Conflux – An Asynchronous 2-to-1 Multiplexor for Time-Division Multiplexing and Clockless, Token-less Readout

James R. Hoff

***Abstract*[1]— This paper describes an independent, self-arbitrating asynchronous two-to-one multiplexor called Conflux. Conflux can be used to implement clockless and token-less Time-Division Multiplexing between two sources. This capability can then be used to create a clockless and token-less aggregate data bus which can be used as a complete asynchronous readout architecture for a chip, a part of a chip or an entire multi-chip system. Conflux utilizes a classic, four-phased asynchronous handshake on both of its input ports as well as on its output port. Asynchronous Request is bundled with Data to ensure consistent propagation delay. The Request is also implemented as a differential signal to account for propagation delay differences between Logical Ones and Zeros. Arbitration between the two Conflux input streams is accomplished by three Set-Reset latches. Finally, Conflux cells were used successfully to implement the readout architecture of the FCP130 prototype chip. Tests indicate single Conflux stage delays of 1.8ns and a bandwidth of approximately 11 Gbps in 130nm CMOS.**

Index Terms—Asynchronous Circuits, Multiplexing, Time Division Multiplexing

I. Introduction

Big Physics experiments are, for the most part, growing larger and more intricate [1]-[4]. As the size and complexity increase, so too does the difficulty associated with delivering the accurate, high-speed clocks necessary for synchronous readout [5]. To make matters worse, in front end systems, the same clocks that are necessary for synchronous readout have become a source of noise interfering with the very signals that the front ends are trying to detect. Conflux originated as engineering research into a high-speed tracking trigger readout for the Compact Muon Solenoid (CMS) experiment located at the Large Hadron Collider (LHC) in Geneva, Switzerland [6]-[8]. An on-chip architecture became necessary to read out a 2-dimensional array of pixels, each of which might or might not be a source of data at any given point in time. These intermittent pixel outputs needed to be merged into a single data stream for output from the chip. Each complete pixel chip would itself be part of a larger array of chips that have to get their data to a concentrator and, from there, off detector. In an effort to minimize data driving distances, the idea was to have each chip accept output from its neighbor (for example) to the left and to time-division multiplex (TDM) that left-neighbor-data stream with its own internally generated data stream and to pass a merged data stream on to the chip's neighbor to the right. Conflux emerged as a clockless architecture that could deal with both the intra-chip and inter-chip data readout.

The following paper will describe Conflux completely. Section II will discuss the background and general requirements of a clockless, aggregate readout architecture. It will describe the chosen protocol, and the architecture of Conflux. Section III will go into the circuit specifics of the Conflux architecture. Section IV will describe the fabrication and testing. Finally, Section V will conclude.

II. A Clockless, Aggregate Readout Architecture

Fig. 1 Generic readout architectures: (a) A shared-bus architecture (b) An aggregate, clocked bus architecture and (c) an aggregate, clockless bus architecture.

All Big Physics detector systems with bussed readout architectures can be generalized to consist of a collection of data generators that seek to send their data to one or more data destinations – e.g. an array of photodetectors sending their data to a data concentrator ASIC or, alternately, an array of pixel detectors sending their data off-chip. Many systems fit this generalized description.

In shared-bus architectures (Fig. 1(a)), each data source is conditionally and individually connected to a common bus that

[1] 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), is with the Fermi National Accelerator Laboratory, Batavia, IL 60510 USA

has its other end connected to the data destination. At any given instant, one and only one data source is connected to the data destination via that bus and only that data source can drive its data to the destination. Such a system requires a clock distributed throughout the shared bus and a means of bus arbitration like a token or a priority encoder. Shared-bus systems are relatively simple, but they suffer from several drawbacks. As the number of individual data generators grows, the size of the bus grows as does the parasitic load on all drivers (input and output). This load sets both the maximum frequency and the minimum power consumption of the shared-bus system [9].

In an aggregate-bus system (Fig. 1(b)), each data generator is paired with its own local bus control circuit and its own local 2-to-1 multiplexor. At any given instant, each local bus control circuit is aware of whether it must drive its local data or drive data from the previous neighbor, and it sets the local multiplexor accordingly. In this type of system, the parasitic loading problem is dramatically reduced because each stage is only required to drive data to its next neighbor. The complexity arises from the implementation of the local bus controllers and multiplexors. On one extreme, they can be implemented as a logical equivalent to the shared-bus system. The select signals of the local bus controllers function as one-hot logic and then one multiplexor grabs data from its local source and all others are in previous-neighbor-to-next-neighbor mode. The maximum clock frequency of such a system is equal to

$$f = \frac{1}{N \times t_D}$$

Where N is the number of data sources in the array and t_D is the propagation delay of a single 2-to-1 multiplexor. On the other extreme, data can be advanced from one neighbor to the next synchronously with a readout clock. Decisions are made locally as to whether a single data source has data and, if so, which data (local or previous neighbor) is to advance. Complexity in this system arises from determining what to do in the event of data collisions.

Both the classic shared-bus and the aggregate-bus architectures of Fig. 1(a) and (b) require a clock. To eliminate the clock and create the architecture shown in Fig. 1(c), several requirements become apparent.

1. The circuit must still behave as a 2-to-1 multiplexor (mux) that gets input from two interfaces (local and previous neighbor) and provides output to one interface (next neighbor).
2. The output stream is a Time Division Multiplexed (TDM) stream of the two input streams.
3. Both input streams must have predictable rights to the output stream.
4. Given the stochastic nature of data from the inputs, the circuit must be able to handle any conceivable arrival sequence – e.g. repeated data from local with nothing from previous neighbor, repeated data from previous neighbor with nothing from local, data arriving from both local and previous neighbor, collisions of data, etc.
5. As a consequence of the stochastic nature of input data, the circuit must be able to recognize that it has captured data in one or both of its inputs and it must be able to hold that data until it has passed that data on to the next neighbor.
6. The circuit must arbitrate for itself depending on all available information. That information can only consist of whether it has local data, whether it has previous neighbor data and the current acknowledgement state of the next neighbor.
7. Whatever asynchronous protocol is chosen, each input should be capable of maintaining and ultimately completing that protocol independently and irrespective of whether the input has won or lost the current arbitration.

A. Asynchronous Protocol

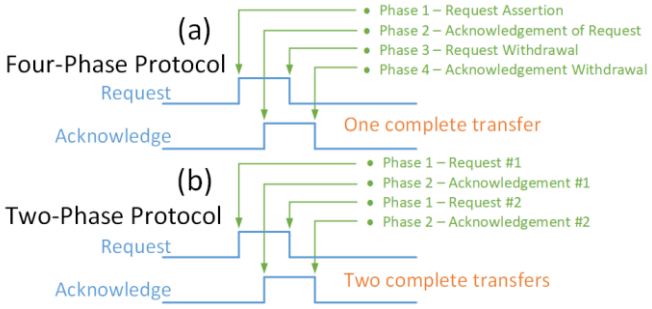

Fig. 2 (a) Single transfer under a Four-Phase Protocol (b) Two transfers under a Two-Phase Protocol.

The two canonical asynchronous protocols are Level Signaling (a.k.a. Four-Phased) and Transition Signaling (a.k.a. Two-Phased) [10]-[12]. They are shown in Fig. 2. Both protocols are typically implemented as Bundled Data Protocols which is to say that both the Request and the Data are transmitted together such that they have similar (preferably identical) drivers, similar (preferably identical) loads and wires that are impedance matched as much as possible. This helps minimize propagation delay dispersion between the request and data bits. Using a differential signal as Request can further minimize the adverse effects of propagation delay dispersion. Since both a Logical 1 and a Logical 0 would be transferred as the Request, any potential propagation delay differences between Logical 1s and Logical 0s are guaranteed to be taken into account simply by looking at the propagation of the Request.

The requirement that the clockless Mux be able to handle any conceivable arrival sequence makes the Four-Phase protocol very appealing. Looking again at Fig. 2(a), all transitions begin and end with the same logical levels in the Four-Phase protocol. Therefore, previous stages always know *a priori* what logic levels should presented before and at the start of a new transfer. If a Two-Phase protocol was used, the proper Acknowledge transition to one input would depend on the transition direction of the corresponding Request. This requirement would exist regardless of which transmission won an arbitration and regardless of how many transmission victories have been granted to the other input. Clearly, extra logic would be required to keep track of the appropriate transition direction for

the Request and Acknowledge of each input.

It is for these simple reasons that a Four-Phase Bundled Data protocol with a differential signal as the Request was chosen for Conflux.

