of Table 14.

TABLE 14. DATAWAVE ASSEMBLER CODE FOR TWO "EM" SUMS + FRONT-TO-BACK IN 1-CELL PER CHIP ARRAY (PIPELINABLE).

Line	Stage 1		Stage 11		Timing
1	loop: r1 = s = w = n * r14	(1a)	loop: r1 = s = w = n * r14	(11a)	0
2	r2 = s = w = n * r15	(1b)	r2 = s = w = n * r15	(11b)	1
3	nop		nop,	s = n (1c)	2
4	nop		nop	s = n (1d)	3
5	nop,	s = n (2a)	nop		4
6	nop,	s = n (2b)	nop		5
7	nop		nop,	s = n (2c)	6
8	nop		nop,	s = n (2d)	7
9	nop,	s = n (3a)	nop		8
10	nop,	s = n (3b)	nop		9
11	nop		nop,	s = n (3c)	10
12	nop		nop,	s = n (3d)	11
13	nop,	s = n (4a)	nop		12
14	nop,	s = n (4b)	nop		13
15	r7 = r1 + n		r9 = r1 + e,	s = n (4c)	14
16	alu = alu - r5		alu = alu - r5	s = n (4d)	15
17	r9 = r1 + e,	s = n (5a)	r7 = r1 + n		16
18	alu = alu - r5,	s = n (5b)	alu = alu - r5		17
19	nop		nop,	s = n (5c)	18
20	r8 = r2 + n		r10 = r2 + e	s = n (5d)	19
21	bpl north, r10 = r2 + e	s = n (6a)	bpl east, r8 = r2 + e		20
22	acc = r7 * r5	s = n (6b)	acc = r9 * r5		21
23	bpl east		bpl north,	s = n (6c)	22
24	bmi nosend1, acc = acc - r8		bmi nosend1, acc = acc - r8	s = n (6d)	23
25	acc = r9 + r5,	s = n (7a)	acc = r7 * r5		24
26	acc = acc - r8	s = n (7b)	acc = acc - r8		25
27	nop		nop.	s = n (7c)	26
28	north: nop,	s = n (7a)	east: nop		24
29	nop,	s = n (7b)	nop		25
30	nop		nop,	s = n (7c)	26
31	nop		nop,	s = n (7d)	27
32	nop,	s = n (8a)	nop		28
33	nop,	s = n (8b)	nop		29
34	nop		nop,	s = n (8c)	30
35	nop		nop,	s = n (8d)	31
36	bmpl sendn	s = n (9a)	bmpl sende		32
37	bmmi nosend2	s = n (9b)	bmmi nosend2		33
38	nop		nop,	s = n (9c)	34
39	nop		nop,	s = n (9d)	35
40	nop	s = n (10a)	nop		36
41	east: nop		north: nop,	s = n (7c)	26
42	nop		nop,	s = n (7d)	27
43	nop,	s = n (8a)	nop		28
44	nop.	s = n (8b)	nop		29
45	nop		nop,	s = n (8c)	30
46	nop		nop,	s = n (8d)	31
47	nop,	s = n (9a)	nop		32
48	nop	s = n (9b)	nop		33
49	bmpl sende		bmpl sendn,	s = n (9c)	34
50	bmmi nosend3		bmmi nosend3	s = n (9d)	35
51	nop	s = n (10a)	nop		36
52	nop	s = n (10b)	nop		37
53	nop		nop.	s = n (10c)	38
54	sndn: nop,	s = n (10a)	sende: nop		36
55	nop,	s = n (10b)	nop		37
56	nop		nop,	s = n (10c)	38
57	nop		nop.	s = n (10d)	39
48	bra loop.	s = n (11a)	bra loop		40
49	nop,	s = n (11b)	nop		41
50	s = 23		s = 23	(11c)	42
51	s = r7		s = r79	(11d)	43
52	sende: nop		sendn: nop,	s = n (10c)	38
53	nop		nop,	s = n (10d)	39
54	bra loop.	s = n (11a)	bra loop		40
55	nop,	s = n (11b)	nop	(11)	41
56	s = 23,		s = 23	(11d)	42
57	s = r9		s = r7		43
58	nosend1: nop		nosend1: nop	,s = n (7d)	27
59	nop,	s = n (8a)	nop		28
60	nop,	s = n (8b)	nop		29
61	nop		nop,	s = n (8c)	30
62	nop		nop,	s = n (8d)	31
63	nop,	s = n (9a)	nop		32
64	nop,	s = n (9b)	nop		33
65	nop		nop.	s = n (9c)	34
66	nop		nop	s = n (9d)	35
67	nop,	s = n (10a)	nop		36

68	nosend2:	nop,	s = n (10b)noesend2:nop			37
69		nop		nop,	s = n (10c)	38
70	nosend1:	nop		norsend3: nop,	s = n (10d)	39
71		bra loop,	s = n (11a)	bra loop		40
72		nop,	s = n (11b)	nop		41
73		s = 0	(1c)	s = 0	(11c)	42
/4		s = 0	(1d)	s = 0	(11d)	43

Due to the fact that input values are pipelined at different times than output values, the pipeline code will be different for all stages in the pipeline. Since the North and South ports can only be used for either connecting to the North/South neighbors or connecting to the pipeline stages, all lines of the trigger algorithm code involving these ports must be placed at different clock times than the pipelined stage code. (See the difference between stage 1 code and stage 11 code in lines 15–18 in Table 14). Although this limitation did not make the pipeline two "em" sum + front-to-back algorithm longer than the non-pipelined (both send the last output value at time t = 43), it is feasible that inserting algorithms with more interaction with neighboring cells into a pipeline stage structure will cause the need for more lines of code to account for this limitation.

11.0 "EM" CLUSTER FINDING (ISOLATION) IN A 4×4 MATRIX (1-CELL PER CHIP)

11.1 Real-time Algorithm Description for "Em" Cluster Isolation

The purpose of this algorithm is to further enhance the electron-finding algorithm by requiring a possible electron to be isolated from surrounding energy. To accomplish this goal a 4×4 matrix is used (see Figure 24). The inner 2×2 "em" matrix containing energy above threshold is considered to be a possible electron. The outer twelve towers of the 4×4 matrix must contain small amounts of energy in order to confirm the trigger tower in a 2×2 matrix as a possible electron. Our isolation algorithm consists of summing the 4×4 matrix energy (both "em" and "had") and except for the 2×2 "em" energy.

11.2 Loading "Em" and "Had" Data and Routing Criteria to Check Isolation

In order to find the (4×4) matrix "em" + (4×4) matrix "had" – (2×2) "em" sum, each cell must receive the "em" and "had" values of the 4×4 matrix (see Figure 25). The received values are added to the ALU unless the value is a part of the 2×2 "em" matrix; in which case the value is routed to a neighboring cell. While each cell is being sent the values of the 4×4 matrix, its own "em" and "had" values are being sent to each cell in the 4×4 matrix that requires its value (see Figure 26).

