FERMILAB-CONF-16-690-PPD

VIPRAM_L1CMS: a 2-Tier 3D Architecture for Pattern Recognition for Track Finding

J. R. Hoff, G.W. Deptuch, *Senior Member, IEEE*, S. Joshi, T. Liu, J. Olsen, and A. Shenai

***Abstract*– In HEP tracking trigger applications, flagging an individual detector hit is not important. Rather, the path of a charged particle through many detector layers is what must be found. Moreover, given the increased luminosity projected for future LHC experiments, this type of track finding will be required within the Level 1 Trigger system. This means that future LHC experiments require not just a chip capable of high-speed track finding but also one with a high-speed readout architecture. VIPRAM_L1CMS is 2-Tier Vertically Integrated chip designed to fulfill these requirements. It is a complete pipelined Pattern Recognition Associative Memory (PRAM) architecture including pattern recognition, result sparsification, and readout for Level 1 trigger applications in CMS with 15-bit wide detector addresses and eight detector layers included in the track finding. Pattern recognition is based on classic Content Addressable Memories with a Current Race Scheme to reduce timing complexity and a 4-bit Selective Precharge to minimize power consumption. VIPRAM_L1CMS uses a pipelined set of priority-encoded binary readout structures to sparsify and readout active road flags at frequencies of at least 100MHz. VIPRAM_L1CMS is designed to work directly with the Pulsar2b Architecture.**

I. Introduction

THE use of hardware-based pattern recognition for fast triggering on particle tracks is well established in High-Energy Physics [1][2][3]. The central concept is to use a massively parallel associative memory architecture to identify patterns on an event by event basis with high speed and low latency. Prior to data taking, significant tracks are determined by Monte Carlo simulation as are the detector layers and addresses traversed by these tracks. These detector layers and addresses are stored within the associative memory structures which can rapidly flag the presence of those tracks within event data.

Of critical importance to the future of CMS and its ability to maintain physics acceptances for basic objects (leptons, photons, jets and MET) in the HL-LHC era is the inclusion of tracks within the Level-1 trigger [4]. As such, the development of a readout architecture for hardware-based pattern recognition with the ability to respond with the speed necessary for Level-1 trigger applications is of vital importance.

The choice of vertical integration serves to simplify the readout architecture and minimize the bus lengths necessary to communicate layer data and matched road data around a large chip.

II. Glossary

- Associative Memory – a memory capable of determining if a particular piece of data is contained within itself.
- CAM – Content Addressable Memory – a special type of associative memory that is loaded with a particular pattern and then responds to the presence of that particular pattern within a data stream. Commercial-Off-The-Shelf CAMs are not typically required to remember pattern matches.
- PRAM – Pattern Recognition Associative Memory – a double-layered associative memory structure containing several CAMs and the means to remember matches and monitor the state of matches within the PRAM. Each CAM flags the arrival of one particular pattern which, in Tracking Detectors, corresponds to a particular detector address within a particular detector layer. Once a CAM is flagged, that flag is remembered by the PRAM until reset. When a sufficient (user definable) number of the component CAMs have flagged a match, the PRAM itself raises a flag. This is called a road flag or road match.

III. The Chosen 3D Technology

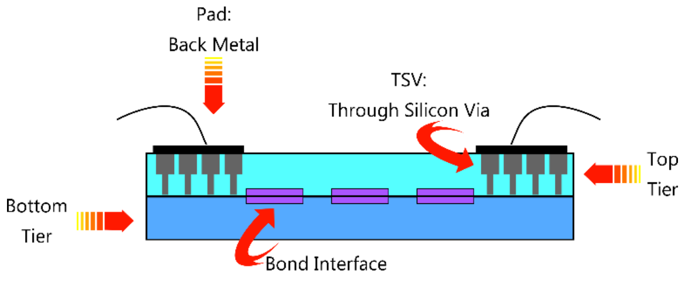


Fig. 1. A sketch of the elements of a two-tier, vertically integrated circuit. Both the Top Tier and the Bottom Tier are common, readily-available CMOS integrated circuits. The two tiers are joined face-to-face by a copper direct bond interface. Through-Silicon Vias (TSVs) join the route metals of the CMOS top tier back through the substrate to back metal. Pads are formed from back metal for off-chip connection.

VIPRAM_L1CMS has been fabricated using Global Foundries' 130nm Low Power CMOS Technology with 3D fabrication through Tezzaron/Novati [5][6]. Fig. 1 shows a

Manuscript received April 3, 2016. (Write the date on which you submitted your paper for review.) This work was supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

J. R. Hoff (telephone: 630-840-2398 email: jimhoff@fnal.gov) , G. W. Deptuch (email: deptuch@fnal.gov), T. Liu (email: thliu@fnal.gov) and A. Shenai (email: shenai@fnal.gov) are with the Fermi National Accelerator Laboratory, Batavia, IL 60510 USA

S. Joshi was an intern with the Fermi National Accelerator Laboratory when this work was performed and is a graduate student with Department of Electrical Engineering and Computer Science, McCormick School of Engineering and Applied Science, Northwestern University, Evanston, IL

simplified sketch of the resulting structure, but this sketch is not to scale.

