On Multiple AER Handshaking Channels Over High-Speed Bit-Serial Bidirectional LVDS Links With Flow-Control and Clock-Correction on Commercial FPGAs for Scalable Neuromorphic Systems

Amirreza Yousefzadeh, Mirosław Jabłoński, Taras Iakymchuk, Alejandro Linares-Barranco, Alfredo Rosado, Luis A. Plana, Steve Temple, Teresa Serrano-Gotarredona, Steve B. Furber, *Fellow, IEEE*, and Bernabé Linares-Barranco, *Fellow, IEEE*

***Abstract*—Address event representation (AER) is a widely employed asynchronous technique for interchanging "neural spikes" between different hardware elements in neuromorphic systems. Each neuron or cell in a chip or a system is assigned an address (or ID), which is typically communicated through a high-speed digital bus, thus time-multiplexing a high number of neural connections. Conventional AER links use parallel physical wires together with a pair of handshaking signals (request and acknowledge). In this paper, we present a fully serial implementation using bidirectional SATA connectors with a pair of low-voltage differential signaling (LVDS) wires for each direction. The proposed implementation can multiplex a number of conventional parallel AER links for each physical LVDS connection. It uses flow control, clock correction, and byte alignment techniques to transmit 32-bit address events reliably over multiplexed serial connections. The setup has been tested using commercial Spartan6 FPGAs attaining a maximum event transmission speed of 75 Meps (Mega events per second) for 32-bit events at a line rate of 3.0 Gbps. Full HDL codes (vhdl/verilog) and example demonstration codes for the SpiNNaker platform will be made available.**

***Index Terms*—Address event representation (AER), event-driven system, neuromorphic engineering, scalable neuromorphic systems, spiking systems, virtual wiring.**

Manuscript received June 27, 2016; revised April 20, 2017; accepted June 4, 2017. Date of publication August 14, 2017; date of current version September 25, 2017. This work was supported in part by the European Research Council under the European Union.s Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement 320689, in part by the U.K. Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/G015740/1, in part by EU FP7 Grant 604102 "The Human Brain Project" (HBP), in part by EU H2020 under Grants 644096 "ECOMODE" and 687299 "NEURAM3," in part by Polish grant from the Ministry of Science and Higher Education AGH UST no. 11.11.120.612, in part by Spanish grants from the Ministry of Economy and Competitivity TEC2012-37868-C04-01/02 (BIOSENSE), TEC2015-63884-C2-1-P (COGNET), and TEC2016-77785-P (COFNET) (with support from the European Regional Development Fund), and in part by Andalusian grants TIC-6091 (NANO-NEURO) and P12-TIC-1300 (MINERVA). The work of A. Yousefzadeh was supported by a Spanish FPI Scholarship from the Ministry of Economy and Competitivity. This paper was recommended by Associate Editor G. Cauwenberghs. *(Corresponding author: Bernabe Linares-Barranco.)*

A. Yousefzadeh, T. Serrano-Gotarredona, and B. Linares-Barranco are with the Instituto de Microelectrnica de Sevilla, IMSE-CNM (CSIC and University of Sevilla), Sevilla 41092, Spain (e-mail: reza@imse-cnm.csic.esl; terese@imse-cnm.csic.es; bernabe@imse-cnm.csic.es).

M. Jabłoński is with the Instituto de Microelectrnica de Sevilla, IMSE-CNM (CSIC and University of Sevilla), Sevilla 41092, Spain, and also with the AGH University of Science and Technology, Kraków 30-059, Poland (e-mail: mjk@agh.edu.pl).

T. Iakymchuk and A. Rosado are with the School of Engineering, University of Valencia, Valéncia 46010, Spain (e-mail: Taras.Yakymchuk@uv.es; Alfredo.Rosado@uv.es).

A. Linares-Barranco is with the Department of Computer Architectures, University of Sevilla, Sevilla 41004, Spain (e-mail: alinares@atc.us.es).

L. A. Plana, S. Temple, and S. B. Furber are with the School of Computer Science, University of Manchester, Manchester M13 9PL, U.K. (e-mail: luis.plana@manchester.ac.uk; steven.temple@manchester.ac.uk; steve.furber@manchester.ac.uk).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TBCAS.2017.2717341

I. Introduction

ADDRESS Event Representation (AER) is now a fairly popular "virtual wiring" technique adopted by many neuromorphic hardware engineers to interconnect spiking neuromorphic systems [1]–[12]. Digital inter-chip communication is several orders of magnitude faster than firing frequencies of biological neurons. This is exploited in AER to time-multiplex numerous synaptic connections between neurons on a high-speed digital bus. In AER, whenever a spiking neuron in a chip (or module) generates a spike, its "address" (or any given ID) is written on a high speed digital bus and sent to the receiving neurons in one or more destination modules/chips. AER started out as a point-to-point protocol for interconnecting neurons from one chip with those on another chip using a hand-shaken parallel digital bus with a fixed number of bits [1]–[3]. As neuromorphic systems have been scaling up in size and complexity over the years, researchers have developed more complex and smarter AER "variations" to improve efficiency, reconfigurability and reliability. It became apparent very early on that bulkiness of the original parallel-AER (pAER) bus would limit the scalability of AER systems to arbitrary sizes, and researchers started looking at serial connectivity options [13]. In 2004 Boahen proposed a word-serial AER link to reduce the number of parallel wires

[14]. The 48-chip SpiNNaker PCB [15] uses six bidirectional 8-wire 4-bit word serial AER-type links per chip, employing a highly power-efficient delay-insensitive 2-of-7 NRZ protocol [16] to asynchronously interchange 32-bit events between the chips, each chip holding 18 ARM968 integer-arithmetic cores.

Fully bit-serial Low-Voltage-Differential-Signaling (LVDS) [17] AER links have also attracted the attention of some neuromorphic engineers, as they allow for multi-gigabit-per-second communication speeds using only one pair of wires. Since only two uni-directional differential wires are available, it is not straightforward how to implement a handshaking protocol per event transmission, or a flow-control scheme to signal data congestion on the receiver side. Miro *et al.* [18] and Berge *et al.* [19] experimented with fully bit-serial links. The former with off-the-shelf MAXIM ser/des components, showing measurements that revealed the links could operate up to 40 Meps (Mega-events-per-second) for 16-bit AEs (Address Events). The latter with the 2.5 Gbps (Giga-bit-per-second) LVDS Rocket-IO IP blocks available in the VirtexII-Pro Xilinx FPGAs, achieving 41.66 Meps for 16-bit AEs, with 8b/10b encoding for byte-alignment comma transmission when the channel is idle. However, in both cases, the FSMs (Finite State Machines) implemented did not include any hand-shaking or flow-control mechanisms to avoid data loss if the event consumer system on the receiver side was temporarily slower than the event generator system on the transmitter side. Fasnacht *et al.* [20] reported a bit-serial interface based on off-the-shelf commercial 16-bit Serializer/Deserializer components (TLK 2501/3101) connected to a Spartan3E parallel event AER processor and using a second, reverse, LVDS link for flow control signaling. The bit-serial link could operate at line speeds of 2.5 Gbps (TLK 2501) or 3.125 Gbps (TLK 3101) and also used 8b/10b encoding to allow for idle commas. On the reverse LVDS link the receiver put a square wave whose frequency signaled whether to stop or resume event transmission from the sender. Using this setup, the link could transmit 32-bit events at a maximum rate of 62.5 Meps (for 2.5 Gbps) or 78.125 Meps (for 3.125 Gbps), at the cost of sacrificing one reverse LVDS link for flow control. Zamarreo *et al.* [21] developed a bi-directional LVDS link using Virtex6 FPGA Rocket-I/O IPs. In this scheme, two 2.5 Gbps LVDS links were used to communicate 32-bit events in each direction with 8b/10b encoding. Flow-control in each direction was implemented by inserting special control symbols in the opposite direction link to toggle between the stop and resume states. However, this design did not include any clock-correction support, thus hampering scalability by limiting this approach to situations in which all FPGAs use the same physical reference clock.

