Adaptive Priority Toggle Asynchronous Tree Arbiter for AER-based Image Sensor

Aung Myat Thu Linn, Do Anh Tuan, Chen Shoushun, Yeo Kiat Seng
VIRTUS IC Design Centre of Excellence, School of EEE
Nanyang Technological University

***Abstract*—In this paper, we reported an adaptive priority toggle asynchronous tree arbiter for Address Event Representation (AER)-based image sensors. Simultaneous requests from event-triggered pixels, event latency, timing error and jitter are the inherent issues in AER-based read-out circuits. Fixed priority arbiter often results in unfair allocation of bus resource to only "privileged" pixels thus resulting in timing error. The proposed arbiter is able to reduce the timing error by toggling the requests priority during simultaneous requests. This also achieves the fair allocation of bus resource to all pixels. The featured eager propagation scheme allows the requests to propagate towards higher hierarchy in the tree during the arbitration process. As a result, latency and jitter problems can be reduced. Simulation result reveals that single event delay for 128-way tree arbiter is 4.2 ns and therefore, for 128 $\times$ 128 array, such arbiter can process up to 238.09M event/s which is more than 15 times faster than the speed of the reported AER sensor. The arbiter layout was realized with 2P4M 0.35μm CMOS process with a silicon area of 78 x 18μm^2 and had been implemented in 80 $\times$ 80 array AER temporal contrast sensor.**

I. INTRODUCTION

In conventional Digital Signal Processing (DSP)-based vision systems, image brightness information is read out from the pixels at a fixed interval using row and column scanners. Therefore, massive amount of data is produced in high speed sensor. In AER-based CMOS vision sensor [1][2][3][4] [5][6], pixels respond asynchronously to the light intensity changes and requests are sent out to the buffers and AER tree for arbitration. A typical AER imager architecture is presented in Fig. 1.

AER is a protocol originally used for communication between chips. It can be characterized as event driven, or pixel driven protocol when used in image sensors. Instead of reading out the pixel values sequentially one by one, in AER-type of imagers, a pixel sends a request for output once the data is ready. An acknowledgment signal is sent back to the pixel by the chip level AER control circuitry when the request signal is received and the transmission of data is initiated. This method of readout can also be viewed as first-come-first-serve, as the first pixel requesting a readout gains the right to do so as long as the bus is not congested. When multiple requests are sent at the same time, an arbiter decides the sequence of readout. Depending on the type of event detection, one pixel may request access to the tree multiple times. Fixed priority often results in an unfair allocation of the output bus to only "privileged" pixels thus resulting in an unbalanced timing error, i.e., for pixels with higher priority, the timing error is small and for pixels with lower priority, the timing error is large. Improving the arbitration speed also helps to reduce the timing error [2].

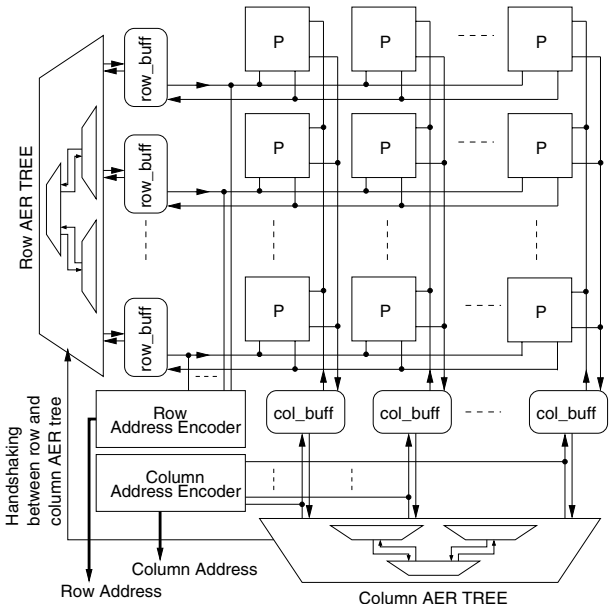

Fig. 1. A typical architecture of Address Event Representation (AER)-based read-out image sensor