The main criterion used to develop this algorithm on the DataWave chip is to always pass the value that is farthest away as soon as possible. As seen in Figure 18, the cell 0,0,5 is the farthest away from cell 0,2,3. Therefore, the data from that chip (as it flows from cell 0,0,5 to cell 0,2,3) is always sent out at the time that a cell receives it, while other values might need to wait a few cycles if more than one value arrives during a clock cycle.
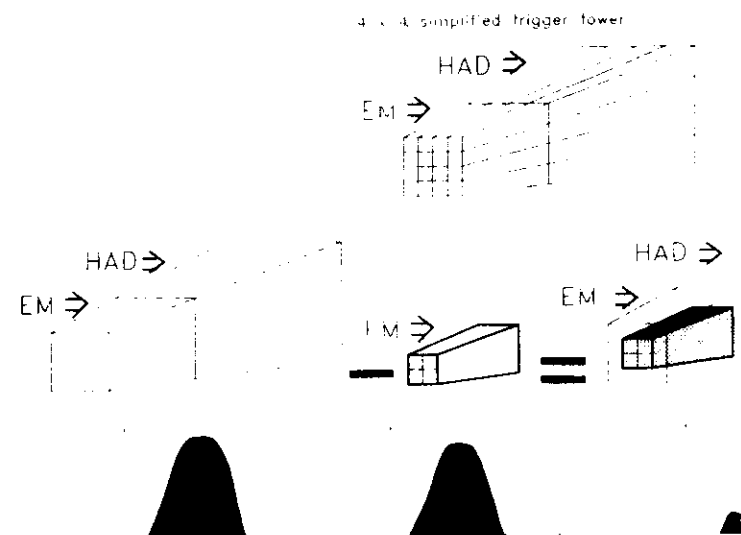
Figure 24. Isolation Cluster Algorithm in a 4×4 Matrix.
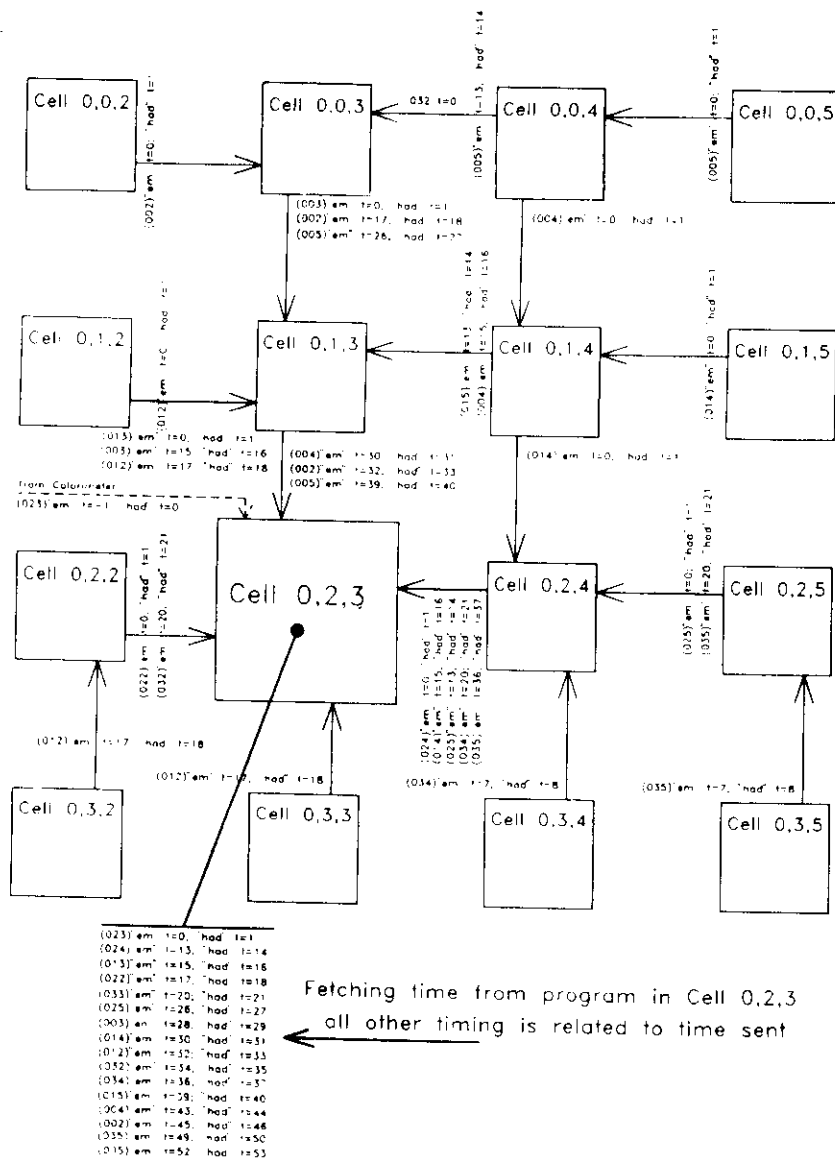
Figure 25. Routing Data to One Cell for Isolation Check in a 4 × 4 Matrix.

Fetching time from program in Cell 0,2,3 all other timing is related to time sent

The routing of the data is not unique, many other routes can be taken between two cells. However, since the last data arrives at time t = 52, which is the first possible time for it to arrive (assuming 13 cycles to transfer data between cells), no other routing procedure would take less time.

Each cell connects to the calorimeter through its North port before the program begins. After it finishes loading the "em" and "had" values from the calorimeter, the cell disconnects from the calorimeter and connects to the cell to the North.

After receiving the values from the calorimeter, each cell multiplies the values by the calibration constant for the individual tower and stores the "had" value in the ALU (which will be used to sum the energy of the 4 × 4 matrix). Then the cell distributes the adjusted values to the East, West, and South. Once the cell disconnects from the calorimeter and connects to its northern neighbor, the cell sends the values North.
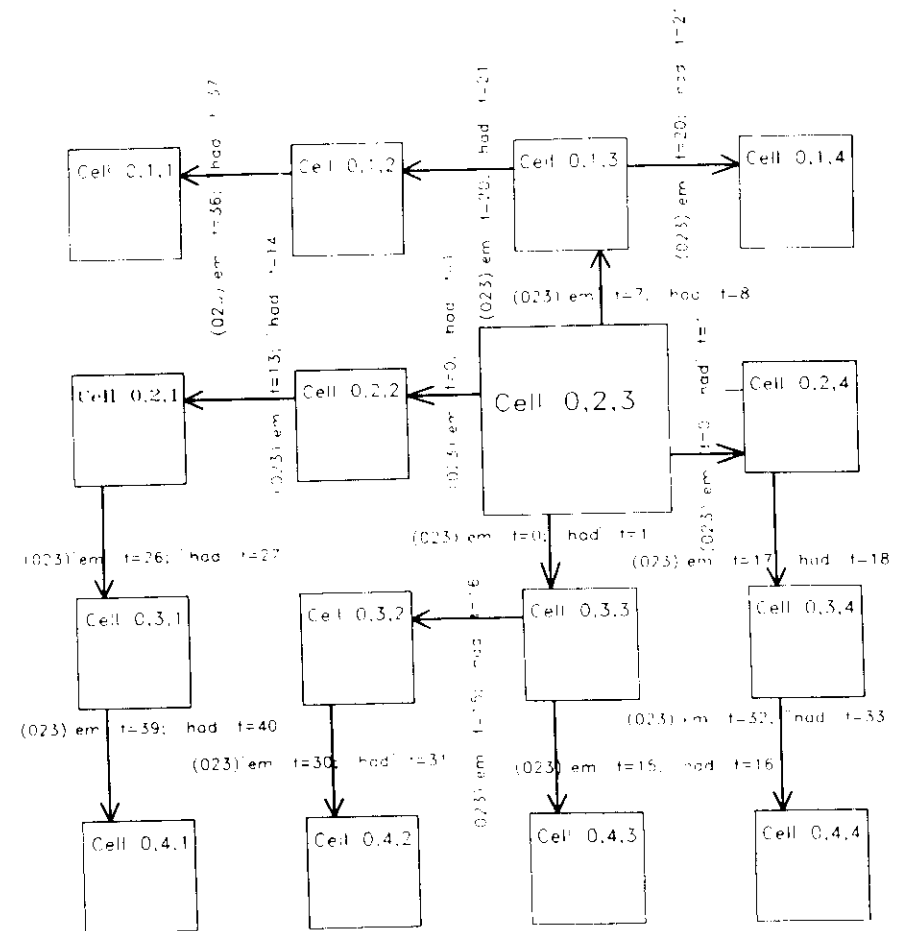
Figure 26. Routing Data from One Cell for Isolation Check in a 4 × 4 Matrix.

42

43

At time t = 17 each cell begins receiving the "em" and "had" values from its neighboring cells and adding the value to the ALU. If the values are needed by other cells, the cell sends them out. Figure 26 shows the routing of all cells in relation to cell 0,2,3.

At time t = 56, all distribution is finished and the ALU contains the sum of the 4 × 4 "em" and "had" matrix except for the 2 × 2 "em" matrix. This sum is then compared with a threshold. If the sum is less than the threshold, then the energy is isolated and the tower id and sum are sent to the North, otherwise null values are sent.

11.3   DataWave Assembler Code and Detailed Timing Description

Each cell is loaded with the same program that accomplishes the task of distributing the "em" and "had" values and comparing the 4 × 4 matrix sum with the threshold. Because the code cannot fit in a 64-word space, the code was tested in segments: one test for the distribution algorithm and one test for the comparison algorithm. A combined listing is shown in Table 15.

Registers 14 and 15 are used as the calibration constants for the "em" and "had" portions of the trigger tower relating to each cell. Register 11 is used as the threshold constant.

All values that are received and added to the ALU are marked in the comments. If a value is received, but not added to the ALU because it is an "em" value of the 2 × 2 matrix, the line of the receipt is marked with "(2 × 2)".

TABLE 15.   DATAWAVE ASSEMBLER CODE FOR 4 × 4 MATRIX ISOLATION.

```
      .cell 0,2,3
      connect to the calorimeter through the north port
1     r14 = 1
2     r15 = 1                          ; calorimeter constant for cell 2,3 "em"
3     r11 = 1                          ; calorimeter constant for cell 2,3 "had"
4  loop:  e = w = s = r1 = n * r14     ; threshold constant
5     e = w = s = r2 = n * r15         ;d=0   fetch & send 2,3 "em" value e,w,s
                                       ;d=1   fetch & send 2,3 "had" value e,w,s
      disconnect from the calorimeter and connect to the North neighbor (1,3)
6     nop
7     nop                              ;d=2
8     nop                              ;d=3
9     nop                              ;d=4
10    nop                              ;d=5
11    n = r1                           ;d=6
12    n = r2                           ;d=7   send 2,3 "em" value n
13    alu = r2                         ;d=8   send 2,3 "had" value n
14    nop                              ;d=9   set alu to the "had" value of cell 2,3
15    nop                              ;d=10
16    nop                              ;d=11
17    w = e                            ;d=12
18    w = e,     alu = alu + e         ;d=13  receive "em" value of cell 2,4 (2x2)
19    w = s = n                        ;d=14  receive "had" value of cell 2,4
20    w = s = n,  alu = alu + n        ;d=15  receive "em" value of cell 1,3 (2x2)
21    s = w,     alu = alu + w         ;d=16  receive "had" value of cell 1,3
22    s = w,     alu = alu + w         ;d=17  receive "em" value of cell 2,2
23    nop                              ;d=18  receive "had" value of cell 2,2
24    e = w = s,  alu = alu + s        ;d=19
25    e = w = s,  alu = alu + s        ;d=20  receive "em" value of cell 3,3
26    nop                              ;d=21  receive "had" value of cell 3,3
27    nop                              ;d=22
28    nop                              ;d=23
29    nop                              ;d=24
                                       ;d=25
```