Fully ASIC designs for 32-bit event transmission over 1 GHz bandwidth LVDS links have also been reported recently using either current-mode [22] or voltage-mode [23] drivers.[1] These use 4 wires: two for high-speed 1GHz bit-serial LVDS data transmission and two for hand-shaking. The extra hand-shake wires allow the drivers to be turned ON/OFF during inter-event pauses (with nano-second latencies), resulting in power consumption proportional to the information transmission rate. This way, no idle commas have to be transmitted and 8b/10b encoding is not necessary. On the down-side, Manchester encoding is necessary to allow for simultaneous data and clock transmission per symbol [24], reducing effective data transmission to half of maximum physical rate. Nevertheless, maximum 32-bit event rates of up to 15 Meps have been measured over a channel with a physical bandwidth 1.4GHz.

All previous asynchronous pAER to bit-serial AER conversion schemes operate correctly if there is only one clock domain at each synchronous side of each link. This is usually the case in fully ASIC designs, where all AER processing is fully asynchronous and there is only one local clock per LVDS Transmitter/Receiver link located at the transmitter side (the receiver uses the clock extracted from the transmission). In the case of the reported FPGA interfaces with unidirectional links, the situation is similar, because normally the FPGA synchronous circuit only implements one state machine to interface with an asynchronous hand-shaken pAER port. One single, isolated clock domain can therefore be used per link. In the case of bi-directional links, the situation is not so straightforward when using commercial FPGAs with bit-serial IPs, and two interfering clock domains might be required. The problem becomes even worse when interconnecting multiple FPGAs, each with its own synchronous event processing subsystem and multiple bidirectional LVDS links per FPGA. In this case, each FPGA will have one or more local clock domains, which may interfere with the clock domains of neighboring FPGAs. Under these circumstances it is necessary to use some kind of clock correction technique to compensate for clock frequency/phase drifts and avoid sudden byte misalignment problems and data loss. In a preliminary work [25], we used the elastic buffers available in Xilinx Rocket-I/O serial LVDS IPs, although the test involved a fully synchronous handshake-less system deployed over two FPGAs. Each link was unidirectional since the reverse LVDS path was fully occupied handling flow-control. More recently, the SpiNNaker team has developed bidirectional bit-serial LVDS links [26] to bundle eight 2-of-7 AER multi-symbol inter-SpiNNaker-chip links [16] into one bit-serial SATA link. This scheme, developed to interconnect multiple (up to 1200) 48-chip SpiNNaker Boards [27], uses flow-control, clock correction, and a complex framing protocol that samples the eight channels and performs CRCs (Cyclic Redundancy Checks) to improve reliability. However, this introduces extra overheads, theoretically limiting the maximum throughput for each of the eight channels to $50/8 = 6.25$ Meps.

In this paper we present an extended version of the solution reported earlier [25], with a fully bidirectional bit-serial LVDS communication capability, a token-based flow-control protocol, a clock correction capability, and a robust interface with conventional parallel AER ports (like those used in AER sensor chips) using 4-phase asynchronous handshaking. At an LVDS line transmission rate of 3.0 Gbps, it is possible to achieve 32-bit transmission at a sustained rate of 75.0 Meps, as shown later in the Experimental Results section. Fig. 1 shows an example target setup consisting of an array of 40 AER-Node Boards

[1]Although the CMOS process used was quite old with a large minimum feature size (0.35 μm), the drivers achieved an impressive performance of up to 1.4 GHz bandwidth.

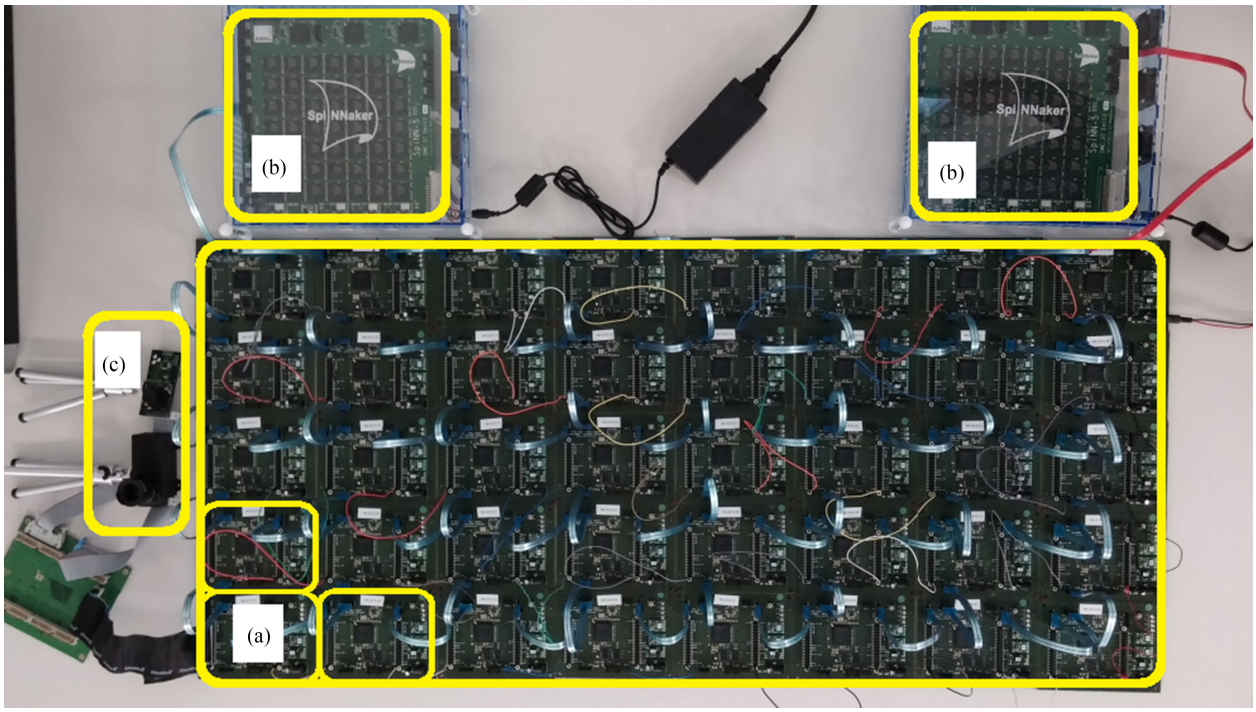


Fig. 1. Example heterogeneous neuromorphic sensing and computing system consisting of (a) 5 × 8 grid of FPGA-based AER-Node boards connected to each other via four bi-directional LVDS links, (b) two SpiNNaker boards comprising each 48 SpiNNaker chips (each with 18 ARM CPUs) organised into grids connected via 2-of-7 asynchronous parallel buses with 2-phase handshaking, and (c) two asynchronous artificial retina vision sensors connected via parallel AER 16-bit external buses with 4-phase handshaking.

[25] all interconnected via SATA wires to their neighbours, to two 48-chip SpiNNaker Boards [27] and to two AER retina sensors [28].

The presented serial intercommunication protocol extensively exploits the Spartan6 GTP transceiver instance using specific configurations together with user designed TX and RX blocks for low bandwidth overhead intercommunication of heterogeneous components (multiple clock-domain synchronous modules with fully asynchronous sensors), while multiplexing multiple AER channels with independent flow control with minimum latency and almost maximum physical channel throughput. This results in very efficient assembly capability of heterogeneous neuromorphic systems. Application examples are illustrated at the end of the paper.