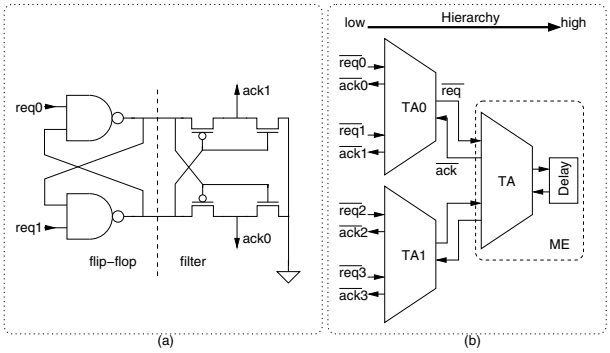

Fig. 2. (a) 2-way basic arbiter, (b) Realization of 4-way tree using 3 tree arbiter (TA)

A simple arbiter can be realized with a flip-flop and a mask or a filter to conceal its meta-state. Fig. 2(a) represents a simple two-way arbiter or a mutual exclusion element (ME) [7]. It operates based on the request-grant-release-acknowledge

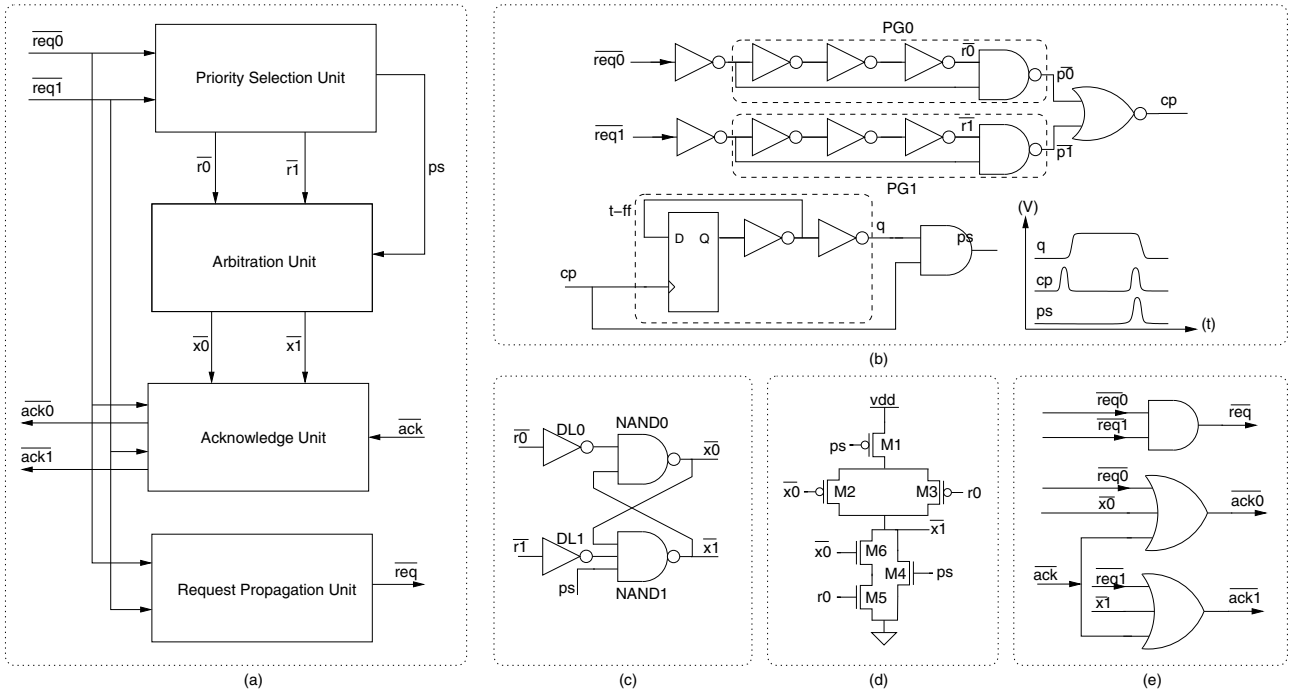

Fig. 3. (a)The tree arbiter architecture; (b) Priority selection unit; (c) Arbitration unit; (d) Transistor level design of NAND1 (e) Request propagation unit and acknowledgement unit.

protocol (req goes high, ack goes high, req goes low, ack goes low). The other kind of arbiter using the same protocol is the tree arbiter element (TA) shown in Fig. 2(b). It has additional $\overline{req}$ and $\overline{ack}$ terminals and it can be attached to an N-way arbiter to form an $N+1$ arbiter [8]. This will require the request signals to propagate though multiple hierarchy levels and the acknowledgement signal issued by the ME to propagate back to lowest level of the hierarchy. As the depth of the tree increases, the corresponding delay also increases.