44

```
30    s = e,     alu = alu + e         ;d=26  receive "em" value of cell 2,5
31    s = e,     alu = alu + e         ;d=27  receive "had" value of cell 2,5
32    alu = alu + n                    ;d=28  receive "em" value of cell 0,3
33    alu = alu + n                    ;d=29  receive "had" value of cell 0,3
34    s = e                            ;d=30  receive "em" value of cell 1,4 (2x2)
35    s = e,     alu = alu + e         ;d=31  receive "had" value of cell 1,4
36    s = n,     alu = alu + n         ;d=32  receive "em" value of cell 1,2
37    s = n,     alu = alu + n         ;d=33  receive "had" value of cell 1,2
38    alu = alu + w                    ;d=34  receive "em" value of cell 3,2
39    alu = alu + w                    ;d=35  receive "had" value of cell 3,2
40    w = e,     alu = alu + e         ;d=36  receive "em" value of cell 3,4
41    w = e,     alu = alu + e         ;d=37  receive "had" value of cell 3,4
42    nop                              ;d=38
43    s = n,     alu = alu + n         ;d=39  receive "em" value of cell 1,5
44    s = n,     alu = alu + n         ;d=40  receive "had" value of cell 1,5
45    nop                              ;d=41
46    nop                              ;d=42
47    alu = alu + n                    ;d=43  receive "em" value of cell 0,4
48    alu = alu + n                    ;d=44  receive "had" value of cell 0,4
49    alu = alu + n                    ;d=45  receive "em" value of cell 0,2
50    alu = alu + n                    ;d=46  receive "had" value of cell 0,2
51    nop                              ;d=47
52    nop                              ;d=48
53    alu = alu + e                    ;d=49  receive "em" value of cell 3,5
54    alu = alu + e                    ;d=50  receive "had" value of cell 3,5
55    nop                              ;d=51
56    alu = alu + n                    ;d=52  receive "em" value of cell 0,5
57    r10 = alu + n                    ;d=53  f=56,62 receive "had" value
                                       :        of cell 0,5 and place sum of 4x4em +
                                       :        4x4h - 2 x 2em in r10
58    alu = alu - r11                  ;d=54  compare the value of the sum with
                                       :        the threshold
59    nop                              ;d=55
60    nop                              ;d=56
61    nop                              ;d=57
62    nop                              ;d=58
63    bmi noimp                        ;d=59  if the thrshld >then goto noimp
64    nop                              ;d=60
65    nop                              ;d=61
66    nop                              ;d=62
67  imprtnt:bra loop                   ;d=63  if the sum > the threshold
68    n = 23                           ;d=64  f=75    then send tower id & energy
69    n = r10                          ;d=65  f=76    found to the North
70    nop                              ;d=66
71  notimp:bra loop                    ;d=63  if the threshold > value then send null
72    n = 0                            ;d=64  f=75    values to the North
73    n = 0                            ;d=65  f=76
74    nop                              ;d=66
      .end
```

45

11.4 Result of Analysis on 4 × 4 Matrix Isolation in 1-cell Per Chip Assembly

The total time required in all the arrays (considering also the dependency of data that must be exchanged between processors) in "T"cycles (1 cycle = 8 ns in the present DataWave version and 4 ns in the future version), is shown in Table 16.

TABLE 16. TOTAL 4 × 4 MATRIX ANALYSIS FOR ISOLATION ALGORITHM ON DATAWAVE.

Number of lines	74
Finished routing "em" and "had" value (in clock cycles	56
Finished sending tower id and energy (in clock cycles)	76

Reducing the time to route data between cells as well as the time to test the result of an ALU or ACC result will significantly reduce the final timings of this algorithm.

12.0 JET FINDING

12.1 Real-time Algorithm Description for Jet Finding

The purpose of these algorithms is to find possible jets by searching 4 × 4 and 8 × 8 calorimeter trigger tower matrixes. Every cell must receive the "em" and "had" values of each cell in its 4 × 4 matrix, while sending and routing other "em" and "had" values to its neighboring cells. The values are routed in the same way the values are routed in the electron isolation algorithm (See Figures 25 and 26).

After the 4 × 4 matrix values have been received in the 4 × 4 algorithm, the sum of the values is compared with the threshold. In the 8 × 8 algorithm, each cell sends the 4 × 4 energy sum to a center cell (see Figure 27) which combines the 4 × 4 sums into the 8 × 8 sum and compares the sum with the threshold.

12.2 DataWave Assembler Code and Detailed Description for the 4 × 4 Jet Finding Algorithm

Both the 4 × 4 and the 8 × 8 algorithms behave similarly to the algorithm for the electron isolation. The difference is that in the jet-finding algorithms all "em" and "had" in the 4 × 4 matrix are added to the sum of the energy (see Table 17)

TABLE 17. DIFFERENCES BETWEEN THE 4 × 4 ELECTRON ISOLATION AND 4 × 4 JET FINDING (DATAWAVE CODE).

11	n = r1		;d=7	send 2,3 "em" value n
12	n = r2		;d=8	send 2,3 "had" value n
13	alu = r1		;d=9	set alu to the "had" value of cell 2,3
14	alu = alu + r2		;d=10	
15	nop		;d=11	
16	nop		;d=12	
17	w = e,	alu = alu + e	;d=13	receive "em" value of cell 2,4 (2x2)
18	w = e,	alu = alu + e	;d=14	receive "had" value of cell 2,4
19	w = s = n,	alu = alu + n	;d=15	receive "em" value of cell 1,3 (2x2)
20	w = s = n,	alu = alu + n	;d=16	receive "had" value of cell 1,3
21	s = w,	alu = alu + w	;d=17	receive "em" value of cell 1,3
22	s = w,	alu = alu + w	;d=18	receive "had" value of cell 2,2
23	nop		;d=19	
24	w = s,	alu = alu + s	;d=20	receive "em" value of cell 3,3
25	w = s,	alu = alu + s	;d=21	receive "had" value of cell 3,3
26	nop		;d=22	
27	nop		;d=23	
28	nop		;d=24	
29	nop		;d=25	

30	s = e,	alu = alu + e	;d=26	receive "em" value of cell 2,5
31	s = e,	alu = alu + e	;d=27	receive "had" value of cell 2,5
32		alu = alu + n	;d=28	receive "em" value of cell 0,3
33		alu = alu + n	;d=29	receive "had" value of cell 0,3
34	s = e,	alu = alu + e	;d=30	receive "em" value of cell 1,4 (2x2)
35	s = e,	alu = alu + e	;d=31	receive "had" value of cell 1,4

At time t = 0<>1 each cell loads the "em" and "had" value from the calorimeter and sends the values to its East, West, and South neighbors. After disconnecting from the calorimeter and connecting to its North neighbor, the cell sends the value North.

Each cell receives and routes values from other cells between time t = 17 <> 59. At time t = 56 all cells contain the sum of the 4 × 4 "em" and "had" matrix. This value is compared with a threshold. If the value is greater the tower id, the total 4 × 4 energy of the possible jet is sent to the North.

12.3 DataWave Assembler Code and Detailed Description for the 8 × 8 Jet Finding Algorithm

After the 4 × 4 sum has been totaled, the 8 × 8 algorithm routes the sums to their final destinations, the center of the 8 × 8 matrix (see Figure 27). Table 18 shows the final 8 × 8 routing code, which can be inserted between line 56 and line 57 of the 4 × 4 jet finding code.
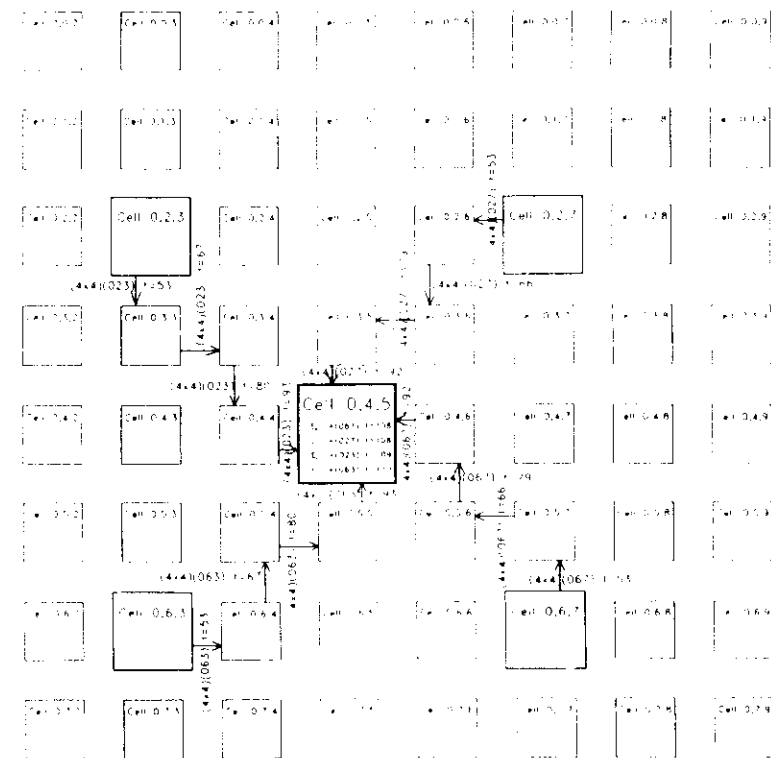

Figure 27. Routing Data for Jet Finding in a 8 × 8 Matrix.