The paper is structured as follows. Section II explains how we used the 8b/10b encoding scheme for bi-directional token-based flow-control and 32-bit event alignment. Section III examines in detail the problem of multiple interfering clock domains when using multiple FPGAs each with multiple bi-directional links, and how to overcome it using elastic buffers and clock-correction. Finally, Section IV provides experimental results.

II. Bi-Directional Token-Based Flow-Control and Event-Alignment Using 8b/10b Encoding

When transmitting GHz range bit-serial data over a single lane (either differentially over a pair of wires as in LVDS, or using one single wire), the transmitted stream must include not only the serial data itself but also sufficient clues to allow clock recovery at the receiver.[2] The Manchester encoding scheme [29] encodes bits '0' and '1' as two different transitions: from high-to-low or from low-to-high. This way, for each symbol (either '0' or '1') there is always a physical transition that makes it possible to recover the clock instantly during symbol extraction. The drawback is that the data rate is only half the channel's maximum possible physical rate.

8b/10b encoding [30] overcomes this drastic limitation by mapping 8-bit words to 10-bit symbols, assuring enough state changes for reasonable clock recovery, while achieving DC balance. The difference between the counts of '1 s' and '0 s' (called "disparity") in a string of at least 20 bits is no more than two and there are not more than five '1 s' or '0 s' in a row.[3] Consequently, this scheme allows effective data transmission at a rate of 80% of the physical channel bandwidth. Clock recovery from the bit stream is not instantaneous and is normally performed by complex PLL (phase locked loop) circuits at the receiver, which may require sequences in the order of thousands of bits to lock to the clock frequency. Also, in order to keep the PLLs locked all the time, the channel

[2]In the case of multiple bit-serial lanes operating in parallel (which is not the case considered in this paper), one lane can be dedicated to transmitting the reference clock. In this case, the lanes have to be very well matched to avoid excessive skew.

[3]Since shuffling five '0 s' and five '1 s' (perfectly DC balanced 10-bit symbols) yields less than 256 10-bit symbols, some 8-bit symbols are mapped into two unbalanced 10-bit symbols (one with four '0 s' and six '1 s' and the other with six '0 s' and four '1 s'). A counter keeps track of the "running disparity" (RD) between '1 s' and '0 s' and picks one of the two possible 10-bit symbols so that the RD is compensated.

TABLE I
CONTROL COMMAS IN 8B/10B CODING

	8-bit in			10-bit out	
Name	DEC	HEX	BIN	RD = − 1	RD = +1
K28.0	28	1C	000 11100	001111 0100	110000 1011
K28.1	60	3C	001 11100	001111 1001	110000 0110
K28.2	92	5C	010 11100	001111 0101	110000 1010
K28.3	124	7C	011 11100	001111 0011	110000 1100
K28.4	156	9C	100 11100	001111 0010	110000 1101
K28.5	188	BC	101 11100	001111 1010	110000 0101
K28.6	220	DC	110 11100	001111 0110	110000 1001
K28.7	252	FC	111 11100	001111 1000	110000 0111
K23.7	247	F7	111 10111	111010 1000	000101 0111
K27.7	251	FB	111 11011	110110 1000	001001 0111
K29.7	253	FD	111 11101	101110 1000	010001 0111
K30.7	254	FE	111 11110	011110 1000	100001 0111

needs to keep transmitting symbols even when no information has to be sent. Fortunately, when mapping the 256 8-bit symbols into 10-bit symbols, there are more than 256 10-bit symbols that satisfy the DC balance and state change restrictions. These extra 10-bit symbols (which do not have a corresponding 8-bit symbol) can be used as "control symbols", also called "control commas". One of them can be used as the "idle comma" to keep transmitting data over the channel and keep the PLLs locked. Table I shows the 12 possible control symbols allowed in 8b/10b coding. The first column shows the names given to these control symbols. The next 3 columns show their 8-bit representation format (in DEC, HEX, and BIN), and the last 2 columns show their 10-bit binary representation depending on the running disparity RD.[4] If K28.7 (FC) is not used, it is easy to use K28.1 (3C) and K28.5 (BC) for synchronizing byte alignment within the bit-stream: that is to say, to find the start/end of the consecutive 10-bit symbols. This is because the unique sequences '0011111' or '1100000' cannot be found at any bit position within any combination of normal codes.

Fig. 2 shows a block diagram of a bi-directional AER link using two FPGA PCBs connected by one single SATA cable (containing two differential pairs of signal wires and three ground wires). Each FPGA connects to one hand-shaken 32-bit pAER sender and one pAER receiver. The "wrapper" block is an IP block generated by Xilinx Core Generator. It contains a wrapper transmitter sub-block wTX and a wrapper receiver sub-block wRX. wTX takes a 32-bit clock-synchronous "sDATA" word and a 4-bit "kchar" word as input. Each of the four "kchar" bits indicate whether the corresponding four bytes in the 32-bit "sDATA" word are control comma bytes or regular user data bytes. In our case, the selected LVDS line rates were 3.0Gbps, obtained from a low jitter differential 150 MHz reference Xtal oscillator on the FPGA PCB. This means that the 32-bit data, transformed into a 40-bit sequence by the 8b/10b code, needed $40/(3 \times 10^9\ \text{Hz}) = 13.3$ ns to be transmitted. The wrapper provides two reference clocks for the user circuitry: one at frequency $f = 1/13.3\ \text{ns} = 75.0\ \text{MHz}$, at the rising edges of which the user circuitry has to provide the 32-bit parallel "sDATA" word and the corresponding 4-bit "kchar" word; and the other at frequency $4f = 300\ \text{MHz}$ (because the 32-bit "sDATA" word contains 4 bytes) and synchronized to clock f. The user designed circuitry within each FPGA in Fig. 2 has 6 blocks:

1) *Asynchronous to Synchronous Converter Block (async2sync)*: This block handles asynchronous handshaking with the 32-bit pAER input port and synchronization between the asynchronous and synchronous domains (or two synchronous domains driven by unrelated clocks). It also provides a 32-bit parallel synchronous version "sDATA" word clock-synchronized with its corresponding "Data Valid" (DV) control signal.
2) *Transmitter FIFO Block (TX-FIFO)*: This block holds a "Stop/Run" state for the transmission of data which is used by the flow control protocol discussed later. It also includes a small FIFO for transient data storage. The need for this FIFO will become more apparent later when we address extension to multi-channel multiplexing.
3) *Transmitter Block (TX)*: For each clock cycle this provides the 32-bit "sDATA" word and the 4-bit "kchar" word required by the "wrapper" (wTX), generates start-up byte alignment sequences, and inserts control symbols for flow control, clock correction, idle commas, or periodic commas for alignment.
4) *Receiver Block (RX)*: This block receives and separates data and control commas. Data symbols are sent to the RX-FIFO block, while control commas are interpreted and executed for proper flow control, word (re)alignment, and clock correction.
5) *Receiver FIFO block (RX-FIFO)*: This block accumulates 32-bit synchronous data symbols from the RX block into a FIFO register, while it empties the FIFO by sending data out to the "sync2async" block. If the fifo fills up above a certain threshold it will trigger the flow control mechanism, as explained below.
6) *Synchronous to Asynchronous Block (sync2async)*. This blocks handles asynchronous handshaking with the 32-bit pAER output port and synchronization with the synchronous data "sDATA" clock domain.

A. Four-Byte Alignment

At start-up, and after all PLLs are properly locked, each TX block will send a sequence of 1024 32-bit comma words (3C BC BC BC) for event word alignment at the RX block on the other side of the link. The Xilinx wrapper IP includes internal circuitry for aligning bytes, using either control comma K28.5 (BC) or K28.1 (3C). In our implementation we selected K28.5 (BC) as the byte alignment comma to be recognized by receiver wRX in the wrapper. As soon as the wrapper receiver circuit detects this comma, from then on all bit sequences will be aligned to recognizable bytes. In our case, our event or data words consisted of 4 bytes. We therefore needed to add extra circuitry in the user receiver to properly align 4-byte events. In principle,

[4]Note that 10-bit "control commas" do not have a corresponding 8-bit data representation, so columns 2 to 4 in Table I would not make sense. In practice, however, the 8-bit to 10-bit conversion circuitry includes an extra control bit (called "kchar") to signal the generation of one of the 12 commas from an 8-bit input code.