Each tier is fabricated in a common, readily-available CMOS process. In this particular case, that process is the GF 130nm LPCMOS for both tiers, but this is not required. In fact, it is not even necessary for both tiers to be from the same process. Moreover, the design rules of the process do not have to be modified in any way though there are a few additional rules regarding Through Silicon Vias and Bond Interfaces. This is so that these elements can be added to the chips after 2D processing has been completed.

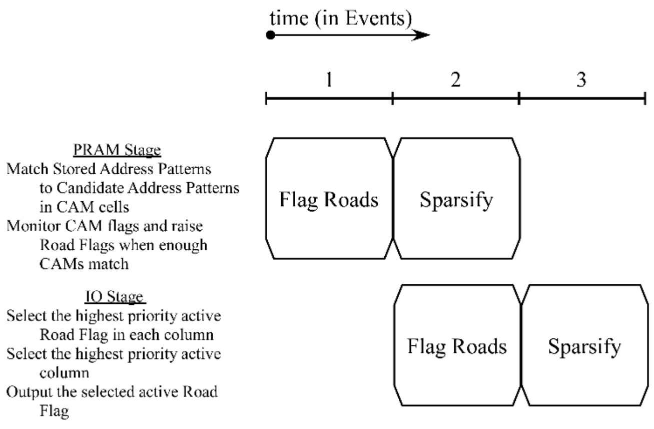
The two tiers are bonded to one another using a low temperature copper Direct-Bond Interface [5]. The metals for the bond interface are placed after 2D integration, so no route layers are lost. The bond interface metals are a 2-dimensional array of 2.5μm wide octagons spaced on a rectangular grid 5μm apart and they serve as both a mechanical adhesion between the tiers and as a means of electrical communication between the tiers. Of course, the vast majority of interface octagons are not used for inter-tier communication and are left electrically inert. The two tiers are bonded face-to-face, meaning that each tier's substrate faces outward of the sandwich created by joining the tiers together.

The two tier stack communicates to the outside world by Through Silicon Vias (TSVs) inserted into the backside of the top tier down to landing pads on the lowest metal layer of the top tier (M1). After the two tiers are bonded into a stack, the top tier's substrate is thinned to approximately 6 microns by chemical-mechanical polishing (CMP). Via cavities are formed by the Bosch process [5] and filled with Tungsten and a spacer layer for substrate isolation.

Finally, Aluminum back metal is placed on the top tier. VIPRAM_L1CMS does no routing on the back metal, so, in fact, all back metal is also pad metal.

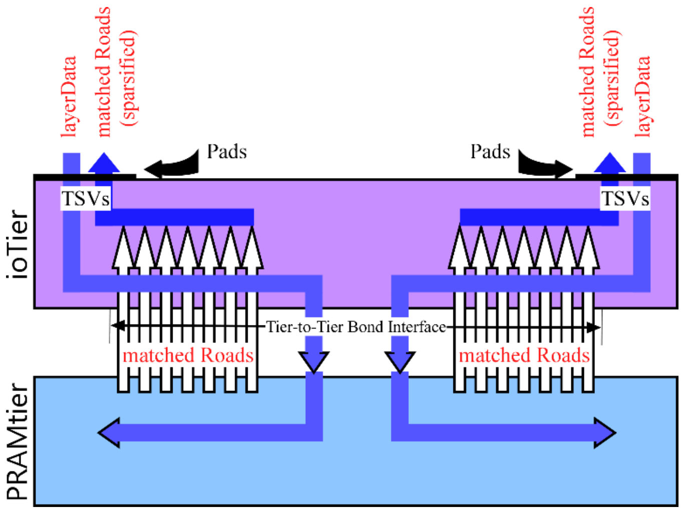
IV. A Two-Tier, Vertically-Integrated Pipeline

At its most basic level, VIPRAM_L1CMS is a very simple, two-stage pipelined device. (See Fig. 2.) The stages of the pipeline are advanced by the arrival of an End-of-Event signal from outside the chip. The first stage of the pipeline is the pattern recognition or PRAM stage. In the PRAM stage,



time (in Events)
1
2
3
PRAM Stage
Match Stored Address Patterns to Candidate Address Patterns in CAM cells
Monitor CAM flags and raise Road Flags when enough CAMs match
IO Stage
Select the highest priority active Road Flag in each column
Select the highest priority active column
Output the selected active Road Flag
Flag Roads
Sparsify
Flag Roads
Sparsify

Fig. 2. Pipeline Diagram for the VIPRAM_L1CMS architecture.



layerData
matched Roads (sparsified)
Pads
Pads
matched Roads (sparsified)
layerData
TSVs
TSVs
ioTier
Tier-to-Tier Bond Interface
matched Roads
matched Roads
PRAMtier

Fig. 3. An abstracted sketch of the two tiers of VIPRAM_L1CMS. Layer hit data arrives to the pads and crosses through Through Silicon Vias (TSVs) to the typical routing metals of the ioTier. From there it is routed in the usual way to the center of the ioTier and passed through the bond interface to the typical routing metals of the PRAM tier. From there it is routed to the PRAMs. Matched roads are passed up from the PRAM Tier to to the ioTier through bond interfaces. Once there, they are sparsified and routed out of the chip through the TSVs and then the pads.

candidate addresses from different detector layers (layerData) are matched against stored patterns in CAM cells within each PRAM. If a sufficient number of CAM cells match, then a matched road is flagged. Over the course of an event, any number of matched roads may fire. The second stage of the pipeline is the readout or IO stage. In the IO stage, matched roads are identified, sparsified and read out.