TABLE 18. ROUTING CODE FOR THE DATAWAVE 8 × 8 JET FINDING ALGORITHM.

ROUTING OF 4x4 JET VALUES

57	n = e = s = w = r10 = alu + n	;d=53 receive "had" value of cell 0,5
		; place sum of 4x4em + 4x4h in r10
		and send to all four neighbors
58	nop	;d=54
59	nop	;d=55
60	nop	;d=56
61	nop	;d=57
62	nop	;d=68
63	nop	;d=59
64	nop	;d=60
65	nop	;d=61
66	nop	;d=62
67	nop	;d=63
68	nop	;d=64
69	nop	;d=65
70	w = s, s = e	;d=66 send "jet 4x4" value of cell 3,2 to w &
		send "jet 4x4" value of cell 2,3 to s
71	e = n, n = w	;d=67 send "jet 4x4" value of cell 1,2 to e &
		send "jet 4x4" value of cell 2,1 to n
72	nop	;d=68
73	nop	;d=69
74	nop	;d=70
75	nop	;d=71
76	nop	;d=72
77	nop	;d=73
78	nop	;d=74
79	nop	;d=75
80	nop	;d=76
81	nop	;d=77
82	nop	;d=78
83	n = e, w = n	;d=79 send "jet 4x4" value of cell 3,3 to n &
		send "jet 4x4" value of cell 1,3 to w
84	s = w, e = s	;d=80 send "jet 4x4" value of cell 1,1 to s &
		send "jet 4x4" value of cell 3,1 to e
85	nop	;d=81
86	nop	;d=82
87	nop	;d=83
88	nop	;d=84
89	nop	;d=85
90	nop	;d=86
91	nop	;d=87
92	nop	;d=88
93	nop	;d=89
94	nop	;d=90
95	nop	;d=91
96	w = s, s = e	;d=92 send "jet 4x4" value of cell 3,4 to w &
		send "jet 4x4" value of cell 1,3 to s
97	e = n, n = w	;d=93 send "jet 4x4" value of cell 1,3 to e &
		send "jet 4x4" value of cell 1,3 to n
98	nop	;d=94
99	nop	;d=95
100	nop	;d=96

101	nop	;d=97
102	nop	;d=98
103	nop	;d=99
104	nop	;d=100
105	nop	;d=101
106	nop	;d=102
107	nop	;d=103
108	nop	;d=104
109	nop	;d=105
110	alu = e + n	;d=106 receive "jet 4x4" value of cell 4,4
111	alu = alu + w	;d=107 receive "jet 4x4" value of cell 0,4
112	alu = alu + s	;d=108 receive "jet 4x4" value of cell 0,0

Between time t = 53 and t = 108 all four 4 × 4 matrix sums are routed to the center cell of the 8 × 8 matrix. After all sums are received and the 8 × 8 sum is calculated, the 8 × 8 sum is compared with the threshold. If the value is greater than the threshold, it is assumed to be a jet and the trigger tower id and energy are sent to the North.

12.4 Result of Analysis on the 4 × 4 and 8 × 8 Jet Finding in 1-cell Per Chip Assembly

The total time required in all the arrays (considering also the dependency of data that must be exchanged between processors) in "T"cycles (1 cycle = 8 ns in the present DataWave version and 4 ns in the future version), is shown in Table 19.

TABLE 19. TOTAL 4 × 4 AND 8 × 8 MATRIX ALGORITHM EXECUTION TIME FOR JET FINDING ON DATAWAVE.

	4×4	8×8
Number of lines	73	129
Finish routing "em" and "had" values (in clock cycles)	56	112
Send tower id and energy (in clock cycles)	76	132

The timing of these algorithms can be considerably shortened by decreasing the amount of time it takes to send and receive from different chips. For the routing of the 4 × 4 sums of the 8 × 8 jet-finding algorithm (Table 18), 70% of the lines contain "nops". If the "nops" could be deleted the time of the algorithm would dramatically decrease.

For the information of the transverse and total energy, all the partial energies from the 8 × 8 trigger towers should be sent to an external logic unit. In the case of the GEM experiment, the unit will total 20 times the 8 × 8 partial sums, while in the case of the SDC experiment, the unit will total 56 times the 8 × 8 partial sums.

13.0 "EM" CLUSTER FINDING (ISOLATION) AND JET FINDING

13.1 DataWave Assembler Code and Detailed Timing Description

The purpose of this algorithm is to show how different algorithms can be combined without the total time being the sum of the individual algorithms' sum, but only a fraction of it. In this study the two "em" sums + front-to-back + electron isolation + 4 × 4 jet finding have been compiled together. The flow of the resulting algorithm is shown in Figure 28.

As one can see, if a cell does not qualify as an electron in phase 1, the cell does not execute the code for the ratio or isolation, but only executes the jet-finding algorithm. However, if the cell passes the "em" threshold and front-to-back tests, the electron isolation test and the 4 × 4 jet finding algorithms are executed in parallel.
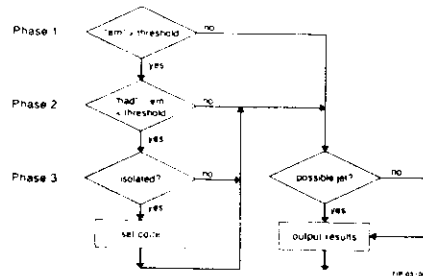

Figure 28.   Flow Diagram Of The "Em" Cluster and Jet Finding on DataWave.

13.2 Result of Analysis on "Em" Cluster Finding (Isolation) and Jet Finding

Due to the length of the code and due to the fact that it is a combination of the ones described in the previous sections without repetition of common code, the code for this algorithm is not listed. However, the results of the execution time are shown in Table 20.

TABLE 20.   TOTAL DATAWAVE ALGORITHM EXECUTION TIME FOR TWO "EM" SUMS + FRONT-TO-BACK + ISOLATION + 4 × 4 JET FINDING.

	Minimum Time (in cycles)	Maximum Time (in cycles)
lines of code	154	
Finish time for decision to dismissing cell as a possible electron or jet —send out null values**	91	114
Finish time for decision of possible electron —sends out the tower id + "em" value**	112	114
Finish time for decision of possible 4 × 4 jet —send out the tower id + (4 × 4) energy sum	94	114
Finish time for decision of possible electron and possible jet —send out the tower id + (4 × 4) energy —sum	112	116

** Timings given are the time that the last value is sent from the cell

Given that 1 clock cycle = 4 ns in the future version of the DataWave chip, the longest time to make a decision using these algorithms would be 464 ns. Although this timing itself is not acceptable for the Level 2 Trigger, with a few optimizations to the DataWave chip, the timing becomes feasible, or it can be useful as a preprocessor of the Level 2 trigger.

14.0   PROGRAMMABLE LEVEL 1 TRIGGER SUSTAINING 16 NS RATE

14.1   Suggested Modifications of the DataWave to Front-end-processor (FEP)

From the experience of using the DataWave and from the specific requirements of the Level 1 trigger algorithms, we can suggest an architecture that will solve the problem of having a fully pipelined programmable Level 1 trigger sustaining the rate of 16 ns more efficiently.

The 16 ns rate cannot be achieved by a single processor cell executing at a clock speed of 4 ns per clock cycle; however, a pipeline of 4 or more processors can allow each cell to execute an algorithm of 64 ns or more before outputting its result. It is possible to make use of the existing DataWave with the pipeline stages described in the conclusion (Table 30 and in Section 10.5), but the timing for routing data between cells and each cell's internal pipeline make the existing DataWave cell less efficient. With modifications to the DataWave, the DataWave will not only become more efficient, but also become a suitable choice for the Level 1 trigger.

Figure 29 shows the suggested modifications of the DataWave to the Front-end Processor (FEP) for the Level 1 trigger algorithms.