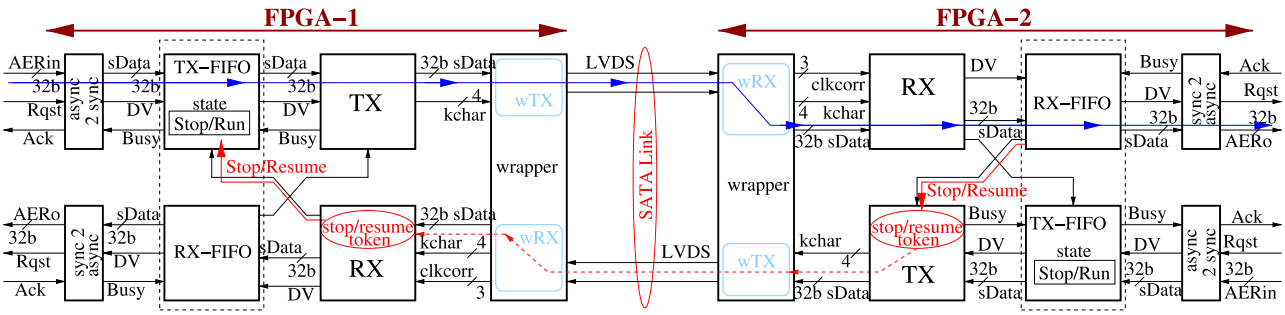


Fig. 2. 2-FPGA LVDS bi-directional AER communication link with flow-control to/from four pAER ports. The figure illustrates how stop/resume control tokens are exchanged via the complementary channel to achieve flow control.

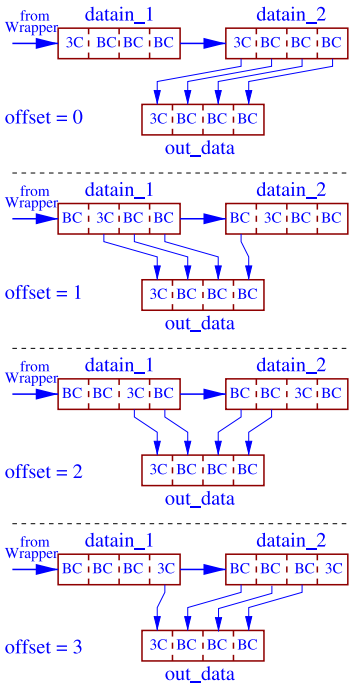


Fig. 3. Illustration of 4-byte re-alignment depending on running offset. The four different offset cases are shown and how the output data is assembled from the flowing input data.

the transmitted control word for alignment "3C BC BC BC" can be received on the receiver side (after correct byte alignment) with four possible offsets: "3C BC BC BC" (offset = 0), "BC 3C BC BC" (offset = 1), "BC BC 3C BC" (offset = 2), or "BC BC BC 3C" (offset = 3). Depending on which of these four possibilities is obtained, the receiver will set its initial "offset" to either '0', '1', '2', or '3', respectively. Once the offset is known, the four byte sequence is then correctly reconstructed by holding the 4-byte data in two consecutive registers, as shown in Fig. 3. First, the 32-bit data read from the wrapper is stored in register 'datain_1', and in the next clock cycle it is copied from register 'datain_1' to register 'datain_2'. Depending on the running value of "offset", four bytes from 'datain_1' and 'datain_2' are copied to output register 'out_data', as shown in Fig. 3, to reconstruct the correct 4-byte event sequence. Once the running offset is detected, it will remain fixed for the forthcoming data stream sequence, unless there is a clock correction (this is discussed in Section III). In this case, the running offset will be updated. For now, let us assume that, after the initial 4-byte alignment, all 32-bit data and control commas sent from the sender side remain aligned and are correctly received on the receiver side. Implementation of the flow control mechanism is explained below.

B. Flow Control

With reference to Fig. 2, let us assume that we are sending events from the left-side pAER sender (top left of FPGA-1 in Fig. 2) to the right-side pAER receiver (top right of FPGA-2 in Fig. 2). Event communication and flow control in the opposite direction is fully symmetric and simultaneous. If no control token needs to be communicated, the default operation is as follows: the async2sync block (see details in Section II-C) acknowledges incoming 32-bit AER events, synchronizes them to the local $f = 75$ MHz clock and transfers the synchronized event "sDATA" in a single clock cycle to the "TX-FIFO" block. By default, this block is in the "Run" state, in which the 32-bit "sDATA" event is transferred to the top left "TX" block in one clock cycle. This TX block will send the 32-bit "sDATA" event to the input port of the "wrapper" (wTX) while setting kchar = '0000' (that is to say, none of the four bytes is a control comma). This way, the events read from the pAER sender will be streamed one after another over the bit-serial LVDS line to the destination FPGA. As will be explained below, the async2sync block needs several clock cycles to acquire one 32-bit event, while the other synchronous blocks need only one clock cycle to transfer "sDATA". Since the "wrapper" needs to read 32-bit data at every $f = 75$ MHz rising clock edge, the TX block will insert "idle commas" (3C BC BC BC) whenever there is no actual event to be transmitted. On the receiver FPGA, the "wrapper" (wRX) will provide reconstructed 32-bit "sDATA" words together with their corresponding 4-bit "kchar" words. Depending on the running "offset" value, the RX block will correctly align the four bytes for each event, and send the correctly assembled 4-byte 32-bit events to the RX-FIFO block. After the sync2async block, the top right pAER receiver will read all these events from RX-FIFO. If the top right pAER receiver temporarily reads events at a speed slower than the top left pAER sender, the RX-FIFO block will accumulate an increasing number of events and eventually fill up. To overcome this, a flow control mechanism is required. We defined two threshold levels for the RX-FIFO block: a "stop threshold"

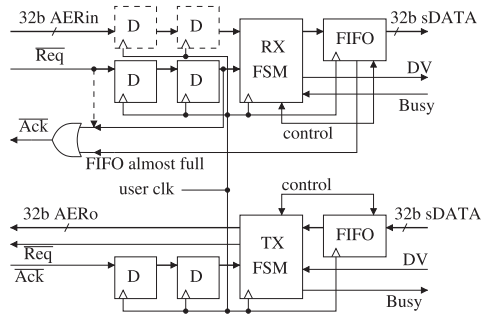


Fig. 4. Simplified diagram of async2sync (top) and sync2async (bottom) blocks. The figure comprises the standard scheme (solid lines) together with a proposed accelerated scheme (dashed lines).

and a "resume threshold". If the FIFO is filled above the "stop threshold", it asserts a "STOP" bit signal for the bottom right TX block (in FPGA-2) of the reverse transmission link. As soon as the TX block detects this active "STOP" signal, it will send a 32-bit "stop token" control word (01 1C 1C 1C), together with kchar = '0111', through the reverse LVDS channel to the bottom left RX block in FPGA-1. This RX block will then activate a "STOP" signal for the top left TX-FIFO block in FPGA-1, which will then change its state to "Stop" and will refuse to accept new events (by activating "Busy"). Consequently, from now on, no more events will be transmitted on the upper forward channel from left to right, and the RX-FIFO in FPGA-2 will be gradually emptied. When the RX-FIFO content crosses the "resume threshold", it will de-assert the "STOP" signal and the TX block in FPGA-2 will send a 32-bit "resume token" control word (00 1C 1C 1C), together with kchar='0111', through the reverse channel to the RX block in FPGA-1. This block will de-assert the "Stop/Resume" bit signal and the TX-FIFO block in FPGA-1 will resume accepting events. Token control word transmission has higher priority than normal event word transmission, so the TX blocks will momentarily stop accepting new events (by activating the "Busy" signal) until the 32-bit token is sent. This will introduce a latency of just one clock cycle in regular event data transmission.