B. *MOUSETRAP*

MOUSETRAP [13] is a simple, elegant, high-performance asynchronous pipeline stage. It is not a multiplexor. Also, it uses Transition Signaling, so it is not directly applicable to the clockless, aggregate multiplexor problem as defined herein. However, it has several valuable features that are of use to Conflux. It employs level-sensitive latches to capture the Data and Request from a previous stage. (See Fig. 3 and note that the figure contains three complete MOUSETRAP pipeline stages in series.) A stage's *captured* Request forms the Acknowledge to the previous stage. All the latches of a stage are simultaneously changed from transparent to opaque by a logical function of the stage's captured Request and the Acknowledge from the next stage. Since this logical function is an exclusive-NOR and the latches are positive active, then the latches are transparent whenever the captured Request is equal to the Acknowledge from the next stage. This is to say that when the captured Request of *this* stage is equal to the captured Request of the *next* stage, then the latches of *this* stage are made transparent. Since MOUSETRAP uses Transition Signaling if the captured Request of *this* stage is equal to the captured Request of the *next* stage, then the *next* stage has captured the bundled data of *this* stage and *this* stage is now free to capture new data. If the captured Request of the *next* stage is not equal to the captured Request of *this* stage, then the *next* stage has not yet captured the bundled data of *this* stage and *this* stage must remain opaque and maintain its captured data bundle.

MOUSETRAP's use of level-sensitive latches and its processing of captured Request signals is what is so elegant and so useful to Conflux. The level-sensitive latches help ensure that the Request and Data are truly bundled in that they have the same drivers, the same loads and, if properly routed, the same parasitics. This serves to minimize the dispersion in propagation delay between the different bits of the bundled data. Since the decision to make the latches transparent or opaque is performed on signals that have already passed through the latches, the latches, in effect, are closed after the Data and Request have already entered the stage.

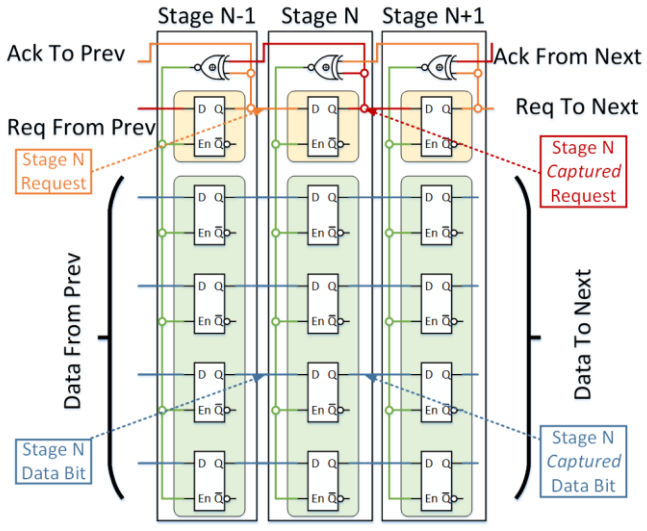

Fig. 3. A sketch of three MOUSETRAP pipelines in series. Note that the same latches are used for both the Request from Previous and the Data from Previous. The captured Request becomes the Acknowledge to Previous and it is gated with the Acknowledge from Next to form the Enable of the MOUSETRAP latches.

To account for possible propagation delay differences between Logical 1s and Logical 0s, Conflux employs a differential signal as its Request and therefore requires two Request latches. Of course, since Conflux uses a Four Phase protocol and since it must arbitrate between two input streams, the capture logic of Mousetrap (i.e. the XNOR gate) will not work for Conflux.

C. *Conflux*

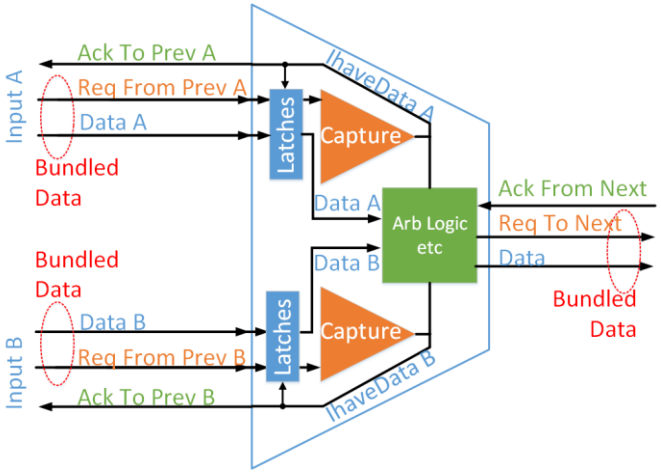

Fig. 4 A block-level sketch of the entire Conflux multiplexor.

Fig. 4 shows a block-level sketch of the Conflux multiplexor. It has two independent, bundled data inputs (Input A and Input B) and one bundled data output. Inputs are captured by two, independent level-sensitive latch sets. These latches are enabled by IhaveData A and IhaveData B, respectively. These signals are generated by independent Capture Logics (one Capture Logic per latch set). These same signals become the Acknowledge to Previous signals (Ack To Prev A and Ack To Prev B) as well as the inputs to the Arbitration Logic. Finally, the respective Captured Data vectors are passed through classic combinatorial multiplexors whose select signal is controlled by the result of the Arbitration Logic. A Request to Next is formed in the Arbitration Logic.

Looking at the block diagram of Fig. 4, imagine a previous stage raises a Request, for example, on Input B. This is Phase 1 of Fig. 2(a). Since this is a bundled data protocol, this Request propagates with its Data. When the Request is captured by the Input B Latches, the Capture Logic recognizes the Request and issues IhaveData B. This is the Acknowledge to Previous B that completes Phase 2 of Fig. 2(a). Since the bundled data has been captured and Phase 2 completed, the previous stage is free to withdraw its Request, completing Phase 3. The Acknowledge to Previous will remain active as long as the

Capture Logic recognizes that it has data in its Latches, and it will do so until it wins arbitration and passes its data along to the next stage. Consequently, in this example, the previous stage of Input B will remain in Phase 3 until its Request wins arbitration. Therefore, the previous stage of Input B cannot begin a new Request until its current Request wins arbitration and output stream passes the Input B data to the next neighbor the Acknowledge From Next.

III. Conflux Circuitry

A. *Latch and Capture Logic*

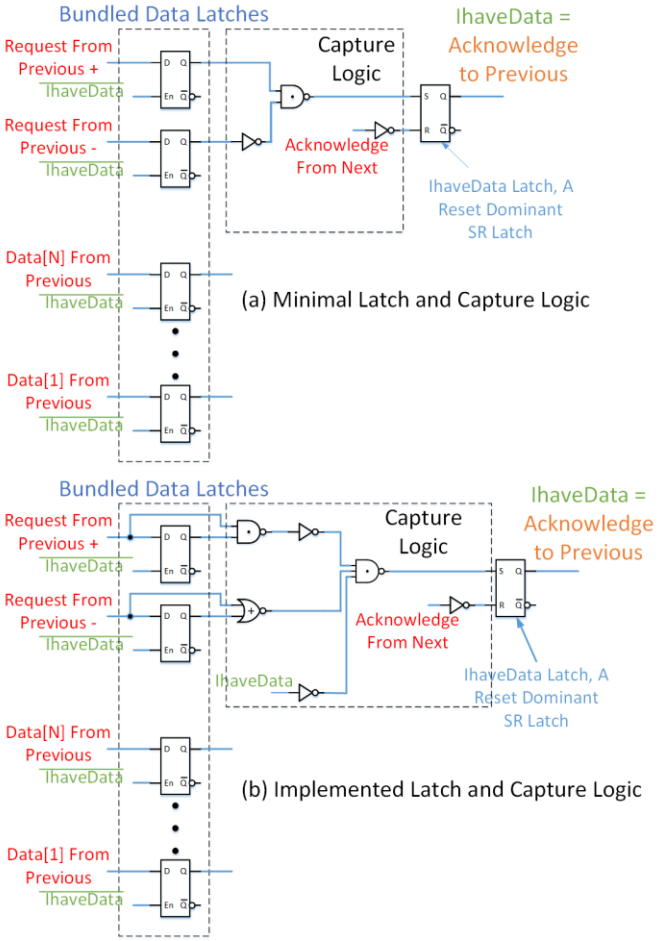

Fig. 5 (a) Minimal Latch and Capture Logic necessary for a differential Request (b) the actual Latch and Capture Logic implemented in the first test of Conflux. (It is worth noting that if the Request is single ended, all that is necessary is for the Request.

Because of the Four Phase protocol and because of the basic requirements of an independent, self-determining, asynchronous multiplexor, the Latch and Capture Logic (Fig. 5) must be able to capture bundled data, recognize that bundled data is present in its latches and then hold the bundled data indefinitely. The signal that does this, IhaveData, is basically the gatekeeper of its input stream. It functions as the Acknowledge To Previous. When active, it locks the previous neighbor at Phase 3 of its Four-Phased transfer and prevents any additional requests on this stream. When active, the latches of the Latch and Capture Logic are made opaque. When inactive, the latches are transparent, Phase 4 has been completed and the Conflux is awaiting the next request on this input stream. IhaveData must be efficiently recognized and latched until used, and once used, the signal must be unambiguously erased.