Figure 29.   Front-End-Processor (FEP) Cell Architecture.
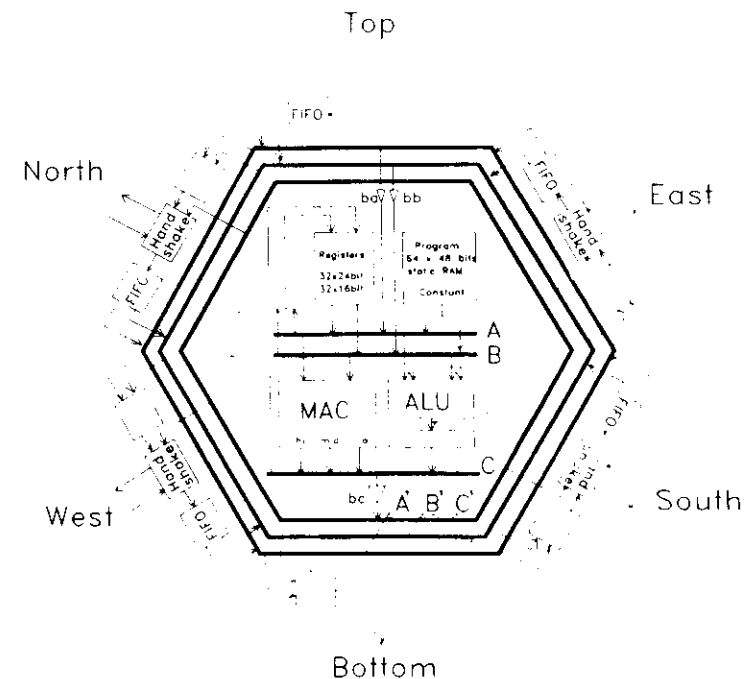
Two new ports are added to the existing DataWave processor, one for the top and one for the bottom (in addition to the North, East, South, and West ports). These ports allow for easy data flow between different stages (see Figure 30), eliminating the routing of data to the different pipelined stages through more expensive and less reliable connectors and multiplexers.
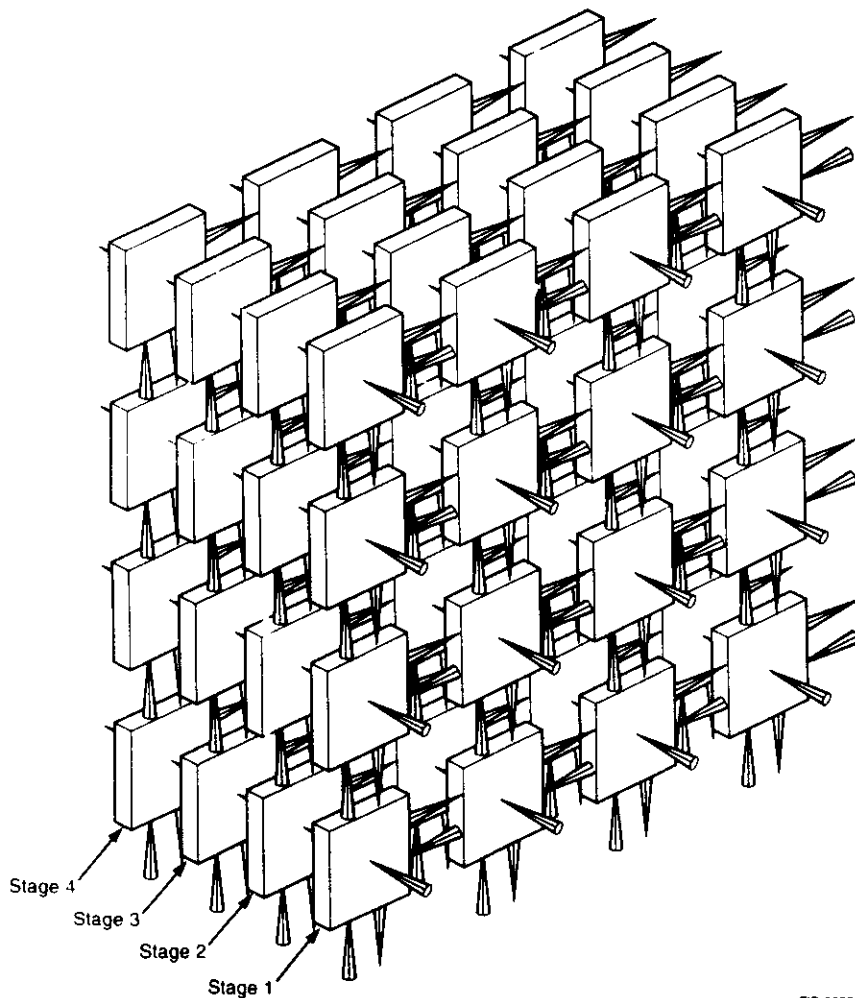
Stage 4
Stage 3
Stage 2
Stage 1

TIP-03059

Figure 30. General Scheme of the Pipelined Parallel Processing Architecture using the FEP.

Since the algorithms only use a small fraction (22%) of the DataWave instruction set, the modified version will simplify the DataWave instruction set by only allowing for instructions that are foreseen to be used in the triggering algorithms (see Table 21). This not only makes the FEP simpler than the DataWave, but also more economical by dropping 78% of the instructions.

52

TABLE 21. FEP INSTRUCTION SET SUITABLE FOR TRIGGER ALGORITHMS.

nop	no operation	alu = A – B	
hi = B	load hi from B	bra	branch
lo = B	load lo from B	bmmi	acc < 0
mid = B	load mid from B	bmpl	acc > = 0
^acc = + A * B		beq	alu = 0
^acc = acc + B		bne	alu <> 0
^acc = acc – B		bmi	alu < 0
^acc = acc + A * B		bpl	alu > = 0
^alu = A + B			

^ NOTE: a port can be used in place f ACC, ALU, A, B except in when the ACC or ALU is used twice in the same instruction, which then it can only be used as in output and not as an input

In addition to reducing the instruction set, removing the pipeline of instructions (resulting in output that can be used after 1 clock cycle) and removing the delay of data between processors (data sent from one cell can be received after 1 clock c cle) will eliminate all nops from the algorithms. A modification in the number and size of registers is shown in Figure 29. It is foreseen that it might be necessary to receive more values from the calorimeter trigger tower, or that it might be necessary to have more calibration constants (for different E_T, E_{TOT}, E_x, L_y, etc.) or different thresholds. Therefore, the number of registers should be increased as well as the size of the registers (at present 12-bit) to fulfill the precision requirement of the Level 1 trigger.

Due to the frequency of use of the 6 ports, a buffer at the receive unit for each port is needed that will allow data that is received from a port to be sent to both an internal unit (ALU, register, etc.) and on the same internal bus be sent to another port. At present these operations require two different buses, and with the new ports the buses become overloaded.

14.2 Differences on the Real-time Algorithm and Data Loading with Respect to the Earlier Algorithms

Since the new assembler instruction of the FEP in the pipelined stages implementation is different from the original DataWave instruction set, the programmer is not limited by extra cycles between the time a value is received, or between the time an ALU/MAC instruction is executed and the time its flags are set. Also, since the "branch" instructions will branch immediately, the three instructions following a "branch" statement are no longer executed.

Due to the staged architecture design, the new algorithm must include pipelining data through the different stages of the processors (see Figures 31 and 32).

At each input port of the FEP processor (as it is also on the present DataWave design) there is a FIFO that is derandomizing the data from the calorimeter to the processor array. This will allow the calorimeter to send two data ("em" and "had") every 16 ns, and the processor fetching the values whenever the program executes the fetch instructions (at 4 ns clock cycles). The program execution at stage 1 must not only route the new incoming data from the calorimeter (one "em" and "had" value every 16 ns) to the next stage in the pipeline staging (stage 2), but must also execute its trigger algorithm in parallel. All processors must likewise pipeline data. When the stage 1 processor has finished its algorithm, it then sends its results to the stage 2 processor, which passes it on. At this point the stage 1 processor begins to re-execute its algorithm: receiving the "em" and "had" values from the calorimeter and processing those values.

The output results from all processors flow (like the input data) through the different processor stages. The last processor will output the results from all processors at a rate of 16 ns.

53

One optical fiber
(see table 2) per
each used input
(.4Δφ x .4 Δη) total
calorimeter area
per board

Stage 4 Stage 3 Stage 2 Stage 1

o
o
o
o
o

(1Gbit/fiber)

Digital pipeline stages
(layers)

Digital filter or digital sums for
trigger tower segmentation
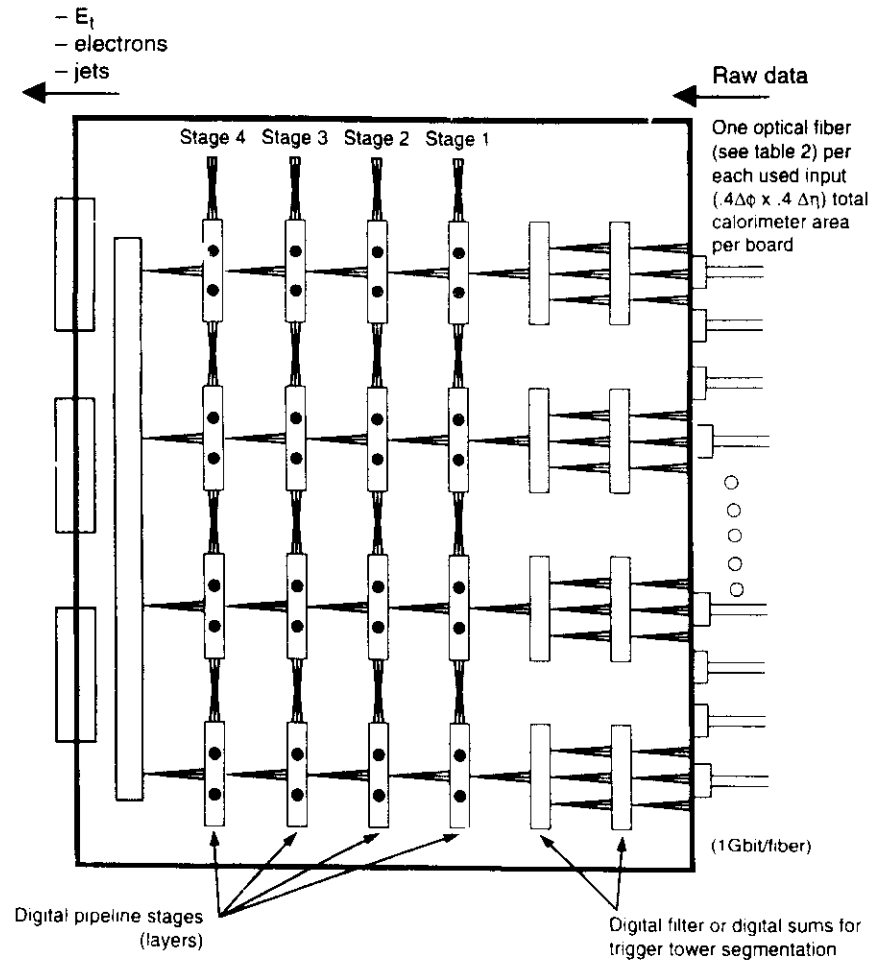
TIP-03061