The stop/resume thresholds can be adjusted experimentally for worst case delay. A simple method to measure this delay is by using two GTP transceivers in one FPGA and a counter to obtain the number of clock cycles between sending and receiving a given event. We also observed in previous research [31] that slight differences ($\sim$ 30 ppm) in Xtal oscillator frequencies can also impact system behaviour by introducing asymmetries in the data rates of the links, therefore justifying also the need for adjusting these thresholds experimentally.

C. Asynchronous to Synchronous Domain Interfacing

Fig. 4 shows simplified diagrams for the Asynchronous to Synchronous (top) and Synchronous to Asynchronous (bottom) blocks in Fig. 2. These blocks interface between the synchronous circuits in one FPGA clock domain and external AER circuits, which may be fully asynchronous or driven by other domain clocks. In either case, synchronization is required. Here we used the conventional two D-flip-flop scheme to read in and synchronize the external incoming handshake signals (either $\overline{Req}$ for the top part, or $\overline{Ack}$ for the bottom part). We have performed tests using two schemes: (a) The conventional two D-Flip-Flop scheme [32] shown in Fig. 4 when ignoring the wires and blocks with dashed lines, and (b) an accelerated scheme that includes the wires and blocks with dashed lines. The conventional scheme requires on average 12 clock cycles for one event transaction, while the accelerated one needs on average 6 clock cycles.

1) Conventional Scheme: The conventional synchronization scheme introduces a delay of an extra two clock cycles per handshake signal ($\overline{Req}$ on the receiver and $\overline{Ack}$ on the sender). The FIFOs are both small (4 registers). On the synchronous side (right) of the local clock domain, data is transferred with a single clock cycle transaction whenever signal DV (data valid) is activated while signal Busy is inactive. On the left side, data is transferred using 4-phase handshaking. Under these circumstances, one event transaction requires 10 to 12 clock cycles [32].

2) Accelerated Scheme: For the accelerated scheme we have to impose some restrictions on the sender, the delays of the interconnection lines, and the relative difference between the clock frequencies of sender and receiver circuits. For the sender, the $\overline{Req}$ signal has to be set one (sender) clock cycle after the data lines, the jitter of the interconnection lines has to be negligible with respect to clock cycles, and the difference in clock frequencies between sender and receiver cannot be greater than a factor two. Under these circumstances, one event transaction can be performed in either 5 or 6 transmitter clock cycles. The scheme works as follows. For the async2sync receiver interface (top in Fig. 4), the (active low) $\overline{Ack}$ signal is formed by the OR of signal (active high) "FIFO almost full" and (active low) $\overline{Req}$. Consequently, the returning $\overline{Ack}$ signal needs to be synchronized on the other side, and this is done using another two D-flip-flop scheme. The receiver also synchronizes all data lines, in order to delay them 2 clock cycles. It is well known that, in general, synchronizing parallel data lines does not work [33]. This is because each data bit can be captured at different receiver clock edges due to jitter in the sender data edges, jitter in the receiver data latches clock edges, different delays of the data bit lines, or different threshold levels of the data bit latches. However, if it is guaranteed that at the receiver side (a) $\overline{Req}$ arrives always after all data lines have stabilized, and (b) $\overline{Req}$ and all data lines stay stable for at least 2 clock cycles, then the synchronized version of $\overline{Req}$ will capture all data bits correctly. The chance of metastability, however, is multiplied by the number data bits plus one (33 in our case). Nonetheless, for the Spartan6 specifications, this chance is still several years. In the experimental results (Section V) we show in-hardware verifications of both schemes, the conventional one and the accelerated one, tested for over 65 hours without any transmission errors.

III. Clock Correction

A. The Problem of Multiple Clock Domains

Fig. 5 shows the bi-directional LVDS link between the two FPGA PCBs in Fig. 2, this time highlighting the reference Xtal oscillator in each FPGA PCB. Each PCB has its own Xtal oscillator which generates a low-jitter reference clock (in our case,

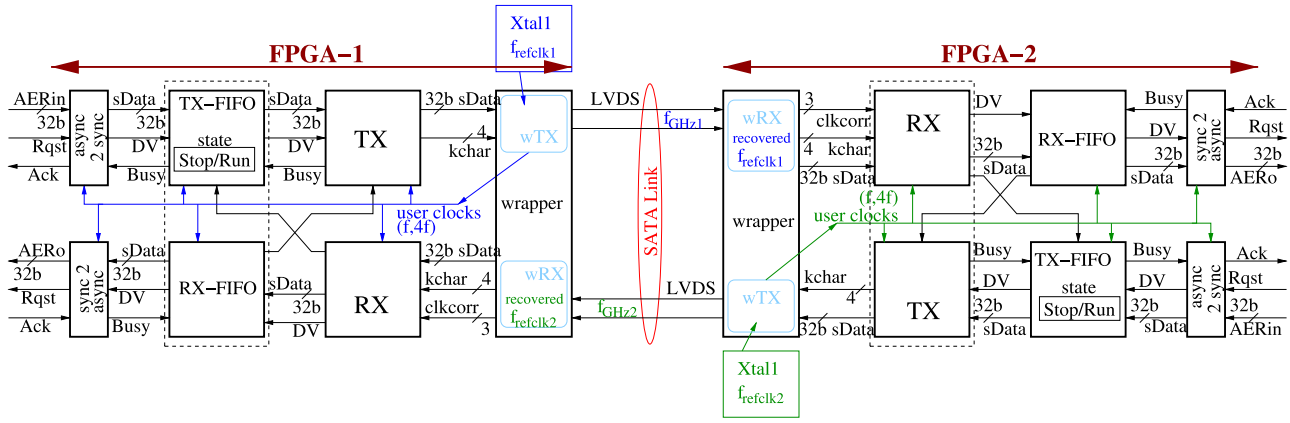


Fig. 5. 2-FPGA LVDS bi-directional AER communication link highlighting the reference Xtal oscillator in each FPGA. Xtal frequencies may differ by a few ppms, thus requiring clock correction techniques for reliable communications.

150 MHz). In FPGA-1 PCB, reference clock f_{refclk1} will be up-converted by special circuitry inside the wrapper to the line frequency f_{GHz1} used by the LVDS transmission (wTX) lane (typically in the range of a few GHz). Extra clock management circuitry also provides two additional reference clocks of frequencies f and $4f$ (when 32-bit 4-byte data is read per clock cycle).[5] Wrapper input data (32-bit "DATA" and 4-bit "kchar") is read at the rising edges of f. Clock f_{GHz1} is used to encode the bit-serial data stream through the top LVDS lane in Fig. 5. The receiver circuitry inside the wrapper (wRX) in FPGA-2 PCB will recover the f_{GHz1} clock to decode the serial data and convert it back to parallel 32-bit "DATA" words. Let us call this recovered clock $f_{r\mathrm{GHz1}}$. It will have exactly the same frequency as f_{GHz1} but with totally uncorrelated phases and higher jitter. On the receiver side it is possible to down-convert the recovered clock $f_{r\mathrm{GHz1}}$ to the original frequency of $f_{refclk1}$. Clock signals $f_{refclk1}$, f, $4f$, f_{GHz1} in FPGA-1 PCB, and recovered clocks $f_{r\mathrm{GHz1}}$ and $f_{refclk1}$ in FPGA-2 PCB are all mutually synchronous because they are all from the same reference Xtal1 oscillator.