In 2000, [9] introduced the tree arbiter element for spiking pixels with fixed priority scheme. The effect of fixed priority tree arbiter element on time-to-first spike AER sensor was highlighted in [2] and it proved by simulation that fixed priority often results in unfair allocation of output bus to half of the pixels. Moreover, also in [2], the author developed the priority toggling arbiter element to over come the problem. However, design only toggles the priority conditionally as witnessed in Fig 5(b). In this paper, we proposed the priority toggle asynchronous arbiter featured with an eager propagation of requests to overcome the problems mentioned above. The tree arbiter element has capability to detect the simultaneous requests and dynamically toggles the priority of requests. The featured eager propagation of request allows the request propagation to higher hierarchy during arbitration to minimize the delay time. The rest of the paper is organized as follows: In the next section, we discuss the detail architecture of the tree-arbiter element and its individual components. After that, we review the simulation results and discuss the implementation of the design. Conclusions are drawn in Section IV.

II. Architecture and Operation Principle

In this section, we introduce the operation principle of the arbiter and its architecture. The tree arbiter element consists of 4 basic units, namely: priority selection unit, arbitration unit, request propagation unit and acknowledgement unit as shown in Fig. 3(a). The priority selection unit is responsible for toggling the priority of the arbitration unit when simultaneous request is detected. The request propagation unit propagates the requests to the higher hierarchy level and the acknowledgement unit sends an acknowledgement back to lower level. Detailed structures of these building blocks are as follows:

A. Priority Selection Unit

The priority selection unit (Fig. 3(b)) is responsible for detecting the simultaneous request and toggles the priority of the arbitration unit by sending out the priority signal, ps pulse. In Fig. 3(b) the two pulse generators (PG0 and PG1) generate two pulses, namely $\overline{p0}$ and $\overline{p1}$ corresponding to $\overline{req0}$ and $\overline{req1}$ respectively. In case of request signals collision or simultaneous requests, the timings of $\overline{p0}$ and $\overline{p1}$ pulses are identical and these pulses produce "collision pulse, cp" through the OR-gate. After that, cp is input into t-ff and AND gate as shown in Fig. 3(b). The signal, q toggles at every cp pulse due to t-ff. The first cp pulse toggles q from low to high. The second cp pulse again toggles q from high to low; however, due to gate delay in t-ff, q goes low only after cp pulse. Such two signals overlapping (highlighted in Fig. 4) produces another pulse, ps through the NAND gate. Therefore, for every two cp pulse, one ps pulse is generated.

B. Arbitration Unit

The arbitration unit consists of two cross-coupled NAND gates (SR-latch), as shown in Fig. 3(c). $\overline{r0}$ and $\overline{r1}$ are delayed $\overline{req0}$ and $\overline{req1}$ respectively. This block is responsible for assigning the priority of the request signal by deciding which one comes first. In the case of concurrent request arrival (i.e. collision), the arbitration unit will assign the priority to one of the request signals (e.g. $\overline{req0}$) and in the next collision, to the other (e.g. $\overline{req1}$). This priority toggling is realized as follows: NAND0 is biased in such a way that $\overline{x0}$ has a higher priority in any circumstances. Such biasing allows it to resolve the meta-state quickly. Additional input signal ps from the priority selection unit is to toggle the priority of $\overline{x0}$ to $\overline{x1}$. With ps signal active, $\overline{x1}$ is purposely pulled down by M4 and at the same time M1 is off to avoid short circuit current (as shown in Fig. 3(d)). $\overline{x1}$ is consequently given higher priority compared to $\overline{x0}$. The inverted delay elements (DL0 and DL1) are carefully designed so that ps signal arrives just before and during the meta-state time window of the latch. Otherwise, glitches happen in $\overline{x0}$ and $\overline{x1}$.