Figure 31.   One Board of the Programmable Level 1 Trigger with FEP Pipelined Array.

14.2.1   Assembler Code of the FEP (Modified DataWave) for the Section 10.0 Algorithm

Table 22 shows the FEP assembler code for the two "em" sum + front-to-back algorithm (previously described using the DataWave assembler code in Sub-section 10.1) loaded with the same algorithm for finding electrons, but the routing between different stages (the top-to-bottom instruction) depends on its stage position. Assembler code for the routing between stages is shown for all stages needed for this algorithm. A graphical representation of the input and output data routing between the stages, the algorithm execution time at each stage, the latency between input data and output results and the data flow in the pipelined architecture, is shown in the timing diagram of Figure 32.

All pipelining is explained in parenthesis after the "b=t" instruction. The number is the processor stage number where the data that is being pipelined will be processed or where the outputted data was originally sent from. The codes are as follows:

a   inputted "em" value
b   inputted "had" value
c   outputted tower id
d   outputted "em" sum (either 1×2 or 2×1)

Figure 32.   Timing Diagram of Four FEP Stages of a Pipelined Programmable Level 1 Trigger.

TABLE 22. NEW FEP ASSEMBLER CODE OF THE FOUR PIPELINED STAGE ALGORITHMS OF SECTION 10.0.

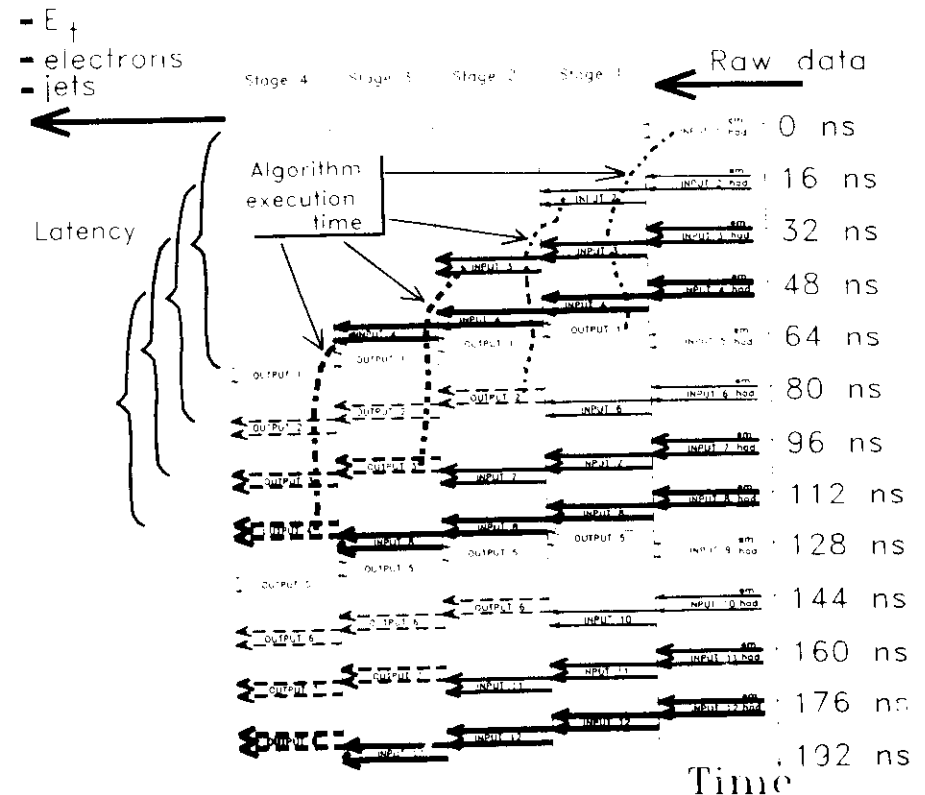|	.cell 0,1,2	STAGE 4	STAGE 3	STAGE 2	STAGE 1	
1	loop: r1 = s = w = t * r14	(4a)	(3a)	(2a)	(1a)	; receive "em" value from calorimeter
2	r2 = s = w = t * r15	(4b)	(3b)	(2b)	(1b)	; receive "had" value from calorimeter
3	r7 = r1 + n,	b = t (1c)				; north 1 x 2 "em" sum
4	r8 = r2 + n,	b = t (1d)				; north 1 x 2 "had" sum
5	r9 = r1 + e	b = t (4a)	b = t (3a)	b = t (2a)		; east 2 x 1 "em" sum
6	r10 = r2 + e	b = t (4b)	b = t (3b)	b = t (2b)		; east 2 x 1 "had" sum
7	alu = r7 – r5,	b = t (2c)	b = t (1c)			; compare 1 x 2 "em" sum to Threshold
8	bpl north,	b = t (2d)	b = t (1d)			
9	alu = r9 – r5,			b = t (4a)	b = t (3a)	; compare 2 x 1 "em" sum to Threshold
10	bpl east	.		b = t (4b)	b = t (3b)	
11	bra nosend,	b = t (3c)	b = t (2c)	b = t (1c)		
12	north acc = r7 * r5	.		b = t (4a)	b = t (3a)	; "em" * Threshold (1x2)
13	acc = acc – r8,			b = t (4b)	b = t (3b)	; "em" * Threshold – "had"
14	bmpl sendn,	b = t (3c)	b = t (2c)	b = t (1c)		
15	bra nosend2,	b = t (3d)	b = t (2d)	b = t (1d)		
16	east acc = r9 * r5,	b = t (3c)	b = t (2c)	b = t (1c)		; "em" * Threshold (2 x 1)
17	acc = acc – r10,	b = t (3d)	b = t (2d)	b = t (1d)		; "em" * Threshold – "had"
18	bmpl sende	.			b = t (4a)	
19	bra nosend3,				b = t (4b)	
20	sendn nop,	b = t (3d)	b = t (2d)	b = t (1d)		
21	nop.				b = t (4a)	
22	nop.				b = t (4b)	
23	b=23	(4c)	(3c)	(2c)	(1c)	send out tower id
24	b=r7, bra loop	(4d)	(3d)	(2d)	(1d)	; send out 1 x 2 "em" energy
25	sende nop,				b = t (b)	
26	b = 23	(4c)	(3c)	(2c)	(1c)	; send out tower id
27	b = r9, bra loop	(4d)	(3d)	(2d)	(1d)	; send out 2 x 1 "em" energy
28	nosend1 nop,				b = t (1d)	
29	nosend2 nop	.			b = t (4)	
30	nop,				b = t	
31	nosend3 b = 0	(4c)	(3c)	(2c)	(1c)	; send out null value
32	b = 0, bra loop	(4d)	(3d)	(2d)	(1d)	; send out null value
.end						

The tower id + "em" energy is sent out at t = 31 and 32 (56 – 60 ns after the processor fetches the data).

14.2.2 New FEP Assembler Code to Realize Trigger Tower Segmentation

In the case that 4 "em" and 2 "had" values must be sent into the FEP pipeline stages, FEP chips that will sum these values into an "em" and "had" sum must be added in front of the processor pipeline stages (see Figure 31). The code for this chip is shown in Table 23. This code assumes that digitized values from the calorimeter have been corrected (linearization, pedestal subtraction and calibration constants in external look-up table).

TABLE 23. NEW FEP ASSEMBLER CODE FOR REALIZING TRIGGER TOWER SEGMENTATION IN ONE STAGE.

1	r1 = w + n		; fetch two "em" values & sum
2	r2 = w + n,	acc = r1	; fetch "em" values & sum, store first sum in acc
3	r3 = w + n,	b = acc + r2	; fetch "had" values & sum, add two "em" sums ; and send "em" result to the first processor stage
4	b = r3		; send "had" result to the first processor stage

Since the values are multiplied by a calibration constant in the processor pipeline, there is no need to do that in this chip.