Likewise, similar clocks are generated derived from the Xtal2 oscillator in FPGA-2 PCB. Both Xtal oscillators (Xtal1 and Xtal2) must have the same frequency. However, there is always a small frequency difference (typically approximately $+/-$ 100 ppm) between reference clock sources. As a result, each wrapper uses a slightly different frequency for its transmit datapath (wTX) and its receive datapath (wRX). There are therefore two independent clock domains and at some interface they are bound to interfere with each other. This situation can be dealt with using a clock correction technique.

Alternatively, instead of clock correction, an attempt can be made to extend the same clock domain to all circuitries by propagating the clock recovered at wRX. The drawback of this approach is that the clock recovered by a receiver in a wrapper has higher jitter than the original clock, resulting in less reliable communication. If, instead of having just one bi-directional link (as in Fig. 5), there are more links per FPGA-PCB and many PCBs are interconnected, then it will be impractical to propagate a single Xtal reference clock to all PCBs through a sequence of clock up-conversions and recovery down-conversions. This would progressively degrade clock jitter and ultimately render the links unusable. In this case, clock correction techniques offer a robust, reliable solution. The use of one such technique, available in some FPGAs, is detailed below.

[5]Frequencies f_{GHz} and f are related by $f_{\mathrm{GHz}} = 10 \times n_{bpe} \times f$, where n_{bpe} is the number of bytes per event. Throughout this paper $n_{bpe} = 4$, $f_{\mathrm{GHz}} = 3\,\mathrm{GHz}$, $4f = 300\,\mathrm{MHz}$, and $f = 75\,\mathrm{MHz}$.

B. Clock Correction Implementation

Consider the situation in Fig. 6(a), which shows an FPGA with 4 bi-directional LVDS links (8 LVDS lanes). This FPGA can be thought of as being held in one FPGA-PCB with its reference Xtal oscillator. Many of these single-FPGA PCBs could be interconnected in a mesh-like fashion using bi-directional LVDS links to form a 2D array of PCBs, as shown in Fig. 1. Each FPGA contains 4 "wrapper" circuits (like the ones discussed in Section II and shown in Figs. 2 and 5), one per bi-directional LVDS link.

Each FPGA-PCB has one low jitter Xtal oscillator (150 MHz in our case). These low jitter oscillators usually provide a differential output signal, which is converted to a single-ended signal inside the FPGA with specially dedicated low-jitter buffers. This reference clock is then routed to all wrapper transmitter (wTX) subcircuits, where it is up-converted to a clock signal at line frequency $f_{\mathrm{GHz}i}$ (in the range of 1–3.2 GHz for Spartan-6). The reference clock $f_{refclki}$ is also routed to a "clock manager module" (*clk mgr* in Fig. 6(a)), where two synchronized clocks of frequency f and $4f$ are provided. These two synchronized clocks are routed to the wrappers and are also available for user logic.

Fig. 6(b) shows the internal structure of the wrapper in more detail (although still overly simplified). In the wrapper transmitter sub-block (wTX), "DATA_out" (32-bit events or control commas) is read synchronously at the rising edges of clock f, together with its corresponding 4-bit "kchar_out" word. This parallel 32-bit word is then converted byte by byte, using 8b/10b encoding, to give a 40-bit parallel word, which is then serialized into a bit-stream using the up-converted clock frequency

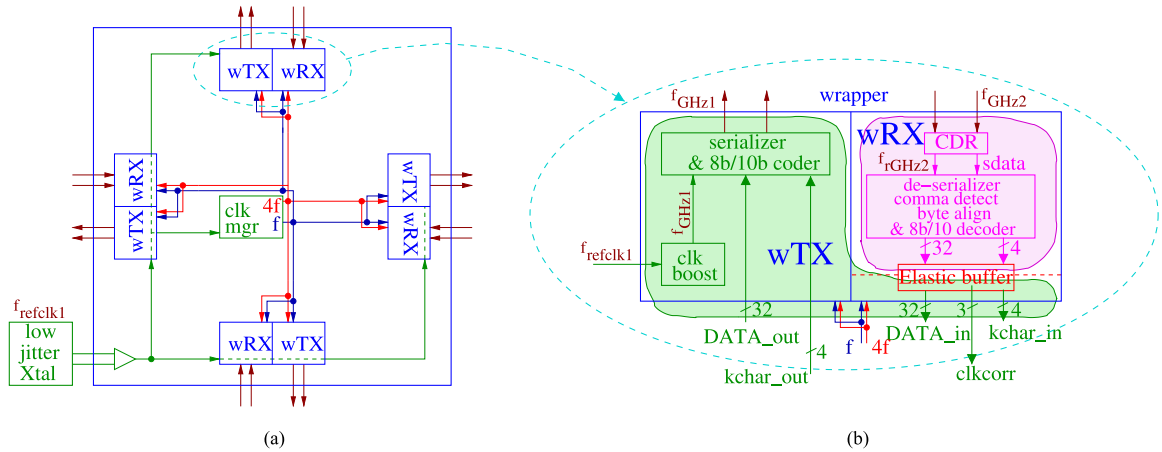


Fig. 6. (a) FPGA PCB with four bi-drectional LVDS bit-serial links. (b) Detail of wrapper transmitter (wTX) and receiver (wRX) sub-blocks.

f_{GHz1}, and sent through the appropriate LVDS driver circuits to a differential pair of wires. Fig. 6(b) also shows the simplified internal structure of the "wrapper receiver sub-block" (wRX). A CDR (Clock and Data Recovery) circuit receives the LVDS bit-serial stream of input data, extracting a recovered clock $f_{r\mathrm{GHz2}}$. A deserializer circuit then converts this bit-stream into a sequence of parallel bytes, detecting byte-alignment commas and immediately aligning the bytes. 8b/10b encoded 10-bit words are then decoded into 8-bit data or comma bytes. Up to this point, logic has been clocked using clocks derived from the recovered line clock $f_{r\mathrm{GHz2}}$, but now the recovered 32-bit events (or commas) need to be transferred to registers and logic has to be clocked by clock signal f, which belongs to a different clock domain (the one derived from $f_{refclk1}$). This clock domain "crossover" is handled using an "*Elastic Buffer*" [34] provided by the FPGA manufacturer within the wrapper. An Elastic Buffer is an asynchronous FIFO which is written using one clock and read using another clock. The frequencies of the two clocks are fairly similar but not exactly equal. As a result, the elastic buffer will slowly either fill up or empty out. This is avoided by defining a clock correction comma. The clock correction comma, which can be defined as a single byte, a 2-byte group, or a 4-byte group, has to be inserted into the data stream sent by the user-designed transmitter TX with certain periodicity (the exact periodicity depends on the expected worst case discrepancy between f_{GHz1} and f_{GHz2}). If the elastic buffer fills up above a given threshold, one clock correction comma is removed and not delivered to the output port. This way, the elastic buffer is suddenly emptied by the amount of one comma. On the other hand, if the elastic buffer is emptied below another given threshold, one clock-correction comma is inserted in the elastic buffer, and this comma has to be ignored at the user-designed receiver, RX in Figs. 2 and 5. The elastic buffer is thus suddenly filled by the amount of one comma. Since we were using 4-byte events, it made sense to use 4-byte clock-correction commas to avoid event de-alignment after clock-correction. However, we decided to use a one-byte clock-correction comma (BC) for lower comma traffic, which combined with our byte-alignment circuit, allows for on-the-fly event re-alignment. The use of one-byte clock correction commas had the effect of changing the alignment offset in Fig. 3, either incrementing it or decrementing it by '1'. The insertion and removal by the wrapper of user-defined clock-correction commas is signaled by the 3-bit signal 'clkcorr'. The user circuit RX state machine is designed to properly handle these commas.