What makes VIPRAM_L1CMS unique is the simple fact that these two pipeline stages are located on two different tiers of a 3D Vertically Integrated Circuit. (See Fig. 3.) The PRAM stage is placed on the PRAMtier which is the bottom tier of the stack and the IO stage is placed on the ioTier which is the top tier. In keeping with the facts of the 3D process, all pads are located on the ioTier. Therefore layerData arrives to the ioTier and is driven to the center of the chip, across the bond interface to the PRAMtier and then around the PRAMtier to the various PRAM road cells. All road flags are connected directly to bond interface octagons and are driven across to the ioTier. All road flags, whether or not they are successful, are captured into the ioTier with the arrival of the End-of-Event signal. In the ioTier, the successful road flags are sparsified and driven to the pads and out of the chip.

V. The PRAM Road Cell and the PRAM Tier

Each PRAM Road Cell searches for one and only one road which represents a particular path of a charged particle through the various layers of a detector. (See Fig. 4.) The VIPRAM_L1CMS PRAM Cells consists of eight CAM cells each dedicated to searching for one detector address on one detector layer. The first CAM is dedicated to detector layer 0 and receives layerData only from detector layer 0; the second CAM is dedicated to detector layer 1 and receives layerData only from detector layer 1; and so on.

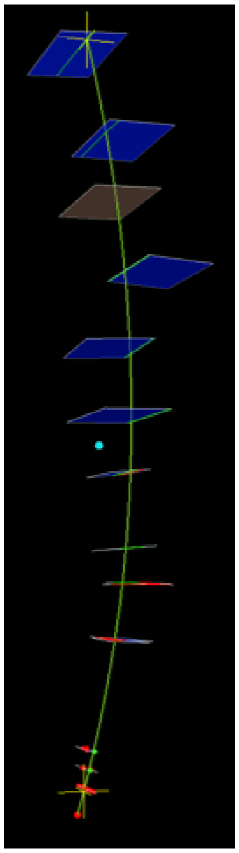


Fig. 4. A simple event display illustrative of the particle path through the various layers of the

In addition to the eight CAM cells, the PRAM Road Cell consists of eight Set-Reset latches (one for each CAM cell) for capturing and remembering matches. Understand that event data arrives to the VIPRAM_L1CMS unordered, so the likelihood that the layerData for a particular track will arrive to the VIPRAM_L1CMS from all of the different detector layers at the same time is vanishingly remote. All of the event data will eventually reach the VIPRAM_L1CMS, so the PRAM Road Cell simply remembers CAM address matches and the final road match itself is accumulated over the arrival time of all of the event data. In addition to the eight CAM cells and the eight latches, the PRAM Road Cell also contains the Majority Logic which monitors status of the stored CAM address matches and flags a road when the number of matches reaches a user defined threshold. The user can request a perfect road match (all eight CAMs match), or 1 missing layer or 2 missing layers. Finally, the PRAM Road Cell contains simple driver circuitry to push the road match across the bond interface to the ioTier.

The CAM cell itself is a classic architecture with a current race scheme and a four bit selective pre-charge [7]. It has fifteen bits, a single matchline per CAM cell, searchlines (called layerData in this paper), and a sense amplifier. The matchline precharge signal (mlpre) when high resets the matchline to zero through the nFET on the right side of Fig. 5 and when low supplies current to the matchline through the pFET on the left side. Bits 0, 1, 2 and 3 in Fig. 5 are NAND Cells [7] which block current flow to the rest of the matchline in the event of a bit mismatch. Bits 5, 6, 7, 9, 10, 11, 13, 14 and 15 are NOR Cells [7] which pull the matchline low in the event of a bit mismatch. Finally, bits 4, 8, and 12 are Ternary NOR Cells [7] which behave like NOR Cells in that they have the ability to pull the matchline low but they also can be placed into a forced mismatch condition (to kill a pattern) or a forced match condition to provide a "don't care" option for the bit.

The layout of the PRAM Road Cell is shown in Fig. 7. The cell is square at 70µm x 70µm. The CAM cells themselves are

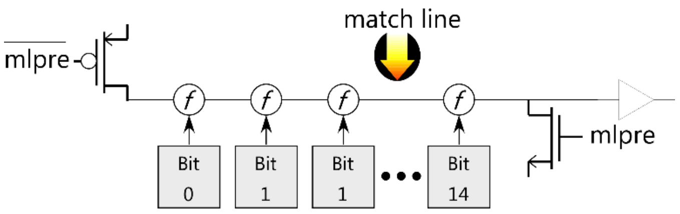


Fig. 5. A functional sketch of the CAM cell architecture. The "f" function is a current block for NAND Cells or a matchline pulldown for NOR and Ternary NOR Cells.