C. Request Propagation Unit and Acknowledgement Unit

The request propagation unit conveys the request $\overline{req}$ to the higher hierarchy through the AND gate while the arbitration unit is still processing. The acknowledgement unit consists of two 3OR gates for $\overline{ack0}$ and $\overline{ack1}$ as shown in Fig. 3(e). For example, this unit issues $\overline{ack0}$ providing that $\overline{x0}$, $\overline{ack}$ and $\overline{req0}$ are low. The OR gates in the acknowledgement unit also serve as a filter to conceal the meta-state of $\overline{x0}$ and $\overline{x1}$.

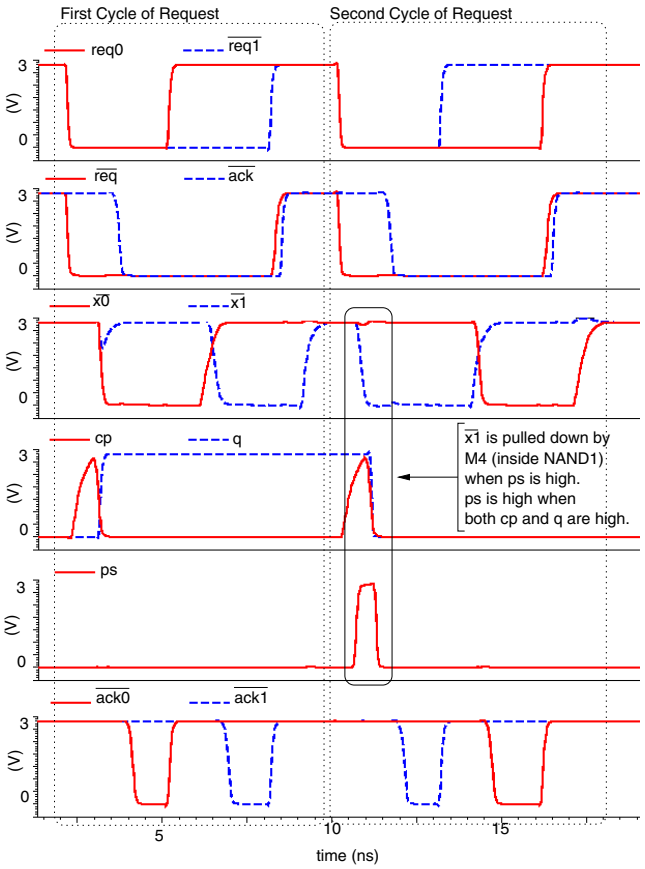

Fig. 4. The simulated waveform showing the intermediate signal of the tree arbiter such as $\overline{x0}$, $\overline{x1}$, cp, q and ps. The two simultaneous requests cycle is simulated and the priority is successfully toggled during the second cycle.

III. Simulation and Implementation

This section describes the simulation results of the priority toggling and demonstrates the feasibility of the design, focusing on the meta-stability of the arbitration unit and the corner simulation to ensure a robust performance. After that, delay and speed performances of the arbiter tree are analyzed based on the simulation data. All simulations were carried out in a 0.35 μm CMOS process using Cadence Spectre simulator.

A. Priority Toggling

Two cycles of simultaneous requests were applied to tree arbiter element and the simulation result is shown in Fig. 4. During the first cycle, the simultaneous requests $\overline{req0}$ and $\overline{req1}$ cause the SR-latch to enter the meta-stable state. At the same time, request signal $\overline{req}$ is sent out to higher hierarchy. At this cycle, cp causes q to toggle from low to high and ps is still low. As the NAND0 is biased, eventually, $\overline{x0}$ wins the competition and its voltage is pulled down to ground. By the time $\overline{ack}$ is low, $\overline{ack0}$ propagates back to the process. After that, process must kill the $\overline{req0}$ to allow the $\overline{req1}$ to propagate forward and later, $\overline{ack1}$ is propagated back when $\overline{x1}$ is low. During the second simultaneous requests, ps pulse is generated due to the overlapping of cp and q and $\overline{x1}$ wins the competition due to configuration of NAND1 as mentioned in SECTION II-B. As a result, $\overline{ack1}$ is issued first followed by $\overline{ack0}$. In short, during the first collision, $\overline{req0}$ wins the competition and its corresponding acknowledge signal, $\overline{ack0}$ is processed first while during the second collision, $\overline{req1}$ wins by acknowledging $\overline{ack1}$ first. This fact can be clearly seen by looking at the bottom graph in Fig. 4.