IV. Multiple Channel Multiplexing

One SATA link with a line frequency of 3.0 Gbps using 8b/10b byte encoding and transmitting 32-bit events can transmit at a rate of 75 Meps (including control commas) per link direction. This event rate is fairly high compared to transmission speed through a standard parallel asynchronous AER port. For example, purely asynchronous 128 $\times$ 128 pixel DVS sensors have been reported to achieve speeds of almost 10Meps for 15-bit events [28]. The synchronization circuitry within the async2sync and sync2async blocks also results in event transactions of a maximum of 6 clock cycles (if the conditions discussed in Section II-C are met). This would result in an event rate of 12.5Meps. To take advantage of the available bandwidth in these SATA links, several asynchronous parallel AER channels can be multiplexed over one link. Fig. 7 illustrates our proposed multiplexing of $2 \times k$ AER channels over one bidirectional SATA link (k channels in each direction). The blocks shown within the broken lines in Figs. 2 and 5 are now replaced by the blocks shown within the broken lines in Fig. 7. If q is the smallest integer such that $2^q \geq k$, then the top q bits of the 32 sDATA bits are sacrificed to encode AER channel number within each event. Each TX-FIFO receives AER events of 32-q bits and writes them into its registers, activating an output signal "Empty" when there is no data left in its registers. The "Fair Tag Encoder" FSM selects one non-empty TX-FIFO to read, following a fair selection algorithm (see below in Section IV-B). This TX-FIFO channel is selected by the Ch-MUX multiplexer block. The 32-q bit data is read, the corresponding top q bits (Channel ID) are appended, and the complete 32-bit sDATA is then sent to the TX block if its "Busy" signal is non-active. On the receiver side, the "Tag Decoder" block reads

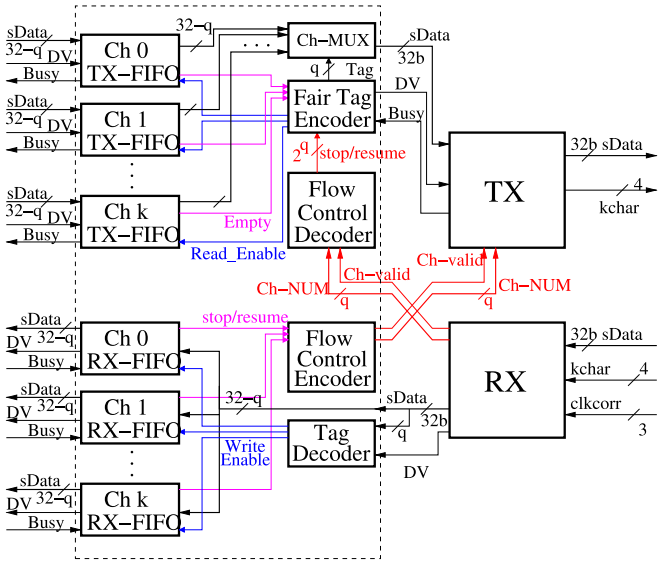


Fig. 7. Block diagram of channel multiplexing arrangement with k transmitting and k receiving channels together with flow control and tag handling blocks.

the top q bits and activates the corresponding "Write-enable" signal for the destination RX-FIFO, which reads the lower 32-q bits of the incoming sDATA (32 bit) word. Each RX-FIFO will be read out through its output channel.

A. Flow Control

If the read out speed at the output of an RX-FIFO is slower than the speed at which events are received, the corresponding RX-FIFO will fill up. RX-FIFOs will activate a "stop" signal if they are filled beyond a pre-set threshold (which should be lower than the FIFO's capacity), or a "resume" signal if they are emptied below a second pre-set threshold. This one-bit stop/resume signal is read by the "Flow Control Encoder" FSM, which will tell the TX block with highest priority the channel number "Ch-NUM" whose RX-FIFO is getting close to full while activating the "Ch-valid" signal. The TX block will send a 32-bit flow control comma (Ch 1C 1C 1C), with kchar = '0111', where Ch is an 8-bit byte in which: (a) the 7 most significant bits encode the channel number (so that up to 128 channels can be multiplexed), and (b) the least significant bit is either '1' to signal "stop" or '0' to signal "resume". This flow control comma is received by the RX block on the other side of the link, which will communicate the channel stop/resume command to the "Flow Control Decoder" FSM. This FSM will then decode the saturating channel's ID and set the corresponding stop/resume signal (one of the 2^q lines) for the "Fair Tag Encoder", which will in turn enable/disable the "Read Enable" signal for the channel, so that the corresponding channel TX-FIFO will stop/resume accepting new input events.

Note that this flow control scheme is very similar to the single-channel scheme explained in Section II-B, except that here the flow control comma uses all 8 bits of the first byte and the upper q bits of the events are sacrificed to encode channel number.

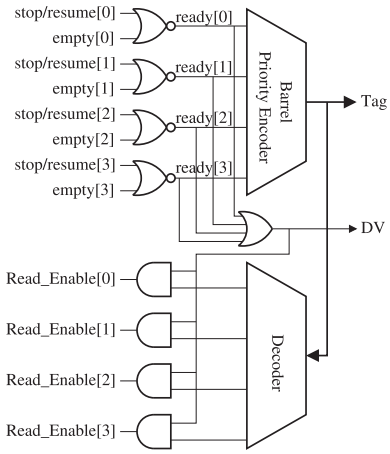


Fig. 8. Fair encoder operation diagram.

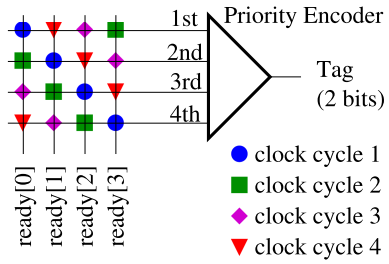


Fig. 9. Schematic operational diagram of barrel priority encoder.

B. Fair Tag Encoding Operation

Fig. 8 shows a simplified diagram of the "Fair Tag Encoder" block. Channel signals 'stop/resume' and 'empty' are OR-ed to generate 'ready' signals. These 'ready' signals are fed to a "Barrel Priority Encoder". This block is basically a priority encoder whose priority preference is circularly shifted one position each clock cycle, as illustrated in Fig. 9. The "Barrel Priority Encoder" generates the corresponding output channel Tag, which is used by a Decoder circuit to generate 'Read_Enable' signals.

V. Experimental Results

In the experimental measurements reported here we use Spartan-6 GTP interfaces operating at 3.0 Gbps with error-free transmission. This data rate is very close to their maximum data rate limit of 3.2 Gbps. This requires careful PCB design and component choices. The high speed traces on the PCB were designed using industry-standard techniques. The PCB manufacturer supplied track width and spacing based on the proposed board stack-up and required impedance. The Cadence Allegro PCB tools were set up to use these parameters. The differential pairs were automatically length matched and all bends were chamfered and vias avoided where possible. The high speed tracks were routed on the outer layers of the PCB with a ground plane beneath. Standard surface mount SATA connectors were used on the PCB. During board commissioning, the drive strength of the FPGA differential drivers was adjusted to ensure adequate noise margin on the links.

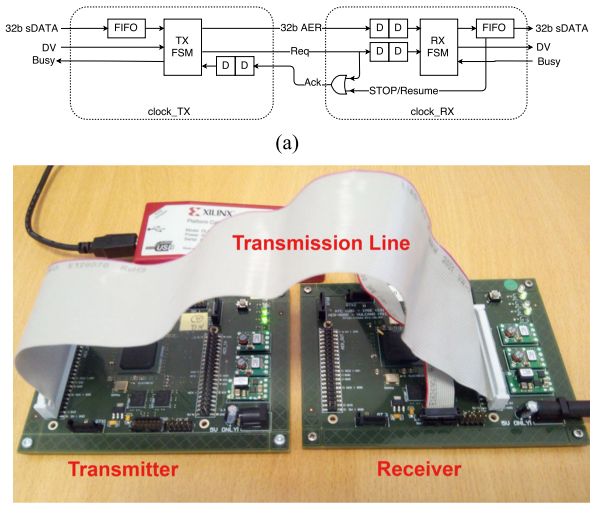


Fig. 10. Separate setup to test and characterize the accelerated Sync2Async and Async2Sync scheme. (a) Schematic diagram, and (b) Experimental setup.