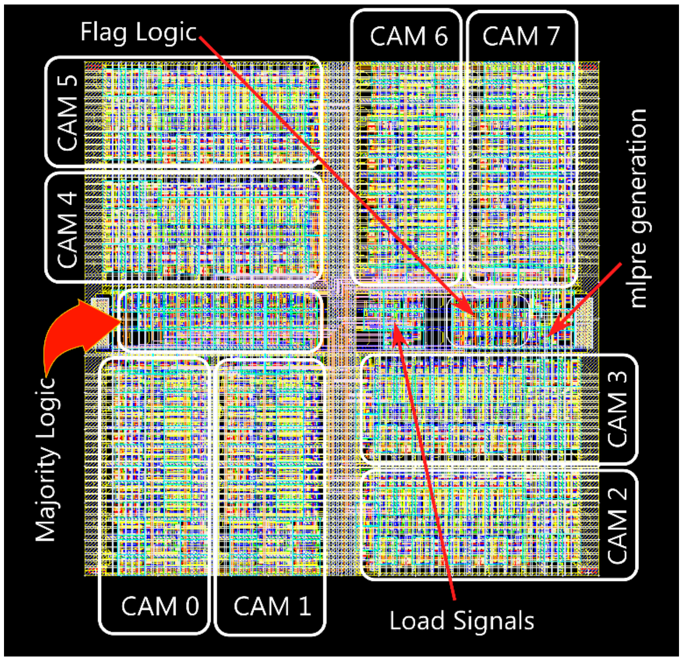


Fig. 7. The layout of the PRAM Road Cell highlighting the locations of the eight CAM cells, the majority logic and the Flag or threshold logic.

twice as tall as they are wide. The layerData for layers 0, 1, 6, and 7 flow horizontally and for layers 2, 3, 4, and 5 flow vertically in the figure. In all cases, the layerData flows into the long side of the CAM cell layout.

The PRAM Road Cells are arranged into a 64x64 array called a Quadrant and then four of these are arranged in a square, resulting in 16k road patterns per VIPRAM_L1CMS chip. (See Fig. 6.) The figure also shows the Quad Driver layout which drives the layerData from the center of the chip outward to all PRAM cells. Routing in the 3rd dimension permits this type of structure in which a long route is dropped strategically in the

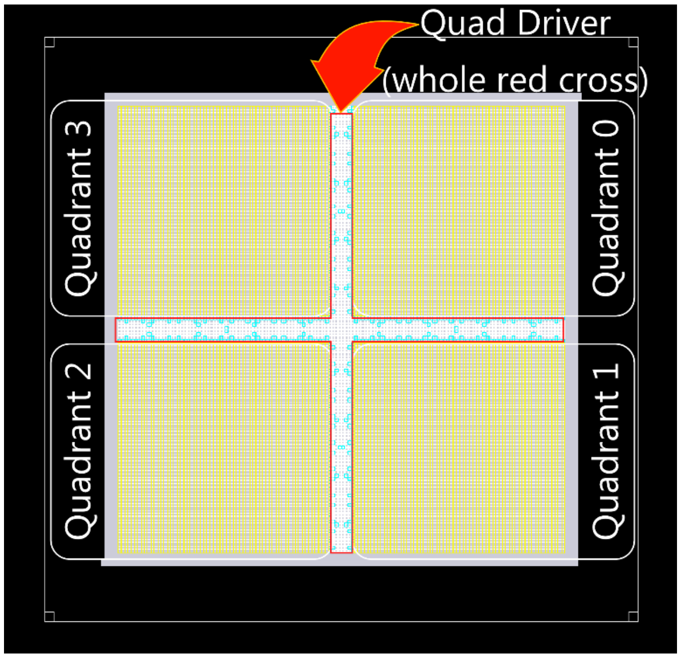
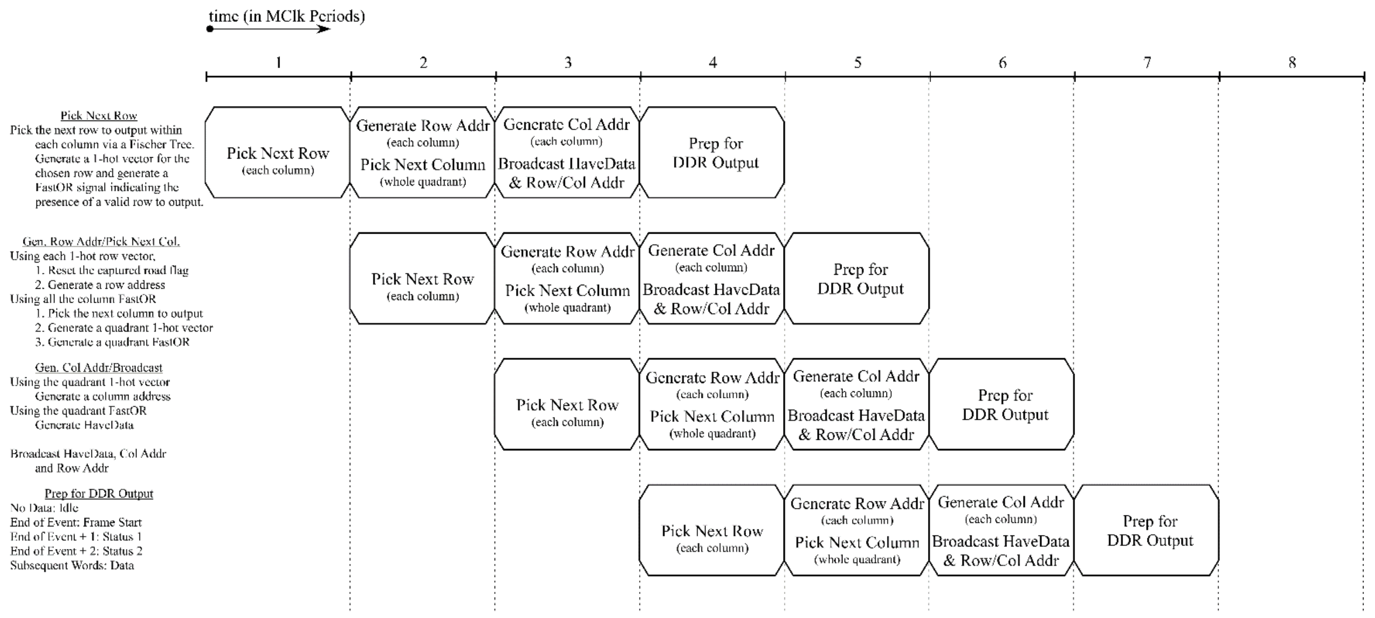