The IhaveData Latches annotated in Fig. 5re are Reset-Dominant SR Latches – i.e. an SR Latch that has one Set condition (S=0, R=1), two Reset conditions (S=0, R=0 or S=1, R=0) and one Hold condition (S=1, R=1) (see Appendix A Section B). Consequently, when the Reset of the IhaveData Latch is active, the Latch and Capture Logic have no choice but to erase IhaveData regardless of the current activity on the Request From Previous (e.g. Is the previous stage trying to enter Phase 1 for a new bundled data transmission?). Furthermore, since IhaveData is also the Acknowledge to the previous stage, no Acknowledge To Previous can possibly be sent until that reset is withdrawn. Section III.E will show that the reset of the IhaveData Latch is actually the Acknowledge from the next stage (Acknowledge From Next). Therefore, the Reset-Dominant IhaveData Latch ensures that the current stage cannot possibly advance the next bundled data transmission to its Phase 2 until the current bundled data transmission has cleanly completed Phase 4 (see Fig. 2(a)). In short, the Latch and Capture Logic capture the incoming bundled data, lock the previous neighbor and no further than Phase 3 of the handshake, hold the data and request indefinitely until it wins arbitration, outputs the data on the output stream and receives acknowledgment from the next neighbor.

Recognition of the presence of data within the Latch and Capture Logic is simple given the bundled data protocol. Data is present when the Request is active.

1. For a single-ended, positive-active Request, the Request is active when *captured* Request From Previous (i.e. the output of the Request From Previous Latch) is high.
2. For a Request that is a differential signal, Request is active when the *captured* Request From Previous + is high and the *captured* Request From Previous – is low. A NAND gate of the captured Request and $\overline{\text{Request}}$ signals is used as shown in Fig. 5(a).
3. Fig. 5(b) shows the circuit that was actually used in this first implementation of Conflux. Here Request is active when
    a. both the *uncaptured* Request From Previous + input and the *captured* Request From Previous + are high and
    b. both the *uncaptured* Request From Previous – input and the *captured* Request From Previous – are low and
    c. IhaveData is low (inactive).

The chosen implementation prevents transient noise (such as charge that could be induced by high energy particles) from accidentally activating IhaveData. Speaking more generally, this more complex implementation of the capture logic shows that functionality can be added within the Latch and Capture Logic depending on the user's definition of what constitutes the presence of data within the latches. For example, if dual-rail data protocols are being used [10], data

recognition could be gated until all the data has successfully arrived; data recognition could also be withheld until a successful parity check; etc.

B. *Demultiplexer for Acknowledge from Next*

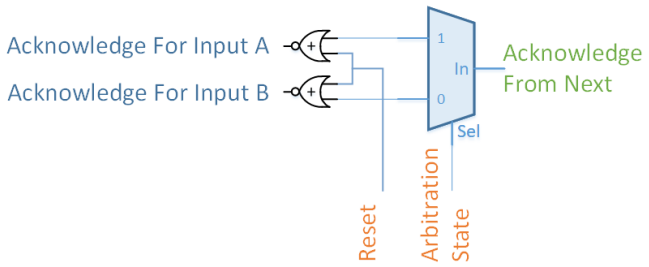

Fig. 6 The demultiplexer and reset logic for the Acknowledge From Next.

Fig. 4 clearly indicates that there must be two independent Latch and Capture Logic sets, one for each input stream. It is obvious that the same Acknowledge from Next signal cannot be distributed in common to both sets. If both IhaveData A and IhaveData B were active and yet the Arbiter chose IhaveData A, a common Acknowledge from Next signal would reset both IhaveData signals even though IhaveData B was not the data being acknowledged. The Acknowledge from Next signal must be demultiplexed depending on the state of the Arbiter.

This simple circuit is shown in Fig. 6. This circuit also provides a convenient location for an asynchronous reset signal.

C. *Arbitration Logic*

Arbitration in digital systems is hardly new [14] and Conflux shares much in common with existing systems. Arbitration in Conflux occurs in response to the state of IhaveData A and IhaveData B. However, even though IhaveData A and IhaveData B can change at any time, the state of arbitration can only be allowed to change *between transfers* – i.e. *after* the withdrawal of an Acknowledge from Next and *before* the next assertion of Request to Next. So, if there is an active transfer on the output stream and the state of IhaveData A or IhaveData B changes, then arbitration must wait. If there is no active transfer on the outputs stream, then arbitration can occur immediately. Additionally, once an arbitration decision has been reached, it must be maintained throughout the Four-Phase Protocol of the output stream.

Ignoring questions of metastability for the moment, a simple solution to arbitration in Conflux is to use a Hold-Dominant SR Latch – i.e. an SR Latch that has one Set condition (S=0, R=1), one Reset condition (S=1, R=0) and two Hold conditions (S=0, R=0 or S=1, R=1) (see Appendix A, Subsection D). If such a latch is fed with logic as shown in Fig 7(a) then

1. Between asynchronous transfers, if both IhaveData A and IhaveData B are inactive, the Arbitration Latch makes no change from its previous arbitration state because this is one Hold condition of the latch.
2. Similarly, between asynchronous transfers, if both IhaveData A and IhaveData B are active, the Arbitration Latch makes no change from its previous arbitration state because this is another Hold condition of the latch.
3. Between asynchronous transfers, if either IhaveData A or IhaveData B are active, the Arbitration Latch selects the appropriate input stream because one active IhaveData and one inactive IhaveData is either the Set or Reset condition of the latch.
4. During asynchronous transfers, the Arbitration Latch is kept in one of its Hold conditions by the presence of the Request to Next and/or Acknowledge from Next signals.

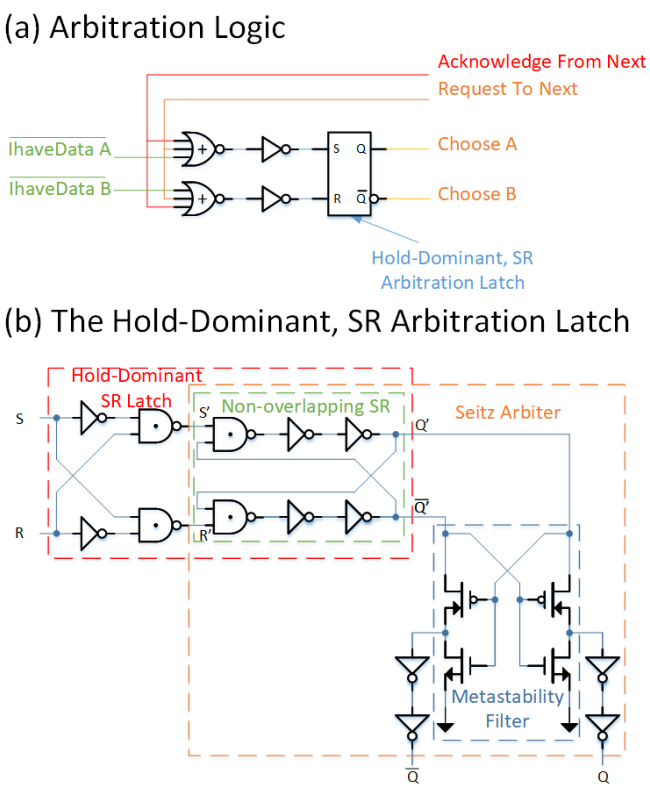

Fig 7. (a) The complete Arbitration Logic surrounding the Hold-Dominant, SR Arbitration Latch. (b) The Hold-Dominant, SR Arbitration Latch itself.

Of course, while a Hold-Dominant SR Latch solves the logic of arbitration in Conflux, metastability and mutual exclusion must be also considered. Fig 7(b) shows a complete arbitration latch. It is a combination of the following:

1. A Hold-Dominant SR Latch that sets up the subsequent arbiter so that it behaves as a self-arbitrating asynchronous two-to-one multiplexor
2. A non-overlapping SR Latch that enhances mutual exclusion, and
3. A Seitz Arbiter or MUTEX [10]-[12][15].

The Hold-Dominant SR Latch is highlighted by the red box within Fig 7(b). It is constructed by adding simple combinatorial logic in front of a basic SR Latch – i.e. an SR Latch that has one Set condition (S=0, R=1), one Reset condition (S=1, R=0) one Hold condition (S=1, R=1) and one Forbidden condition (S=0, R=0) (see Appendix A, Subsection A). The non-overlapping SR Latch is highlighted by the green box within Fig 7(b). It is constructed by adding delay (in this

case, two inverters) within the feedback structure of an SR Latch. The Seitz Arbiter is highlighted by the gold box within Fig 7(b). It is constructed by combining an SR Latch with a Metastability filter which itself is highlighted by the blue box within Fig 7(b).