Fig. 5 reports the priority toggling simulation results of three arbitration schemes. One can note that the conventional design uses a fixed priority so that $\overline{req0}$ is always processed earlier than $\overline{req1}$. The design [2] toggles the priority only if it received non-simultaneous request between two simultaneous request and this was demonstrated in Fig. 5(b). As proven in Fig. 5(c), our proposed design unconditionally toggles the requests priority at every simultaneous request, thanks to the priority selection unit which detect the signal collision and twists the input priority of the arbitration unit.

Simulation had also been done for two 4-way trees structure with and without priority toggle and the results are shown in Table. I. During the first cycle of simultaneous requests, both trees response in same manner by acknowledging $\overline{ack0}$, $\overline{ack1}$, $\overline{ack2}$ and $\overline{ack3}$ in sequence. At the second cycle of simultaneous requests, the later responses in the same way as in first cycle of requests while our design acknowledges in a reverse sequence, i.e. $\overline{ack3}$, $\overline{ack2}$, $\overline{ack1}$ and $\overline{ack0}$. Therefore, the proposed design provides an equal chance to all requests during collision.

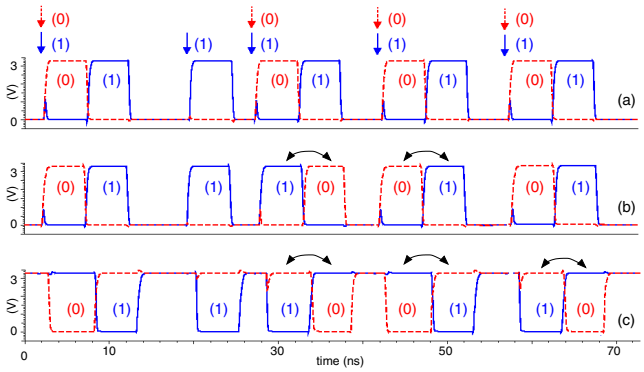

Fig. 5. Simulation result showing the priority toggling capabilities of three arbiters. The double arrow represents two simultaneous requests, i.e. collision; (a) conventional design - fixed priority, (b) design in [2] - fair priority in a special case (c) proposed design - fair priority in any cases.

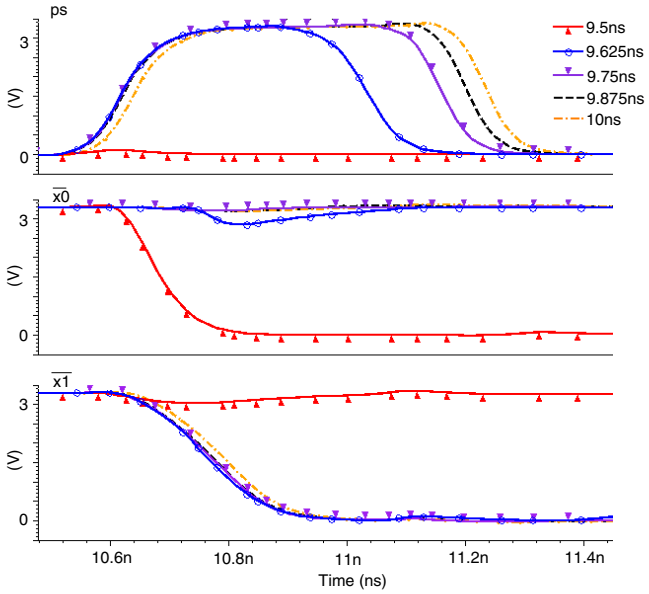

Fig. 6. $\overline{req1}$ was low at 10ns and the low point of $\overline{req0}$ was swept from 9.5ns to 10ns in 5 steps. When the two request is closed enough which results in production of ps pulse from priority selection and subsequently pulls down the $x1$. Therefore, no meta-stability had been observed.