In the case it is desired to realize a flexible trigger tower segmentation with each received digital value from the calorimeter corrected by pedestal subtraction, the code for the processor cells in the two stages is shown in Table 24 and Table 25.

TABLE 24. NEW FEP ASSEMBLER CODE FOR THE CELL OF THE FIRST STAGE OF THE TOWER SEGMENTATION.

1		alu = t * r10	; "em1" + ped (em)
2		alu = alu + n	; "em2"
3	loop:	alu = alu + e	; "em3"
4		b = alu + t	; "em4", send to "b" the result of "em" sum
5		alu = n + r11, s=t	; "had1" + ped (had) {"em1"}
6		b = alu + e, w = n	; "had2" {"em2"}, send to "b" the result of "had" sum
7		s = e	; {"em3"}
8		w = t	; {"em4"}
9		s = n, alu = t + r10	; {had1}
10		w = e, alu = alu + n, bra loop	; {had2}

TABLE 25. NEW FEP ASSEMBLER CODE FOR THE CELL OF THE SECOND STAGE OF THE TOWER SEGMENTATION.

1	loop:	b = t	; send to "b" the result of "em" sum from 1st stage
2		alu = n + r10	; "em1" + ped (em)
3		alu = alu + e, b = t	; "em2", send to "b" the res. of "had" sum from 1st stage
4		alu = alu + n	; "em3"
5		b = alu + e	; "em4", send to "b" the res. of "em" sum from 2nd stage
6		alu = n + r11	; "had1"
7		b = alu + e, bra loop	; "had2", send to "b" the res of "had" sum from 2nd stage

14.2.3 New FEP Assembler Code of a Digital Filter Applied to Calorimeter Signals

To sustain the 16 ns rate, the digital filter must be comprised of two FEP processors (see Section 7 for more information on digital filters). The code for both stages are listed in Tables 26 and 27.

TABLE 26. NEW FEP ASSEMBLER CODE FOR THE CELL OF THE FIRST STAGE OF THE DIGITAL FILTER.

1		acc = t * r11	fetch a value and multiply it by a coefficient —> acc
2	loop:	acc = acc + n * r12	; add a value * a coefficient to the acc
3		acc = acc + e * r13	
4		acc = acc + t * r14	
5		b = acc + n * r15, s = t	; send the result of the filter south, send the first value ; received from the next group to the bottom ;. NOTE: both south and bottom are connected to the ; second chip of the digital filter
6		w = n	; routing all of the next group of values to bottom
7		s = e	
8		w = t	
9		acc = t * r11, s = n, bra loop	; begin filtering values in this cell, send the last value ; of the last group to bottom, and repeat the filtering ; algorithm

TABLE 27. NEW FEP ASSEMBLER CODE FOR THE CELL OF THE SECOND STAGE OF THE DIGITAL FILTER.

1	loop:	acc = n * r11, b = t	; fetch a value from the first chip and multiply it by a ; coefficient —> acc, send the result of the first chip to ; the first stage of the processor pipeline
2		acc = acc + e * r12	; add a value * a coefficient to the acc
3		acc = acc + n * r13	
4		acc = acc + e * r14	
5		b = acc + n * r15, bra loop	; send the result of the filter south, and branch to the ; beginning of the program to wait for data from the top

The first chip received the tower's five inputs from the calorimeter into ports: Top, North and East. After the values are filtered through the algorithm, the result is sent to the next stage through the Bottom port.

at this time the chip needs to fetch another set of data. Since the first chip can not process this set of values until the next clock cycle, the values are passed to the second chip, which filters the values and sends its result and the result of the first chip through its Bottom port to the first stage of processors.

As one can see from Table 27, an output from the second chip occurs every four clock cycles (1 clock cycle = 4 ns) and thus sustains the 16 ns rate.

14.2.4 New FEP Assembler Code of the Two "Em" Sum + Front–to–back + Jet Finding Algorithm

This code uses the code in Table 22 (Sub–section 14.2.1) for the two "em" sum + front-to-back, but changes the algorithm for isolation and jet–finding from the one described in Section 13. The flow of the program is modified from the previous algorithm and is shown in Figure 33.
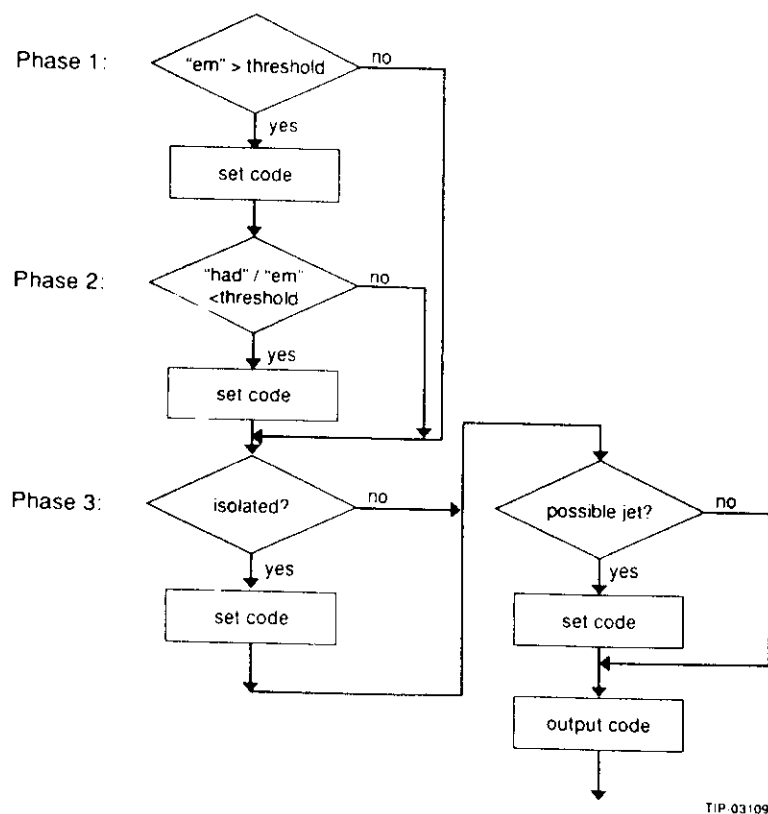

Phase 1:
Phase 2:
Phase 3:

TIP-03109

Figure 33. Flow Chart of the Two "Em" Sum + Front-to-back + Isolation + Jet Finding (FEP Pipelinable Version).

The earlier code was limited by cells waiting for input from their neighboring cells for both the isolation and jet-finding algorithms. Because of this wait, it was more efficient to send data one-by-one to each cell, o..ly adding the values that needed to be added (all values except the 2×2 "em" values). Because of this flow of data, the algorithm did not have to calculate the 2×2 "em" sum and subtract it from 4×4 sum; once the cell added the last data set, the sum was already the $(4 \times 4) - (2 \times 2$ "em").

Since this limitation does not apply to the FEP processor, a new algorithm (see Figures 34 and 35) was developed to take advantage of the FEP speed.

Each cell begins the isolation algorithm by first summing the "em" and "had" in it own tower and sends the resulting sum to the South (t = 13). On the next cycle each cell receives the energy sum of its northern cell and calculates the 1×2 sum, which it sends East. At time t = 15, each cell receives the 1×2 sum from the East and adds its own 1×2 sum creating the 2×2 sum for that cell. After the 2×2 sums are calculated, the sums are sent to the middle cell, which adds them together to form the 4×4 sum, see Figure 34.

The cell now needs to subtract the 2×2 "em" sum. It begins by sending its own "em" value South (see Figure 35). After a cycle it receives from the North, its northern cell's "em" value, which it adds to its own to form a 1×2 "em" value and sends it East. At the next cycle the cell receives the 1×2 "em" sum from the West and adds it to its own, creating the 2×2 "em" sum, which is then subtracted from the 4×4 sum. At time t = 24, the $(4 \times 4) - (2 \times 2$ "em") is compared with the threshold.

Since each cell already has the 4×4 value, for the jet algorithm it only needs to compare the 4×4 sum with the threshold and test the result.

The new code for the two "em" sum + front-to-back + isolation + jet finding is listed in Table 29. Since the code must check for all these criteria, at the end of its algorithm, each cell outputs a code that has encoded the result of its test. The output codes are listed in Table 28.

TABLE 28. OUTPUT CODES FOR TWO "EM" SUM + FRONT-TO-BACK + ISOLATION + JET-FINDING ALGORITHM ON FEP.

1	two "em" sum (north 1×2) > threshold
2	two "em" sum (east 2×1) > threshold
4	"had"/"em" (north 1×2) < threshold
8	"had"/"em" (east 2×1) < threshold
16	isolation achieved
32	possible jet found

Combinations of these codes are allowed. For example, a cell may return a code of 37 (1 + 4 + 32) stating that the possible electron was found, but it was not isolated from surrounding energy and that the cell may be part of a 4×4 jet.

Each cell also outputs the 4×4 sum which will be used to calculate the E_t (78 values should be added externally in the case of GEM calorimeter and 224 values in the case of SDC experiment)

The assembly code for finding E_t, electrons, isolation and jets is shown in Table 29. Due to lack of space, only the code for the stage 1 processor is shown. The numbers on the left of the algorithm are the instruction line numbers, while the right-most number is the clock cycle the instruction is executed (assuming the first instruction is executed at time t = 0).