Fig. 6. The layout of the entire PRAM Tier highlighting the four Quadrants and the Quad Driver.


time (in MClk Periods)
1
2
3
4
5
6
7
8
Pick Next Row (each column)
Generate Row Addr (each column)
Pick Next Column (whole quadrant)
Generate Col Addr (each column)
Broadcast HaveData & Row/Col Addr
Prep for DDR Output

Pick Next Row
Pick the next row to output within each column via a Fischer Tree. Generate a 1-hot vector for the chosen row and generate a FastOR signal indicating the presence of a valid row to output.

Gen. Row Addr/Pick Next Col.
Using each 1-hot row vector,
1. Reset the captured road flag
2. Generate a row address
Using all the column FastOR
1. Pick the next column to output
2. Generate a quadrant 1-hot vector
3. Generate a quadrant FastOR

Gen. Col Addr/Broadcast
Using the quadrant 1-hot vector
Generate a column address
Using the quadrant FastOR
Generate HaveData

Broadcast HaveData, Col Addr and Row Addr

Prep for DDR Output
No Data: Idle
End of Event: Frame Start
End of Event + 1: Status 1
End of Event + 2: Status 2
Subsequent Words: Data

Fig. 8. The ioTier pipeline diagram.

center of some area (in this case, it is in the center of the entire Tier) and then routed in all directions. This technique can be used, for example, to minimize overall route lengths or to equalize propagation delays across a wide area. This version of VIPRAM_L1CMS uses a single central drop point with four 64x64 arrays, but other structures are possible – e.g. 4 drop points each sourrounded by four 32x32 arrays.

VI. The ioTier

The ioTier is itself another pipeline, this one advanced on the rising edge of the master clock (MClkA) rather than the End-of-Event signal. The pipeline diagram is shown in Fig. 8. In both the first and second pipeline stages, modified Fischer Trees are used to pick the next row within each column and then the next column within the quadrant. Fischer Trees are also known by their original name, the Mephisto Binary Readout Architectures [8]. Since the operation of the Fischer Tree is important to VIPRAM_L1CMS, it will be discussed first. A discussion of the pipeline and its use of the Fischer Tree will follow.

A. The Fischer Tree (Mephisto Binary Readout Architecture)

A Fischer Tree is a binary tree of Leaf Cells that process data both forward and backward. Fig. 9 (a) shows a simple logic gate implementation of a Leaf Cell. Moving from left to right (forward), if either alertTop OR alertBot are active, then Alert will be activated. Moving from right to left (backward), if Pick is active AND alertTop is active, pickTop is activated but if Pick is active AND alertTop is NOT active, pickBot is activated. Fig. 9 (b) shows the symbol used for Leaf Cells in this paper and Fig. 9 (c) shows an 8-input, 3-stage Fischer Tree. The aggregate function of all these Leaf Cells processing data in the forward direction is a giant OR-gate. If any of the Alerts are active, the FastOR signal will activate in $O(\log_2(N))$ time. This FastOR signal is then passed back through the same tree in the reverse direction. The reverse processing has the effect of following one and only one Alert signal back to its origin by