B. Design feasibility

In this section, behavior of ($\overline{x0}$ and $\overline{x1}$) signal was observed during the time of priority toggle by ps. The request time of $\overline{req0}$ was swept from 9.5ns to 10.0ns while that of $\overline{req1}$ was kept at 10ns. Fig. 6 shows that at 9.5ns request time of $\overline{req0}$, $\overline{x0}$ goes low first without any meta-state as $\overline{x0}$ requests earlier than $\overline{req1}$. At request time 9.625ns onward, $\overline{x1}$ is already pull down by $M4$ providing that ps pulse is long enough until just

TABLE I
COMPARISON OF ACKNOWLEDGEMENT SEQUENCES BETWEEN 4-WAY CONVENTIONAL TA AND 4-WAY PROPOSED TA

	First Cycle	Second Cycle
Conventional Arbiter	$\overline{ack}0, 1, 2, 3$	$ack0, 1, 2, 3$
This Work	$\overline{ack}0, 1, 2, 3$	$ack3, 2, 1, 0$

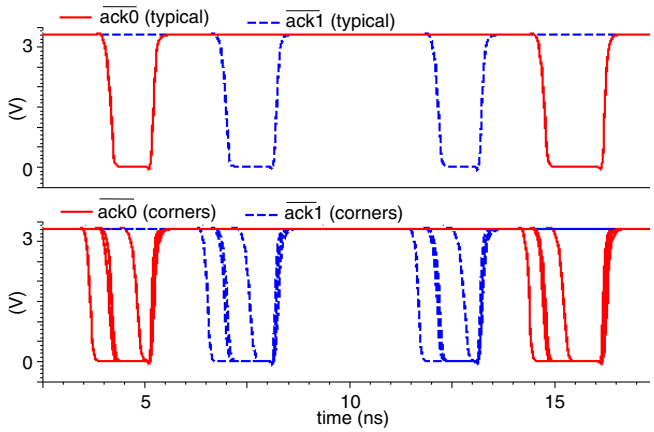

Fig. 7. Priority is given to $\overline{ack0}$ in first request cycle and it is toggled to $\overline{ack1}$ in second cycle. The corner simulation verifies the stability of priority toggling operation.

after the point where SR-latch enters the meta-state. This is to make sure that $\overline{x1}$ is successfully pulled down without letting SR-Latch to enter meta-state. Therefore, arbitration time is cut off to minimum. A glitch appears in $\overline{x1}$ when the request time of $\overline{req0}$ is at 9.625ns. However, this glitch does not affect the $\overline{ack0}$ as this signal has to wait until $\overline{ack}$ is issued by higher hierarchy. Therefore, such a delay allows $\overline{x0}$ and $\overline{x1}$ to stabilize before acknowledgement is sent back to the requested process.

Corners simulation had also been carried out to make sure that logical operation of the proposed TA works at any condition. Fig. 7 shows the simulation results at different corners. It can be seen that the priority of the requests is successfully toggled in second simultaneous request cycle without any glitches occurring in the acknowledgement signals at any process corners. This conclusively proves that the proposed circuits can cope with process variations and produce the desirable results.

C. Delay and Speed of the Tree

Here we discussed the timing delay of our 128-way tree structure. In particular, two analysis had been done; single request-to-acknowledgement delay and 128 simultaneous requests-to-acknowledgements delay. Fig. 8 shows the flow of the signal in the arbiter tree. As the 128-ways arbiter tree has 7 hierarchy levels, single request-acknowledgement delay can be represented as:

$$T_{single} = 7t_{req} + 6t_{ack} + t_{ack-ME}$$

where t_{req} is the propagation delay of request to higher hierarchy; t_{ack-ME} is the propagation delay of acknowledgement to lower hierarchy in ME; t_{ack} is the propagation delay of acknowledgement to lower hierarchy. t_{ack-ME}, unlike t_{ack}, is longer due to the contribution of delay in priority selection unit and arbitration unit. Based on simulation, it is worth mentioning that $t_{req} = 0.2\ ns$, $t_{ack} = 0.3\ ns$ and $t_{ack-ME} = 1\ ns$, $T_{single} = 4.2\ ns$.