1) Mutual Exclusion

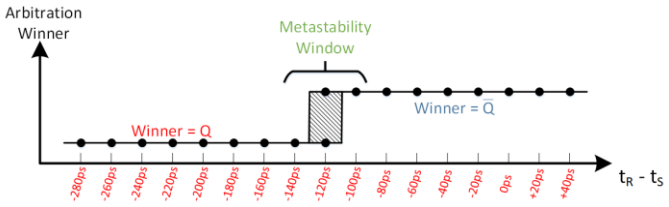

Fig. 8. A graph of the Arbitration Winner as a function of the time difference between the activation time of IhaveData B (t_S) and the activation time of IhaveData B (t_R) given that the Conflux is between transfers and that Q (IhaveData A) won the last arbitration. Metastability Window not drawn to scale.

The combinatorial logic added to make a Hold-Dominant Latch is symmetric across S and R and is such that an active S (or R) will actually block R (or S) before it activates the S' (or R') of the basic SR Latch inside the Hold-Dominant SR Latch. This has a beneficial side effect on arbitration in Conflux. Not only does the Hold-Dominant Latch perform the correct logical functions enumerated above, but it guarantees that a change to the state of S or R will not cause a re-arbitration even before Request To Next activates and blocks re-arbitration through the logic of Fig 7(a). In short, it helps guarantee that only time-of-arrival determines the arbitration winner.

The extra delay inserted in the feedback paths of the non-overlapping SR latch force both Q' and $\bar{Q}'$ to Logical 1s for a period of time equal to the propagation delay of the NAND gate plus the two inverters. This coupled with the Metastability Filter force Q and $\bar{Q}$ to Logical 0s for the same time, ensuring that there is a fixed period of inactivity between arbitrations with different winners. In other words, it provides predictable mutual exclusion. Fig. 8 shows the arbitration results when both IhaveData A and IhaveData B activate very close to one another. The x-axis is the simulated time difference in picoseconds between the activation of IhaveData B and IhaveData A assuming that the Conflux is between simulations and IhaveData A won the last arbitration. A similar graph can be shown plotting the simulated time difference between the activation of IhaveData A and IhaveData B assuming that the Conflux is between simulations and IhaveData B won the last arbitration. Of course, this is obvious from the fact that the circuits are symmetrical with respect to S and R. The arbitration winner depends only on time of arrival and the winner of the last arbitration. Arbitration winner favors the last winner because of the mutual gating effect of the Hold-Dominant SR Latch. At time differences less than approximately 120ps activation of S (in Fig 7(b)) does not have enough time to pull S' below threshold and the basic SR Latch does not change state. Next, the arbitration winner does not change once arbitration has been decided. In fact, if metastability is not a concern, the Hold Dominant logic and the non-overlapping SR Latch do a very good job of ensuring mutual exclusivity on their own. If desired, the metastability filter of Fig 7(b) can be replaced by simple inverters between Q' and $\bar{Q}$ and between $\bar{Q}'$ and Q.

2) Metastability

The positive feedback present in any latch implies that there are two stable states (($Q = 0, \bar{Q} = 1$) and ($Q = 1, \bar{Q} = 0$)) and one metastable state ($Q \cong \bar{Q} \cong \frac{VDD}{2}$) [11]-[13][15]. Ordinarily, this is not a problem, but when S and R change very close to one another in time, this can cause a time delay in resolving the final state of the latch. In most SR Latches and in particular in the Conflux arbiter, this time delay is the result of an oscillation in the feedback network whose duration is dependent on the arrival times of S and R, the rise and fall times of S and R, the propagation delay of the Latch and the rise and fall times of Q and $\bar{Q}$. In all cases there is a time window (sketched in Fig. 8) where disruption of normal operation can occur. By simulation, this window is approximately 20ps wide for the full arbiter shown in Fig 7(b). If the metastability filter is eliminated and replaced by inverters, the window is reduced to approximately 10ps wide due to the reduction in the rise and fall times of Q and $\bar{Q}$. If the non-overlapping SR Latch is replaced by a simple SR Latch, the metastability window reduces to the single digit picosecond range due to the reduction in the propagation delay of the latch.

However, while the metastability window is reduced by these measures, the consequences of metastability can increase. In the presence of metastability, the metastability filter holds the outputs of the arbiter at ($Q = 0, \bar{Q} = 0$), meaning that neither state wins until the metastability resolves. Replacing the metastability filter in Fig 7(b) with inverters does reduce the metastability window, but it does filter the oscillations from Q and $\bar{Q}$.

D. Request Release Logic

The purpose of the Request Release Logic is, first, to convert the internal, single-ended Request signal into a differential Request To Next signal in keeping with the chosen bundled data format of Conflux and, second, to ensure the correctness of the bundled data output stream. Correctness means that the Request within a bundled data is not active until its Data is stable [12].

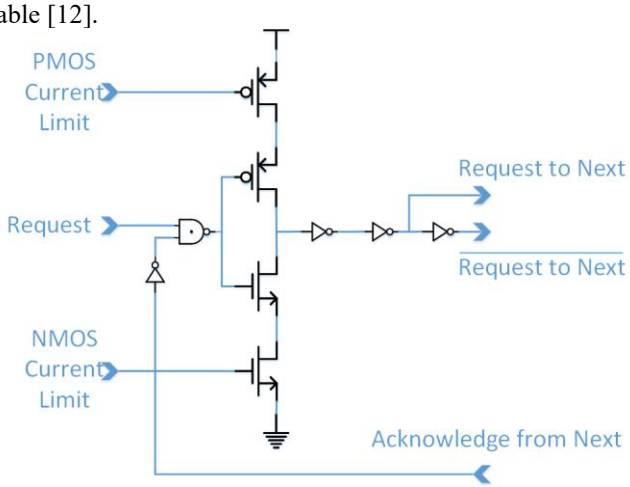

Fig. 9. A schematic of the Request Release Logic

The Request Release circuitry is shown in Fig. 9. The complete Conflux is shown in Fig. 11 and in that figure, the Request Release logic is annotated with letter (k). The boxes "Captured Data #1" and "Captured Data #2" represent the data captured in the two Latch and Capture Logics. They are passed through a multiplexor selected by the output of the arbiter and the selected data becomes Data To Next. Similarly, the two IhaveData signals are passed through an identical multiplexor also selected by the output of the arbiter. However, the selected IhaveData does not become Request To Next directly. First, it passes through the Request Release circuitry. The IhaveData signals are used to form Request To Next because they come into existence after data is present in an input stream and then they are removed unequivocally by the presence of Acknowledge from Next through the action of the Reset-Dominant Latch in the Latch and Capture Logic. Fig. 8 shows that the Arbitration circuitry does not permit the Arbitration Winner to change back and forth. In other words, regardless of which input stream won the last arbitration, request and data from this arbitration will be released together and will not be removed until the Acknowledge from Next is received. Finally, only the Request, not the Data is passed through the Request Release Logic, guaranteeing that Request To Next comes into existence after the Data of the output stream. Moreover, this prototype contains a variable delay element controlled by NMOS and PMOS Current Limit, permitting user control of the duration of the delay between Data To Next and Request To Next.

E. Data Flow

Fig. 10 is a basic timing diagram for two transfers across Conflux, the first from Input Stream A while the Conflux is between transfers, and the second from Input Stream B while the Conflux is engaged in the first transfer.

Initially, all latches in the conflux are transparent. The bundled data arrives from Input Stream A and the most immediate consequence is that IhaveData A activates. This, in turn, results in the activation of Acknowledge To Previous A and the changeover of the Input Stream A latches to opaque. The Inputs Stream B latches remain transparent. The activation of IhaveData A also results in an arbitration whose outcome is Choose A. This causes the captured data, DA1, to be routed to the Data To Next and the IhaveData A to be routed to the Request Release Logic whose outcome is the activation of Request To Next. After some time, Acknowledge From Next activates, resulting in the removal of Request To Next and the Resetting of IhaveData A. The resetting of IhaveData A results in the removal of Acknowledge To Previous A and the changeover of the Input Stream A latches to transparency.

The arrival of bundled data on Input Stream B is essentially the same except that the activation of IhaveData B does not lead directly to a new arbitration. The next arbitration must wait until the arrival and removal of Request To Next and Acknowledge from Next. The outcome of the second arbitration is Choose B and the next transfer continues as did the last one.

These transfers are simple, but what happens if there is more data on Input Stream A? Can one input stream lock out another? The Latch and Capture Logic's IhaveData latch is reset upon reception of Acknowledge From Next. This means that Acknowledge to Previous A also is deactivated, finishing the last phase of Input Stream A's four-phase transfer, and permitting Input Stream A to start its next transfer. However, the fact that the IhaveData latch is reset-dominant means that as long as Acknowledge From Next is active, IhaveData A cannot re-activate. Once Acknowledge From Next deactivates, the IhaveData Latch of Input Stream A must be set and the signal must be driven to the arbitration latch. Meanwhile, if IhaveData B is already active when Acknowledge From Next deactivates, IhaveData B is already at the arbitration latch and is guaranteed by the logic of the Hold Dominant latch to win the arbitration. Due to the symmetry of the arbitration logic, in the presence of an abundance of data, the arbitration winner will always alternate between transfers from one input stream to the other. One input stream cannot resource starve the other.