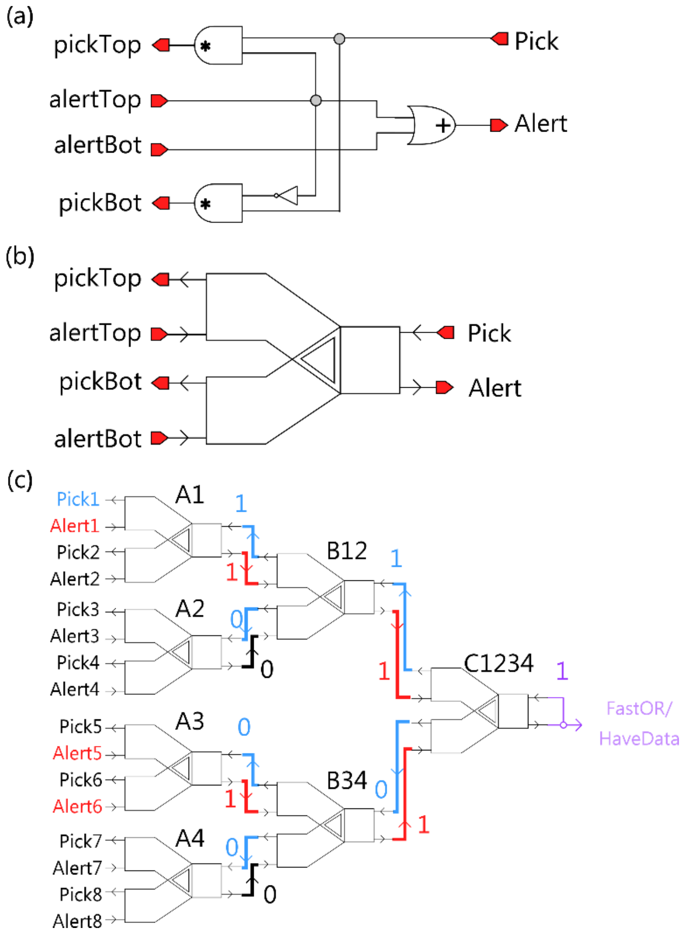


Fig. 9. Three diagrams illustrating the Fischer Tree. (a) Shows the Fischer Leaf Cell with its forward and backward processing. (b) Shows the symbol used in for the Fischer Tree in (c) and (c) An 8-input Fischer Tree showing forward and backward processing to select one of three alerts.

choosing any alertTop - if it is active - over the corresponding alertBot. alertBot signals will be chosen only if they are active and the alertTop is not. This reverse processing is also accomplished in $O(\log_2(N))$ time. In the example illustrated in Fig. 9 (c), Alert1, Alert5 and Alert6 are all active (red). This causes the Alert outputs of Leaf Cells A1, A3, B12, B34 and C1234 to activate. Of course, the Alert output of Leaf Cell C1234 is the FastOR output and it is fed back through the Pick input of C1234. Since both the alertTop and alertBot inputs of C1234 are active, alertTop is selected by activating C1234's pickTop. This activates the Pick input of B12 and B12's alertTop is active, so the pickTop output of B12 is activated. This activates the Pick input of A1 and ultimately the pickTop output of A1 which is the Pick1 signal. All other PickX signals are inactive. In short, a Fischer Tree which picks alertTop over alertBot will form a priority encoder with the highest priority being the topmost alert. Similarly, a Fischer Tree which picks alertBot over alertTop will form a priority encoder with the highest priority being the bottommost alert. In either case, the time from alert to selection will be $O(2x\log_2(N))$.

In the VIPRAM_L1CMS implementation of a Fischer Tree, the PickX vector (i.e the vector formed from Pick1, Pick2, Pick3…, PickN) is captured at the rising edge of MClkA as one of the products of whatever pipeline stage the Fischer Tree is in. It is possible to extract the pick address directly from the Fischer Tree itself [8], but this approach is not fast enough for the VIPRAM_L1CMS implementation.

B. *The ioTier Pipeline*

1) *Pick Next Row*

- Inputs
  - Captured roadFlags (64 bits)
  - releaseFischerTree (1 bit)
- Outputs
  - FastOR (1 bit)
  - PickX vector

This pipeline stage is simply a Fischer Tree and this pipeline stage exists independently within each column of the ioTier. The FastOR output and the PickX vector of the Fischer Tree are the outputs of the pipeline stage. By virtue of the Fischer Tree, if the FastOR output is active, one and only one bit of the PickX vector will be active and it will be the bit corresponding to the highest priority Alert (roadFlag) input currently active. If the FastOR output is inactive, all bits of the PickX vector will be inactive and it means that no Alert (roadFlag) inputs are active.

Since all columns contain one of these pipeline stages and since they operate in parallel and, finally, since only one column can be selected to output at any given time, this means that the captured PickX vector of any column cannot change until that column has been released to do so by the next pipeline stage through the signal releaseFischerTree.

The Alert inputs to each Fischer Tree are the roadFlags captured by the last End-of-Event signal. These roadFlags are reset by the PickX vector outputs of this pipeline stage during the next pipeline stage. Consequently, every time a particular roadFlag is selected by the Fischer Tree and this pipeline stage has been released to advance its PickX outputs then that roadFlag is reset at the next rising edge of the master clock (MClkA). In this fashion, the selection of the Fischer Tree changes every time a new roadFlag is selected by the PickX output of this pipeline stage.

2) *Generate Row Address and Pick Next Column*

- Inputs
  - Each Column FastOR (64 bits)
  - Each Column PickX vector (64x64 bits)
- Outputs
  - Full Quadrant FastOR (1 bit)
  - Each Row address (64x6 bits)
  - Full Quadrant PickX vector (64 bits)

Each 64-bit PickX vector from the "Pick Next Row" stage is converted into a 6-bit binary address and captured as one of the outputs of this stage. Simultaneously, each 64-bit PickX vector is used to reset the corresponding roadFlag input to the "Pick Next Row" stage.