As a result, the delay for 128 simultaneous requests-to-acknowledgements can be expressed as:

$$T_{totol} = 7t_{req} + 6t_{ack} + t_{ack-ME}$$

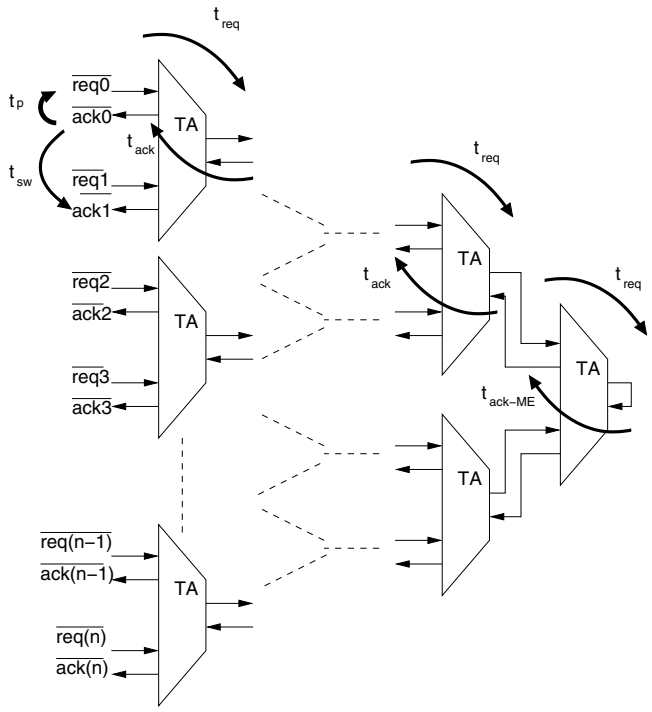

Fig. 8. This figure shows the delay paths encountered in arbiter tree.

$$
\begin{aligned}
&+ (3t_p + 3t_{sw} + 4t_{ack} + t_{req}) \times 32 \\
&+ (2t_{req} + t_{sw} + 3t_{ack} + t_p) \times 16 \\
&+ (3t_{req} + t_{sw} + 4t_{ack} + t_p) \times 8 \\
&+ (4t_{req} + t_{sw} + 5t_{ack} + t_p) \times 4 \\
&+ (5t_{req} + t_{sw} + 6t_{ack} + t_p) \times 2 \\
&+ (6t_{req} + t_{sw} + 7t_{ack} + t_p) \\
&= 127t_{req} + 253t_{ack} + + t_{ack-ME} + 127t_{sw} + 127t_p
\end{aligned}
$$

where: t_{sw} is the switching delay from one acknowledgement to next within one particular TA; t_p is the delay in each processor to retrieve its request or processing time of each process. Additional assumption in this equation is that every process possesses the same t_p. t_p is also depending on the application itself and in this analysis, it is assumed to be zero. After solving the above equation, T_{totol} is equal to $0.293\mu s$. T_{totol} is divided by 128 to get the single request-acknowledgement delay. Therefore, each event delay is about 2.29 ns which is much lower than the previous result, 4.2 ns. This is because not all the requests are required to propagate through all the hierarchy so that average single event delay is reduced. In order to calculate the speed of the tree, only worst case scenario was considered and the resulting speed is 238.09M event/s which is about 15 times faster than the speed reported in [3].