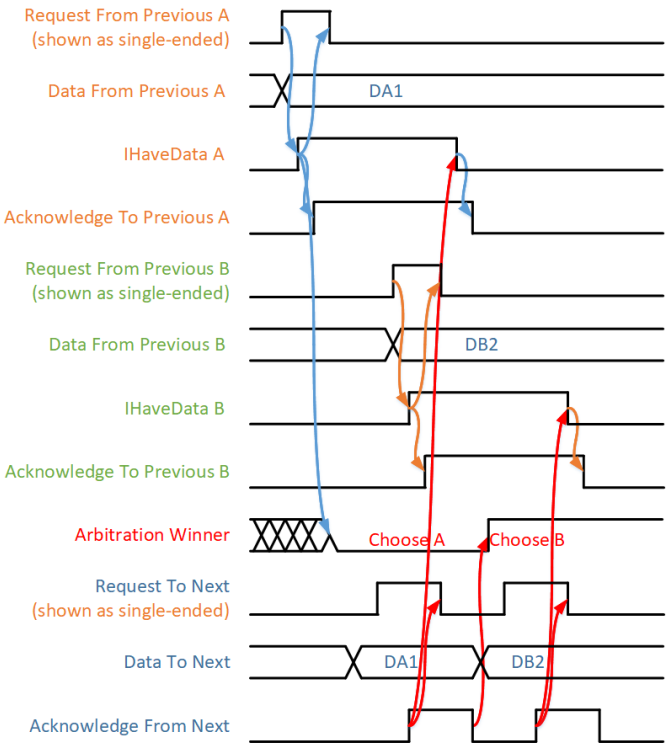

Fig. 10. A timing diagram of two transfers across Conflux, one from Input Stream A and the next from Inputs Stream B. It is meant to illustrate timing flow in Conflux.

F. Complete Conflux

The complete Conflux as designed for this first implementation is shown in Fig. 11. (a) and (b) highlight the two input streams, each with a differential Request From Previous and single-ended Data From Previous. (c) and (d) highlight the two Latch and Capture Logics. Note that the slightly more complicated version shown in Fig. 5(b) is used. It was known at the time of this chip's design that this chip could be used in a high-radiation environment. This implied that high energy particles could be incident on this chip. Under such circumstances, transient charge can be deposited randomly

on this circuitry through processes collectively called Single-Event Effects (SEEs) [17]. A complete description of SEEs is beyond the scope of this paper. However, a standard heuristic analysis of the consequences of SEEs led to the determination that it was possible to prevent accidental activation of IhaveData by requiring:

1. both the *uncaptured* Request From Previous + input and the *captured* Request From Previous + are high and
2. both the *uncaptured* Request From Previous – input and the *captured* Request From Previous – are low and
3. IhaveData is low (inactive).

The Capture Logic in Fig. 5(b) and Fig. 11 is functionally equivalent to the Capture Logic of Fig. 5(a). However, the extra circuitry in Fig. 5(b) and Fig. 11 acts to filter out transient signals that are shorter than the propagation delay of the Latch and the NAND gate. This should cover most of the noise anticipated through incident radiation.

The latches used for both Request and Data in the Latch and Capture Logic are Earle Latches [15] (See Appendix B). Simulations indicate that ordinary D-type Latches work as well. However, the fact that Earle Latches are free of static hazards was viewed as a positive in this first implementation.

Fig. 11(e) and (f) highlight the fact that the IhaveData output of each Latch and Capture Logic ((c) and (d)) become the Acknowledge to Previous of each respective input stream. They also become the inputs to the Arbitration Logic (Fig.

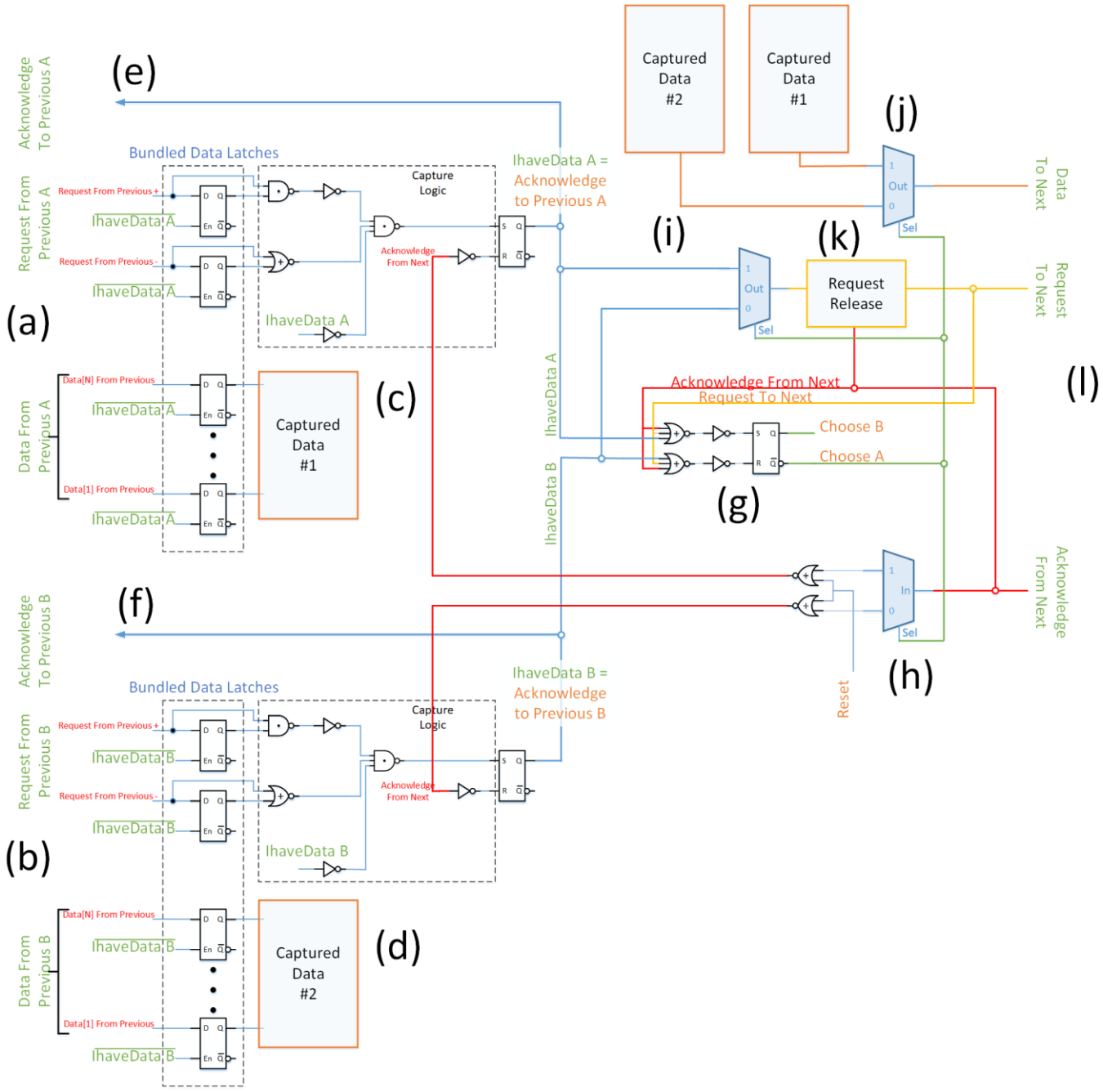

Fig. 11 A sketch of the complete Conflux cell

11(g)). Given that this was a prototype chip that would not be used in a wider system, it was decided not to implement the metastability filter of Fig 7(b) and instead use the two inverters. This could possibly allow additional testing if time and circumstances allowed. In the end, the narrowness of the metastability window meant that this was never a problem.

The outputs of the Arbitration Logic are connected to the Select inputs of two multiplexors and one demultiplexor. The demultiplexor is the logic of Fig. 6 and is highlighted as Fig. 11(h). The two outputs of this logic are connected to the reset of the Latch and Capture Logic (Fig. 5(b)) and steer the Acknowledge from Next to the appropriate Latch and Capture Logic. Fig. 11 (i) and(j) highlight the fact that the arbitration result steers the appropriate IhaveData signal to the Request Release Logic (k) and steer the appropriate Captured Data to the Data To Next output.