All lines that refer to the outputted codes (defined in Table 28) are marked with a "^". All stage pipelining code is explained to the right of the comments in parenthesis (using the symbols as explained above, in Table 22, Sub–section 14.2.1).

Figure 34.   Routing 4 × 4 Sum for Electron Isolation and 4 × 4 Jet Finding (FEP).

Figure 35.   Routing 2 × 2 "Em" for Electron Isolation and 4 × 4 Jet Finding (FEP).

TABLE 29.   NEW FEP ASSEMBLER CODE OF THE PIPELINED ALGORITHM TO FIND E_T, ELECTRONS, ISOLATION AND JETS.

```
         .cell 0,1,2
1    loop:  r1 = s = w = t * r14        ; receive "em" value from calorimeter        0
2           r2 = s = w = t * r15        ; receive "had" value from calorimeter       1
3           r7 = r1 + n                 ; north 1 x 2 "em" sum                        2
4           r8 = r2 + n                 ; north 1 x 2 "had" sum                       3
5           r9 = r1 + e,        b = t   ; north 2 x 1 "em" sum          (2a)          4
6           r10 = r2 + e,       b = t   ; north 2 x 1 "had" sum         (2b)          5
7           alu = r7 - r5               ; compare 1 x 2 "em" sum to Threshold         6
8           bpl north, alu = r9 - r5    ; compare 2x1 "had" sum to Threshold          7
9           bpl east            .b = t                                 (3a)          8
10          bra nosend1, r0 = 0,  b = t                                (3b)          9
11   north: acc = r7 * r5, r0 = 1  b = t  ; ^"em" * Threshold (1 x 2)  (3a)          8
12          acc = acc - r8       b = t   ; "em" * Threshold - "had"    (3b)          9
13          bmpl sendn                                                              10
14          bra nosend2                                                             11
15   east:  acc = r9 * r5, r0 = 2  b = t  ; ^"em" * Threshold (2 x 1)  (3b)          9
16          acc = acc - r10              ; "em" * Threshold - "had"                 10
17          bmpl sende                                                             11
18          acc = r1, b  iso,    b = t   ; set acc="em" for iso algrthm (4a)        12
                                        ; ^                                         11
19   sendn: r0 = r0 + 4                                                            12
20          acc = r1, bra iso,   b = t  ; set acc="em" for iso algrthm  (4a)        12
21      sende: r0 = r0 + 8, acc = r1, bra iso,  b = t  ; ^set acc="em" for iso algrthm (4a)  12
22   nosend1: nop                                                                  10
23          nop                                                                    11
24   nosend2: r0 = 0, acc = r1,   b = t  ; set acc="em" fos iso algrthm (4a)        12
25   iso:   s = acc = acc + r2    b = t  ; add "had" and send s          (4b)       13
26          w = acc + n                 ; add n tower for 1 x 2 sum, send w         14
27          n = s = acc + e             ; add e 1 x 2 for 2 x 2 sum, send n         15
28          w = e = s            b = t  ; routing 2 x 2 sums             (5a)       16
29          w = e = n,           b = t  ; routing 2 x 2s, store nw 2 x 2 sum (5b)   17
30          alu = e + w                 ; add sw 2 x 2 sum to acc                   18
31          alu = alu + w               ; add ne 2 x 2 sum to acc                   19
32          r5 = alu = alu + e, s =acc = r1,  b = t  ; add nw 2 x 2 to acc, "em"-->alu  (6a)  20
33          s = acc = acc + n    b = t  ; add n "em" -> alu              (6b)       21
34          r6 = acc + e                ; add e 1 x 2 "em" -> alu                   22
35          alu = r5 - r6               ; (4 x 4) - (2 x 2)                         23
36          alu = alu - r3       b = t  ; (4 x 4) - (2 x 2) - Threshold  (7a)       24
37          bmi sendiso          b = t  ;                                (7b)       25
38          bra cont                                                               26
39   sendiso: r0 = r0 + 16                                            ; ^           26
40   cont:  alu = r5 - r4                ; compare 4 x 4 with jet value             27
41          bpl jet,             b = t  ;                                (8a)       28
42          bra send,            b = t  ;                                (8b)       29
43   jet:   r0 = r0 + 32         b = t  ; ^                             (8b)        29
44   send:  b = r                        ; ^send out code                          30
45          b = r5                       ; send out 4 x 4 energy value             31
```

The result of the algorithm is a fully programmable 8-processor-stage design for the Level 1 trigger which identifies possible electrons and jets as well as outputs 4 × 4 values for calculating the E_T.

15.0 CONCLUSIONS

The simulation of trigger algorithms on the DataWave chip has demonstrated that a processor even simpler than the DataWave (implementing only 20% of the instructions, making it both more economical and easier to program) can offer the possibility of a flexible programmable Level 1 trigger (sustaining 16 ns clocking).

The new discovery, as a result of this study, was that the combination of very few instructions, a number of simple algorithms, and specific hardware can meet the needs of the Level 1 trigger. Since modifying the existing DataWave can be shown to allow for all three of these conditions, the most natural way to implement the fully-programmable Level 1 trigger would be a modification of the existing DataWave chip.

TABLE 30. RESULTS OF A FULLY PROGRAMMABLE LEVEL 1 TRIGGER SUSTAINING 16 NS CLOCKING.

Algorithm	PRESENT DATAWAVE		MODIFIED DATAWAVE	
	Algorithm time (in clock cycles)	Number of processor stages	Algorithm time (in clock cycles)	Number of processor stages
Flexible trigger tower segmentation	—	—	4 7	1 2
Filter	10	1-2	6	2
3 × 3 cluster identification (1-cell per chip)	50	13*	22	6*
3 × 3 cluster identification (16-cells per chip)	72	18*	26	7*
"em" < threshold + front-to-back	43	11	15	4
electron isolation	65	17*	17	5*
jet-finding (4 × 4)	65	17*	14	4*
jet-finding (8 × 8)	120	30*	20	8*
E_T, E_{TOT}, E_X, E_Y	55	16*	11	3*
"em" < threshold + front-to-back + isolation + jet-finding (4 × 4)	116	29*	31	8

*Note Estimated number of stages

With a FEP processor running at 250 MHz, an algorithm for two "em" sum + front-to-back could be implemented in 4 stages (sustaining the rate of 16 ns) resulting in a total of 5000 processors for the GEM experiment and 14,336 for the SDC (for the "em" + front-to-back + isolation + jet-finding algorithm the number of processors will double). The design of this processor is not more expensive than a standard ASIC; thus this solution is not only flexible, but can be affordable.

The flexibility of this solution can be demonstrated by the ease of programming on a DataWave or FEP cell. Any physicist can change the algorithms of the FEP by coding a simple program, consisting of less than 64 operations and using an instruction set of 17 instructions. Due to this simplified instruction set, the effort to learn to program the FEP is minimal.

Experience shows that trigger algorithm tuning usually begins after acquiring a few full events. The possibility of a flexible, programmable system at an affordable cost (compared with cabled logic), makes exploring this solution not only to be beneficial to the GEM and SDC experiments, but also to other experiments as well.

REFERENCES

1. D. Crosetto, "A Fast Cluster Finding System for Future HEP Experiments," *Nuclear Instruments and Methods in Physics Research*, **A311**, (1992), 49–56.

2. N. Bains *et al.*, "The UA1 Upgrade Calorimeter Trigger Processor," *Nuclear Instrument and Methods in Physics Research*, **A292** (1990) 401–423.

3. N. Ellis, "Eagle Trigger/DAQ/FE Group," CERN DAQ-TR-109 20/2/92.

4. B. Aubert *et al.*, "Liquid Argon Calorimeter with LHC-Performance Specifications, CERN/DRDC/90-31.

5. Solenoidal Detector Collaboration. Technical Design Report, SDC-92-201, 1 April 1992.

6. D. Marlow, Private Communications.

7. Gamma (photons), Electron and Muon Letter of Intent. SSCL-SR-1184. GEM TN-92-49. 30 November 1991.

8. G. Jarlskog and D. Rein. "Large Hadron Collider Workshop," CERN 90-10, ECFA 90-133. *Vol. I, II, III* Aachen, 4–9 October 1990.

9. W. H. Smith *et al.*, "Isolated Electron Pattern Logic Design and Performance at SSC." Solenoidal Detector Notes, SDC-91-00087. November 11, 1991.

10. A. J. Lankford, "Issues for Trigger Processing at High Luminosity Colliders," ECFA Study Week on Instrumentation Technology for High-Luminosity Hadron Colliders, Barcelona 14–21 September 1989.

11. K. Caesar, U. Schmidt, S. Mehrgardt and T. Himmel, *Elektronik Magazine* **12** (June 8, 1990).

12. D. Crosetto, "Level 1 and 2 Trigger Architecture, Data Acquisition/Compaction System," North-Holland Physics Publishing. *Proceedings of the 3rd International Conference on Advanced Technology and Particle Physics*. Como, Italy, 22–26 June, 1992. NUPH-B. Ed. E. Borchi *et al.*

Superconducting Super Collider Laboratory is an equal opportunity employer