D. VLSI Implementation

The tree arbiter layout was realized in a 2P4M $0.35\mu m$ CMOS process with a silicon area of 78 x 18 μm^2 as shown in Fig.9. An AER-based image sensor which includes an 80×80 pixels array had also been implemented to test the functionality of the arbiter tree. Pre- and post-layout simulations have confirmed that the proposed arbiter tree

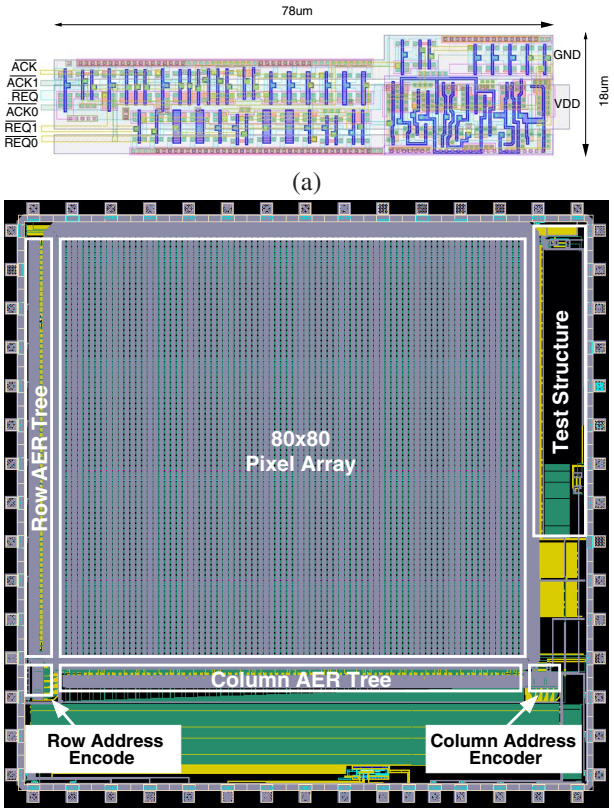

Fig. 9. (a)arbiter cell layout (b) layout of an AER sensor which includes an 80×80 pixel array and the proposed AER tree.

communicates smoothly with the pixel arrays as well as the other external I/O blocks with similar power consumption and delay as mentioned above. The total chip area is 5x5 μm^2. This chip has been sent out for fabrication.

IV. Conclusion

The innovative part of this work lies in the fair allocation of output bandwidth to the requests in AER type readout sensors. The proposed tree arbiter is able to toggle or alter the acknowledge sequence in order to allocate the bandwidth resource fairly to all pixels. One add up advantage to this design is its ability to avoid the meta-stability at the every two concurrent requests. Moreover, extensive simulations verify the correct operation of the arbiter at any corner condition. The two (row and column) 128-way AER trees can reach 238.09M event/s readout speed and this enables the AER sensor to process high speed motion detection, object tracking with little latency, timing error and jitter.

Acknowledgment

This work was supported by Nanyang Assistant Professorship (M58040012) and ACRF Project (M52040132).

References

[1] E. Culurciello, et al., "A biomorphic digital image sensor," Solid-State Circuits, IEEE Journal of, vol. 38, pp. 281-294, 2003.

[2] S. Chen and A. Bermak, "Arbitrated time-to-first spike CMOS image sensor with on-chip histogram equalization," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 15, pp. 346-357, 2007.

[3] P. Lichtsteiner, et al., "A 128×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," IEEE Journal of Solid-State Circuits, vol. 43, pp. 566-76, 2008.

[4] N. Massari, et al., "A 100μW 64×128-Pixel Contrast-Based Asynchronous Binary Vision Sensor for Wireless Sensor Networks," in IEEE ISSCC Dig. of Tech. Papers, 2008, pp. 588- 638.

[5] P. Lichtsteiner, et al., "A 128×128 120dB 30mW Asynchronous Vision Sensor that Responds to Relative Intensity Change," in ISSCC Dig. of Tech. Papers, San Francisco, 2006, pp. 508-509

[6] S. Chen and A. Bermak, "A second generation time-to-first-spike pixel with asynchronous self power-off," in Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on, 2006, pp. 4 pp.-2292.

[7] C. L. Seitz, "Ideas about arbiters," Lambda, vol. 1, no. 1, pp. 10-12, 1980.

[8] M. B. Josephs and J. T. Yantchev, "CMOS design of the tree arbiter element," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 4, pp. 472-476, 1996.

[9] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, vol. 47, pp. 416-434, 2000.