IV. Fabrication and Testing

A. Two Basic Readout Configurations

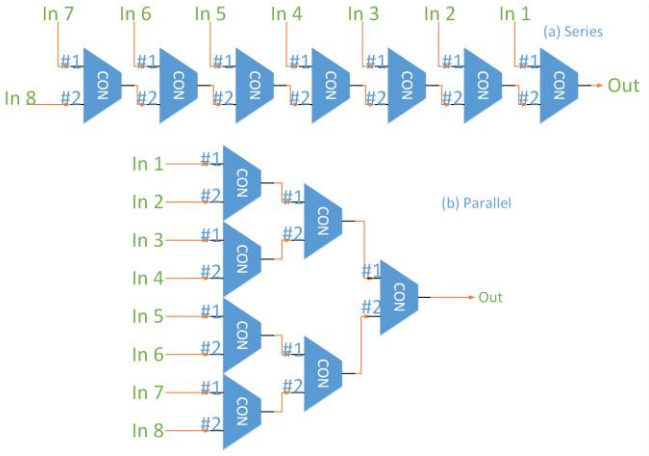

Fig. 12 (a) Conflux in a Series Configuration with 7 Conflux circuits and 8 inputs (In 1 has the highest priority; In 7 and In 8 have the lowest priority.) (b) Conflux in a Parallel Configuration with 7 Conflux circuits and 8 inputs (All equal priority.) "CON" is the label for each Conflux cell.

When using Conflux as a complete readout architecture, there are two obvious configurations, Series and Parallel. These are shown in Fig. 12.

In the Series configuration (Fig. 12.(a)) N Conflux cells are arrayed end-to-end. The output of the i^{th} Conflux output is connected to the #2 input of the $(i+1)^{th}$ Conflux. The output of the Nth (last, rightmost) Conflux is the output of the entire readout architecture. The #1 inputs of each Conflux as well as the #2 input of the first Conflux provide N+1 gateways into the TDM output stream. Since Conflux performs Time Division Multiplexing of the N+1 input streams, the Series configuration effectively acts as a priority encoder giving highest priority to the #1 input of the Nth (last, rightmost) Conflux and lowest priority to #1 and #2 inputs of the first (leftmost) Conflux. Assuming unlimited data input streams on each input, In 8 through In 1, the output stream of Fig. 12(a) over a statistically significant period of time will be 50% In 1, 25% In 2, 12.5% In 3, etc. Inputs on the left also have greater latency due to their lower priority.

In the Parallel configuration (Fig. 12. (b)), 2^N inputs are funneled down to one output through $\log_2(N)$ stages of Conflux. The inputs all have equivalent priority in the Parallel configuration of Fig. 12(b). Assuming unlimited input data streams on each input, In 8 through In 1, the output stream over a statistically significant period of time will contain 12.5% of each of the input data streams and the latencies will be equivalent.

Admixtures of Series and Parallel configurations in which the outputs of Series configurations become inputs to Parallel configurations and vice versa are easily imaginable.

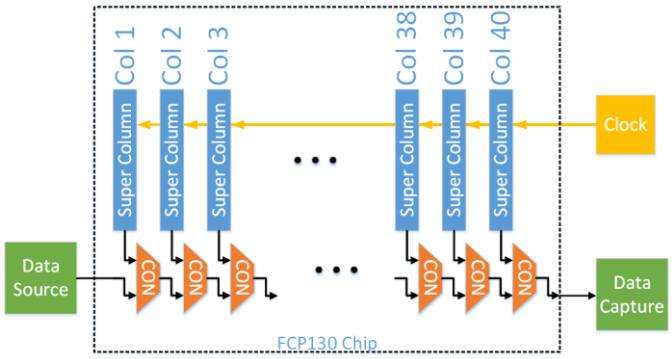

Fig. 13 A sketch of the FCP130 chip showing its Series configuration Conflux readout. "CON" is the label for each Conflux cell.

B. Fabrication

Conflux was first implemented as the readout architecture for FCP130 [19], a test vehicle for research into synchronous pixels for front-end vertex detectors in High-Energy Physics. This is the reason why steps were taken in the design to minimize the possible effects of radiation on performance. FCP130 consists of 40 "super-columns" each consisting of 4 columns of 128 pixels each. Forty Conflux cells, one per super-column, were arranged in a Series readout configuration as shown in Fig. 13. The #2 inputs of the first (leftmost) Conflux as well as the outputs of the 40^{th} (rightmost) Conflux were connected to pads to allow multiple FCP130 chips to output their data in one large aggregate bus system. The Conflux bus width was set at 20 bits with fixed locations for chip address, pixel address, column address, super-column address, and hit magnitude. FCP130 was fabricated in a commercial, low-power 130nm CMOS process.

C. Testing

The test station was designed to examine the function of the FCP130 as a novel pixel front-end architecture. Testing Conflux solely as an asynchronous readout system was thereby prevented. Specifically, the chip was bump bonded to an oversized detector and this blocked access to the leftmost pads of the chip. Unfortunately, these pads are the direct inputs to the #2 input stream of the leftmost Conflux stage. As such the test station itself prevented testing things like Bit Error Rate Tests that might explore the limits of the capabilities of

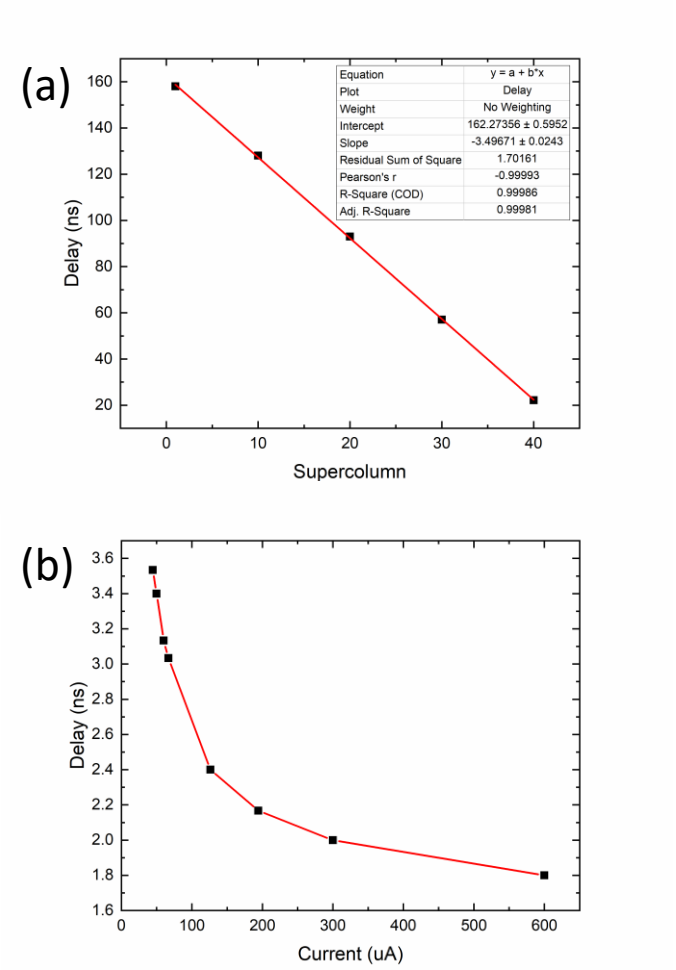

Fig. 14 (a) A linear fit analysis of Conflux propagation delay. The Y-axis is the delay to the output of the final Request To Next. The X-axis is the super-column of the hit. The slope is the individual Conflux propagation delay for this bias current. (b) A plot of the propagation delays of individual Conflux cells as a function of current in the starved inverter cells of the Request Release Logic.

Conflux. Nevertheless, some tests on Conflux were possible.

1) Propagation Delay

For any given super-column under low-luminosity conditions, the following equation defines the propagation delay from the time a hit is transferred into a Conflux cell to the time the final Request To Next signal emerges out of the FCP130 chip:

$$t_{PD} = t_1 + N * t_{Conflux} + t_2$$

where t_1 is the time to transfer data to the Conflux within a super-column, t_2 is the extra time necessary to get the Request To Next from the last Conflux to test equipment, $t_{Conflux}$ is the Propagation Delay of an individual Conflux, and N is the number of Conflux cells from the hit column to the end of the chip. The delays t_1 and t_2 are constants regardless of which pixel in which column is hit.

A single signal simultaneously releases all data from the FCP130 super-columns to the Conflux readout. It is possible to plot of the delay from that signal to the appearance of the Request To Next signal at the output of the FCP130 chip for a single hit pixel as a function of the super-column number that was hit. Fig. 14(a) shows such a plot. The slope of that line is the propagation delay of a single Conflux. That slope must be a function of the bias current sent to the starved inverter delay elements of the Request Release Logic (Fig. 9). That analysis was repeated for a range of bias currents and the slopes are plotted in Fig. 14(b). The minimum achievable propagation delay is approximately1.8ns per Conflux. This corresponds to a operational frequency of 555.6MHz. The 20 bit wide data bus for this implementation of Conflux at this operational frequency yields a bit rate of 11.1Gbps.