Each FastOR output from a "Pick Next Row" stage will be active if there is something to output in that column and inactive if not. Selecting a column to output is logically identical to selecting a row to output within a column of roadFlags. The solution, again, is a Fischer Tree. This time, the Alert inputs to the Fischer Tree are the 64 FastOR signals from the preceding pipeline stage. The FastOR output of this Fischer Tree will be captured as one of the outputs of this stage. The 64-bit PickX vector of this Fischer Tree will also be captured as another of the outputs of this stage. Each bit of the PickX vector also serves as the releaseFischerTree signal for the corresponding "Pick Next Row" stage.

3) *Generate Row Addr and Broadcast HaveData, Row Address and Column Address*

- Inputs
  - Full Quadrant FastOR (1 bit)
  - Each Row address (64x6 bits)
  - Full Quadrant PickX vector (64 bits)
- Outputs
  - Column Address (6 bits)
  - Row Address (6 bits)
  - Have Data (1 bit)

The 64-bit PickX vector from the "Pick Next Column" stage is converted into a 6-bit binary address and captured as Column Address output of this stage.

The Full Quadrant PickX vector is used to place one of the 64 6-bit Row addresses from the "Pick Next Column" stage onto a bus. This bus is captured as the Row Address output of this stage.

The Full Quadrant FastOR output from the "Pick Next Column" stage is captured as the HaveData output of this stage.

4) *Prepare for DDR Output*

- Inputs
  - Column Address (6 bits)
  - Row Address (6 bits)
  - Have Data (1 bit)
  - End-of-Event signal (1 bit)

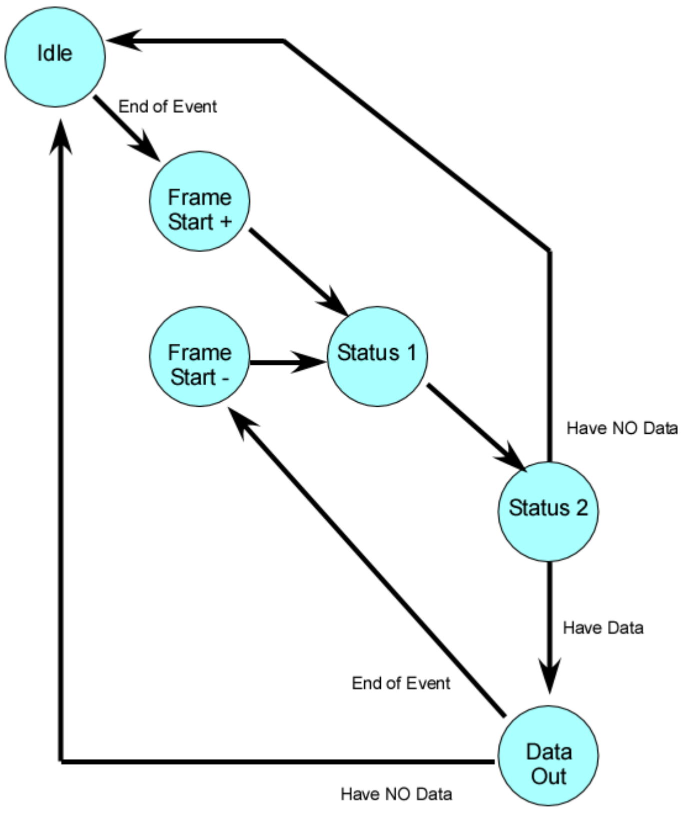


Fig. 10. The State Diagram that controls this Pipeline stage.

- Outputs
    - DDR Clock=1 Data (8 bits)
    - DDR Clock=0 Data (8 bits)

This pipeline stage is actually a state machine whose state diagram is shown in Fig. 10. There are two start states for the state machine. If the state machine is "Idle" – i.e. has no data to output – when the End-of-Event signal arrives, it moves to the "clean start" state (Frame Start +). If the state machine is "Data Out" when the End-of-Event signal arrives, it is still trying to output data from the previous event. The state machine moves to the "dirty start" state (Frame Start -). Regardless of whether or not the start is dirty or clean, the outputting of the next event must begin because the End-of-Event signal also captures new roadFlags into the ioTier. Any data from the previous event must be overwritten. The purpose of the clean start and dirty start is to notify the user of potential data loss in the previous event. The state machine then passes through two additional states, Status 1 and Status 2, to allow time for the ioTier pipeline to start generating data. Currently, Status 1 and Status 2 just output a fixed data pattern, but it will be possible to insert error or status data into them in future versions. From Status 2, the state machine returns to Idle if there is no data to output (i.e. if HaveData is inactive) or it moves on to Data Out if HaveData is active. The state machine remains in Data Out as long as HaveData is active.

The data output is shown in Table 1. All data displayed in the table is in hexadecimal except for the Data Out state which is shown in binary. In the data out state the two most significant bits are 00 during both the MClkA=1 and the MClkA=0 phases. The remaining 6 bits are the Row Address (when MClkA=1) and the Column Address (when MClkA=0).