2) Data Transfer Integrity and Arbitration Reliability

To test both data and arbitration efficiency, a charge was first injected into Pixel A of super-column 1 and then a second charge was injected into Pixel B of super-column 2 where A and B were arbitrary. This process was then repeated again and again, forcing new bits and new arbitrations through the Series Conflux readout architecture of FCP130. Of course, all 40 Conflux cells performed a new arbitration each time a new event passed through themselves. However only super-column 2 had its arbitration winner change from Input #1 to Input #2 and back. Super-column 1 always found Input #1 to be the winner and Super-columns 3 through 40 always found Input #2 to be the winner.(See Fig. 12(a) and Fig. 13.)

A further limitation to this efficiency test was the speed at which the test could be performed. Again, this was imposed by the decision to focus on FCP130 as a novel front-end architecture. The dual charge injection process of this test could only be repeated at a rate of approximately 115 kHz because of the setup necessary to inject charge into a pixel. Two test periods were ultimately performed. One lasted 44 hours and produced 2.09e9 events without error. The other lasted 66 hours and produced 3.26e9 events without error.

V. Results and Conclusions

As a readout architecture, the Conflux Series readout configuration was very successful for FCP130. It performed without fail throughout the entire single-chip pixel test process. Unfortunately, decisions made during testing limited the types of experiments that could be performed. Bit Error Rate Tests became impossible and the full readout system with multiple FCP130 chips driving their data in series from one FCP130 to another as a giant self-propagating daisy chain could not be realized. This severely limited the number of Conflux-specific tests that could be performed.

Nevertheless, Conflux was highly successful as an asynchronous readout architecture for the FCP130 through months of testing the front-end system. It was very reliable throughout all phases of the testing and it demonstrated a 1.8ns propagation delay.

Appendix A – SR Latches used in Conflux

Some readers may not be familiar with the terms for the various SR Latches used in this paper – e.g Reset Dominant and Hold Dominant. This appendix addresses nomenclature more than anything else.

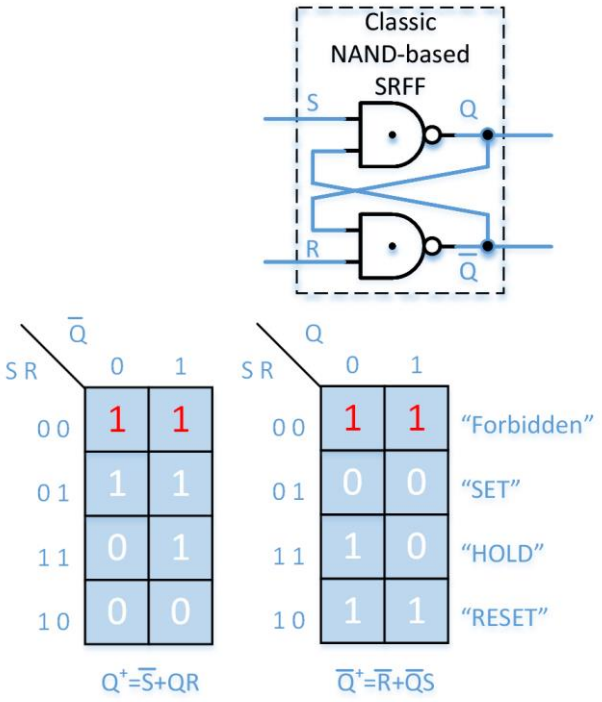

Fig 16 The classic SR Latch and its associated Karnaugh maps. The so-called forbidden state is highlighted in red.

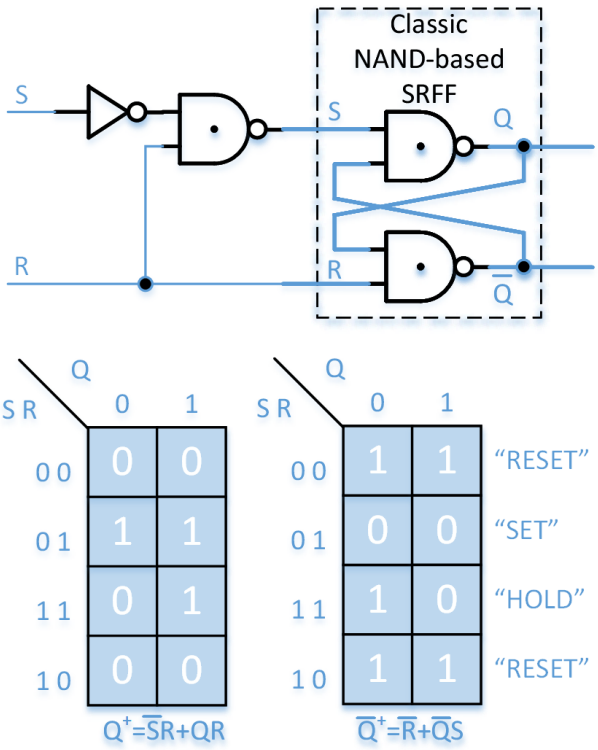

Fig. 15 The Reset-Dominant Latch at its associated Karnaugh maps. Note that the forbidden state has been eliminated and replaced with a second RESET state

A. *The SR Latch*

The SR Latches in Conflux are all constructed from two cross-coupled NAND gates. They could just as easily be constructed from two cross-coupled NOR gates. The surrounding Conflux circuitry would require a very slight adjustment, but the two constructions are functionally equivalent so, for brevity, that construction will not be discussed. The basic cross-coupled NAND gate SR Latch circuit and its associated Karnaugh maps are shown in Fig 16.

If S=0 and R=1, the latch is being "SET" and Q+ will be forced to a 1 and $\overline{Q+}$ will be forced to a 0. If S=1 and R=0, the latch is being "RESET" and Q+ will be forced to a 0 and $\overline{Q+}$ will be forced to a 1. If S=1 and R=1, the latch is in its "HOLD" state and Q+ and $\overline{Q+}$ will maintain their value. Finally, S=0 and R=0 is the "Forbidden" state. It is so called for two reasons. First, while S and R are being held to zero, Q+ and $\overline{Q+}$ are both forced to a 1 and are therefore not inverse to one another. Second, a transition from the Forbidden state to the HOLD state is logically unpredictable [10]. If a SET state is followed by a HOLD state, Q+ and $\overline{Q+}$ will transition to 1 and 0, respectively, during the SET and then retain that value during the HOLD. Similarly, if a RESET state is followed by a HOLD state, Q+ and $\overline{Q+}$ will transition to 0 and 1, respectively, during the RESET and then retain that value during the HOLD. However, in the case of the forbidden state followed by the HOLD state, logic alone is insufficient to determine the final states of Q+ and $\overline{Q+}$.

B. *The Reset-Dominant SR Latch*

A Reset-Dominant SR Latch eliminates the Forbidden state logically by making it a second RESET state. It is shown in Fig. 15.

In this case, the SET, HOLD, and original RESET states remain logically unchanged. However, if S=R=0, the R input gates the S input, presenting S=1 and R=0 (RESET) to a classic SR Latch.

C. *The Set-Dominant SR Latch*

A Set-Dominant SR Latch is not used in Conflux, but it is included here for completeness. The Set-Dominant Latch eliminates the Forbidden state logically by making it a second SET state. Its schematic is analogous to the Reset-Dominant Latch's schematic. However, in the Set-Dominant Latch the S input runs directly to the S input of the Classic NAND-based SR Latch. The R input is inverted and then gated by the S input through a NAND gate.

In the case of the Set-Dominant SR Latch, the original SET, HOLD, and RESET states remain logically unchanged. However, if S=R=0, the S input gates the R input, presenting S=0 and R=1 (SET) to a classic SR Latch.

D. *The Hold-Dominant SR Latch*

A Hold-Dominant Latch eliminates the Forbidden state logically by making it a second HOLD state. It is shown in Fig. 17.

In this case, the SET, original HOLD, and RESET states remain logically unchanged. However, if S=R=0, the S input gates the R input and the R input gates the S input presenting S=1 and R=1 (HOLD) to a classic SR Latch.

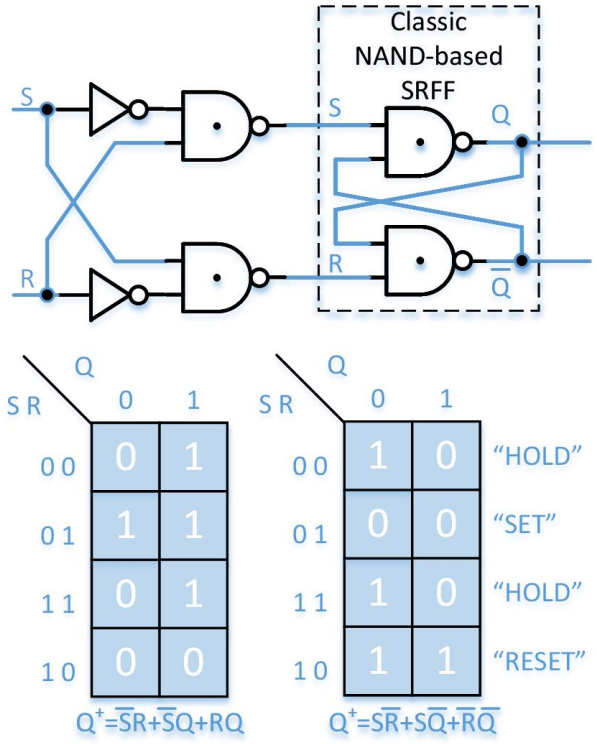

Fig. 17. The Hold-Dominant Latch at its associated Karnaugh maps. Note that the forbidden state has been eliminated and replaced with a second HOLD state.

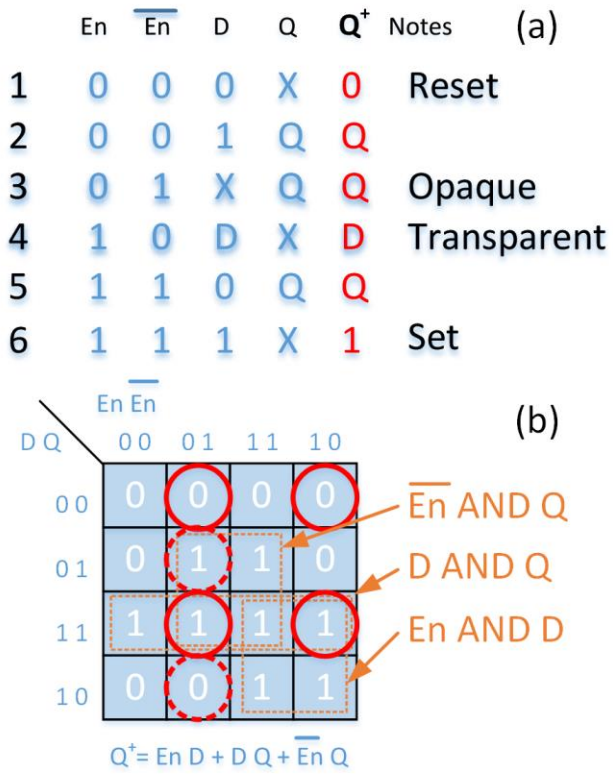

Fig. 19 A truth table (a) and Karnaugh Map (b) for an Earle Latch. The red circles indicate the stable asynchronous states. The orange dashed boxes called out by the arrows and labels indicate the minterms covered by the gates in Fig. 18. The red circles indicate the six stable asynchronous conditions of the latch.

Appendix B – E. The Earle Latch

Conflux implements Earle Latches as its data path. This is not strictly necessary. Conflux will still function with standard D-Latches. However, Conflux implements Earle Latches for the same reason as that for which they were originally developed [18]. Namely, the Earle Latch has a consistent 2-gate input-to-output delay regardless of input, and it is free of asynchronous static hazards. The Earle Latch schematic is shown in Fig. 18

Fig. 19 shows the logic underlying the Earle Latch. The Latch has both a set condition and a reset condition, provided that a user deliberately forces all inputs (including the nominally inverted En and $\overline{En}$) to 0 (reset) or 1 (set). These conditions are not used in Conflux. In Conflux, the Earle Latches only make transitions between their Opaque and Transparent conditions depending on the state of the En signal which is IHaveData.

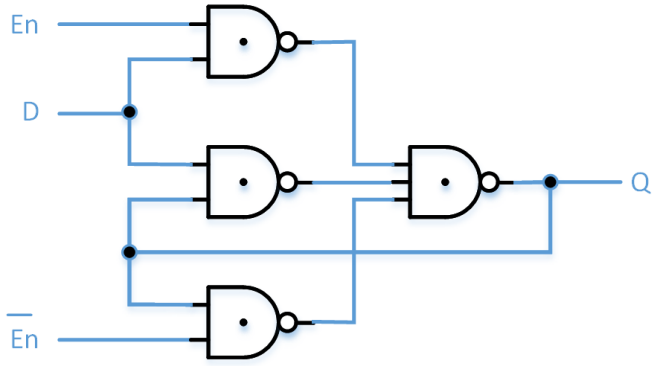

Fig. 18. The Earle Latch

Acknowledgment

The author would like to acknowledge Petra Merkel and the KA-25 research program that sponsored this work. The author would also like to acknowledge Dave Christian, the project manager of FCP130 for his oversight and Farah Fahim, the lead engineer on the FCP130 for her efforts to incorporate Conflux into her chip. The author would also like to acknowledge Lou Dal Monte and Benjamin Bentele for their work on the FCP130 testing from which the Conflux timing and reliability were extracted. The author would also like to acknowledge Grzegorz Deptuch and Marvin Johnson for numerous helpful discussions.

References

[1] J. Christiansen and M. Garcia-Sciveres, "RD Collaboration Proposal: Development of pixel readout integrated circuits for extreme rate and radiation", CERN-LHCC-2013-008, LHCC-P-006 (2013).

[2] CMS Collaboration, "The Phase-2 Upgrade of the CMS Tracker", CERN-LHCC-2017-009, CMS-TDR-014 (2017)

[3] ATLAS Collaboration, "Technical Design Report for the ATLAS Inner Tracker Pixel Detector", CERN-LHCC-2017-021, ATLAS-TDR-030 (2018)

[4] B. Abi et al., *DUNE Collaboration The Single-Phase ProtoDUNE Technical Design Report*, pp. 1-165, July 2017, [online] Available: https://arxiv.org/abs/1706.07081.

[5] F. Carrió and A. Valero on behalf of the ATLAS Tile Calorimeter System, "Clock Distribution and Readout Architecture for the ATLAS Tile

Calorimeter at the HL-LHC", IEEE Trans. Nucl. Sci, 2018 (Early Access) DOI 10.1109/TNS.2018.2885456
[6] R. Yarema, *et al*, "Vertically Integrated Circuit Development at Fermilab for Detectors", 2013 JINST 8 C01052, presented at Topical Workshop on Electronics for Particle Physics, Oxford, Eng., Sept. 2012
[7] J. Hoff, M. Johnson, R. Lipton, and G. Magazzu, "Readout Chip for an L1 Tracking Trigger Using Asynchronous Logic", JINST, vol. 7, July, 2012 [Online] http://ccd.fnal.gov/techpubs/tech-pubs-search.html, Report Number = FERMILAB-CONF-12-411-PPD
[8] J. Hoff, "Time-division multiplexing data bus," U.S. Patent 9,928,202, Mar. 27, 2018.
[9] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design: A Systems Perspective*, Reading, MA: Addison-Wesley, 1993
[10] C. Myers, *Asynchronous Circuit Design*, New York, NY: John Wiley & Sons, 2001
[11] J. Sparso and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective, 2002 edition*, Springer, 2001, pp. 3-8
[12] Steven Nowick and Montek Singh, "Asynchronous Design – Part 1: Overview and Recent Advances, IEEE Design and Test, May/June, 2015, pp. 16
[13] Montek Singh and Steven Nowick, "MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines", IEEE Trans VLSI Sys, Vol. 15, No. 6, June 2007
[14] D. Kinniment and J. Woods, "Synchronisation and Arbitration Circuits in Digital Systems", Proc. IEEE, Vol. 123, No. 10, Oct. 1976, pp. 961- 966
[15] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley: Reading, MA, 1980, pp. 218-261
[16] T. Kacprzak, "Analysis of Oscillatory Metastable Operation of an RS Flip-Flop", IEEE J. Solid-State Circuits, vol 23, no. 1, Feb 1988, pp.260-266
[17] P. Dodd and L. Massengill, "Basic Mechanisms and Modelling of Single-Event Upset in Digital Microelectronics", IEEE Trans Nucl Sci, vol 50, no. 3, June 2003, pp. 583-602
[18] J. Earle, 'Latched Carry-Save Adder", IBM Technical Disclosure Bull., vol. 7, March 1965, pp. 909-910
[19] F. Fahim, "Fermi CMS Pixel (FCP130) test ASIC" [Online] http://ccd.fnal.gov/techpubs/tech-pubs-search.html, Report Number = FERMILAB-CONF-14-435-PPD, 2015

James Hoff was born in New Jersey, USA, in 1965. He received the B.S.E.E. degree in electrical engineering from the University of Notre Dame in 1987, the M.S.E.E. degree in electrical engineering from the Illinois Institute of Technology in 1992 and Ph.D. in electrical engineering from Northwestern University in 1997.

In 1988, he joined Fermilab's Particle Physics Division as an Integrated Circuit designer for the Electrical Engineering Department. His current research interests include digital readout architectures, digital design in extreme environments, and design verification.