TABLE 1. THE OUTPUT DEPENDING ON STATE

State	MClkA = 1	MClkA=0
Idle	0xC3	0xCC
Frame Start +	0x40	0x40
Frame Start -	0x40	0x4F
Status1	0x85	0x80
Status2	0x8A	0x80
Data Out	00 Row Addr	00 Col Addr
Bad1	0xD3	0xDC
Bad2	0xE3	0xEC

VII. SUMMARY

The VIPRAM_L1CMS architecture is a readout architecture pipelined by event across two vertically integrated VLSI tiers. It takes advantage of the 3D integration to pass matched roads from the lower PRAMtier to the upper ioTier instantly at the conclusion of each event. An additional pipeline, this one advanced by the master clock and confined to the ioTier first sparsifies each row and then funnels them to the output by sparsified column. The output each quadrant of the chip is designed to indicate if the chip is idle or not and if there was or was not any data loss from the previous event in the case of high event rates.

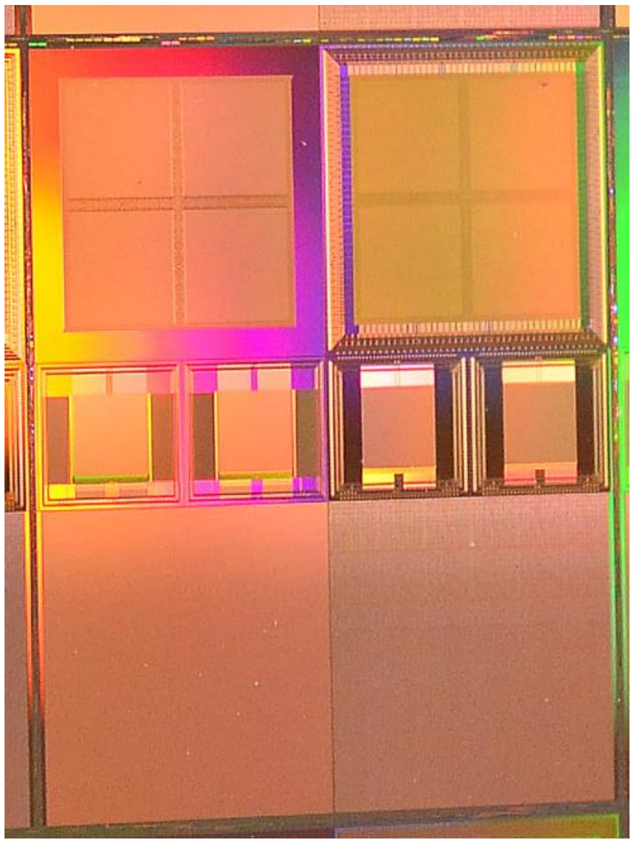


Fig. 11. A picture of the complete reticule. The chip on the top and right is VIPRAM_L1CMS ioTier. The chip on the top and left is VIPRAM_L1CMS PRAMtier. The four chips in the middle are the VIPRAM3D chips and the two on the bottom are the VIPIC chips.

The first implementation of this architecture was submitted in March of 2016. Wafers were returned from Global Foundries in August of 2016 (see Fig. 11) and 3D fabrication began in September of 2016. Final chip delivery is anticipated in early 2017.

REFERENCES

[1] M. Dell'Orso and L. Ristori, "VLSI Structures for Track Finding," Proceedings in Nuclear Instruments and Methods, vol. A278, pp. 436-440, 1989.

[2] A. Annovi et al., "The GigaFitter: A next generation track fitter to enhance online tracking performances at CDF," Nuclear Science Symposium Conference Record (NSS/MIC), pp.1143-1146, 2009.

[3] A. Annovi, et al., "A VLSI Processor for Fast Track Finding Based on Content Addressable Memories," IEEE Trans. Nucl. Sci, vol. 53, no. 4, pp. 1-6, 2006.

[4] T. Liu, J. Hoff, G. Deptuch, R. Yarema, "A New Concept of Vertically Integrated Pattern Recognition Associative Memory", published in the Proceedings of the TIPP 2011 Conference, DOI number 10.1016/j.phpro.2012.02.521

[5] G.W.Deptuch, G.Carini, P.Grybos, S.Holm, P.Kmom, R.Lipton, P.Maj, D.P.Siddons, A.Shenai, R.Szerzygiel, R.Yarema, " Fully 3D-integrated Pixel Detectors for X-Rays," IEEE Trans. Electrib Devices, vol. 63, no. 1, pp. 205-214, Jan. 2016.

[6] G.W. Deptuch, "Status of 3D Integration", presented at the 10th International Meeting on Front End Electonics (FEE 2016), Krakow, Poland, 2016. Available: https://indico.cern.ch/event/522485/contributions/2145788/attachments/1282830/1906666/FEE_05_2016.pdf

[7] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) Circuits and Archtectures: A Tutorial and Survey", IEEE Journal of Solid-State Circuits, Vol. 41, No. 3, pp. 712-727, March, 2006.

[8] P. Fischer, "First Implementation of the MEPHISTO Binary Readout Architecture for Strip Detectors", Nuclear Instruments and Methods in Physics Research A, vol. 461, pp. 499-504, 2001.