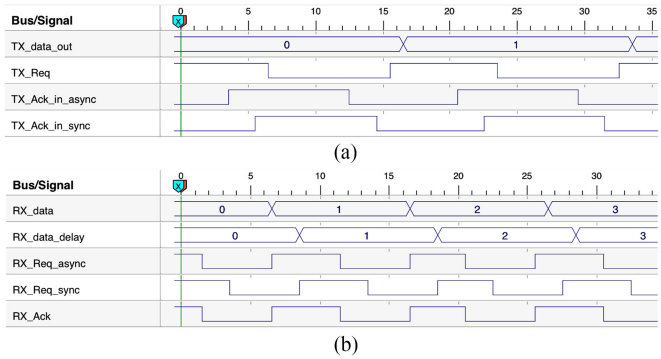


Fig. 11. Chipscope captured signals at (a) TX circuit with 250 MHz clock and (b) RX circuit with 143 MHz clock frequency.

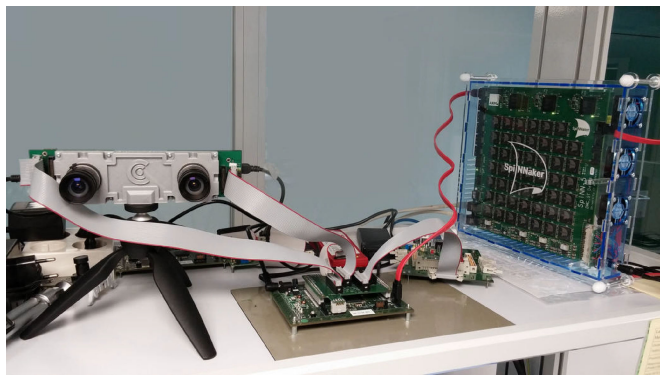


Fig. 12. Example setup with two ATIS retinas [35] connected via AER parallel ports to the AER-Node board [25], which connects via SATA to a 48-chip SpiNNaker Board [27].

A. Test of Accelerated sync2async and async2sync Scheme

In a preliminary characterization, we tested first the correct operation of the accelerated async2sync and sync2async scheme presented in Section II-C. Fig. 10(a) shows the schematic diagram of the setup used for this, with the TX on the left and the RX on the right sides, each with its own independent clock. Let us call their clock periods P_{TX} and P_{RX} respectively. At TX the 32-bit AER data is set one clock cycle before Req. Ack returns (if STOP/resume signal is low) without passing through any synchronization nor state machine at the receiver. If wire delays are negligible, the TX requires $6P_{TX}$ to perform one full data transfer: 1st to write data, 2nd to write Req, 3rd to capture Ack, 4th and 5th to capture the synchronized version of Ack by the TX FSM, and 6th to allow the FSM to write the new data. If there are delays in the wires, pads, and OR gate, then one event transmission requires $6P_{TX} + 4\tau_{\text{line}}$ (where $2\tau_{\text{line}}$ includes one full round: two physical wires and connectors, four pads, and the OR gate). On the RX side, the circuit needs $3P_{RX}$ to capture one event. If RX clock is faster than TX clock ($P_{TX} > P_{RX}$) there are no communication problems. However, if TX clock is faster, we have to guarantee that $6P_{TX} + 4\tau_{\text{line}} > 3P_{RX}$. The worst case is when τ_{line} is negligible, which results in $2P_{TX} > P_{RX}$ (or $2f_{RX} > f_{TX}$). Consequently, if TX clock frequency is not more than twice the RX clock frequency, correct communication is guaranteed, independent of the physical delays of lines, pads, connectors, etc.

To verify this, we assembled the experimental setup shown in Fig. 10(b) with intentional long external wires. Two AER-Node boards [25] were used, interconnected through their parallel ports with a relatively long parallel bus ribbon cable. The TX circuit was clocked at 250 MHz while the RX was clocked at 143MHz. This setup showed error-free transmission tested over several days. Fig. 11(a) shows signals Data, Req, Ack at pad, and Ack after synchronizers, captured inside the TX FPGA using Chipscope. We can see that one event cycle transmission requires 17 TX clock cycles (68 ns), although sometimes it would require 16. The measured average was 16.98 cycles (67.93 ns). Since we estimated this delay as $6P_{TX} + 4\tau_{\text{line}}$, it results in $\tau_{\text{line}} = 10.98$ ns (equivalent to 2.75 TX clock cycles). Fig. 11(b) shows signals Data at the pads, after synchronizers, Req at the pad, after synchronizers, and Ack at the pad, captured inside the RX FPGA using Chipscope. We can see one event cycle transmission of 10 RX clock cycles (69.93 ns) and another one of 9 RX clock cycles (62.94 ns). The measured average was 9.71 RX clock cycles (67.93 ns). Therefore, this accelerated scheme setup was able to transmit at an average of 67.93 ns per event, or equivalently, 14.72 Meps. By using the non accelerated scheme the average event transmission rate was 9.75 Meps.

In the experiments that follow, the async2sync and sync2async interfaces are completely inside the FPGAs thus minimizing τ_{line}. The measured results shown next also demonstrate error free transmissions.

B. Serial Link Characterization

Figs. 12 and 13 show two typical setups in which the proposed bidirectional serial link was used. The figures show two separate AER retina sensors, each connected to the AER-Node Board [25] by a parallel AER connector (using a custom adapting

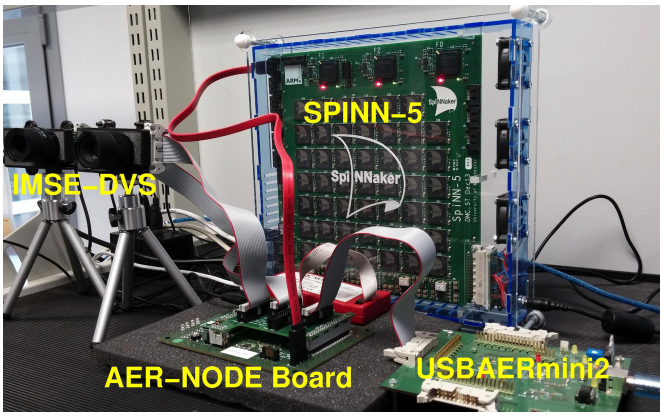


Fig. 13. Example setup with two DVS [28] retinas connected via AER parallel ports to the AER-Node board [25], which connects via SATA to a 48-chip SpiNNaker Board [27].

PCB that makes it possible to connect up to 4 AER parallel ports). The two retinas communicate with the Spartan6 FPGA on the AER-Node Board, which in turn communicates with one 48-chip SpiNNaker Board [27] through serial SATA. The SpiNNaker board receives events from the two retinas' AER ports, processes them and sends the resulting event flow back to the AER-Node Board via the same SATA wire. The AER-Node Board sends the results through another parallel AER port to a USBAERmini2 Board [10] which communicates with a host computer by USB to display the results in real time.

The proposed communication scheme was experimentally characterized using a pair of Spartan6 FPGAs located on two different (SpiNNaker) PCBs. Each Spartan6 (XC6SLX45T-3) FPGA used its own 150MHz Xtal oscillator. To test the multiplexed link at maximum throughput (thereby, forcing flow control), we used one GTP port on each FPGA, as illustrated schematically in Fig. 14. Each FPGA used its local 150 MHz reference clock plus an additional 67 MHz clock, so each FPGA included two separate clock domains. The setup was configured with 4 separate bidirectional AER Channels (3 synchronous and one asynchronous) multiplexed over the SATA link. For this we used a "Test Pattern Generator" (TPG) and "Test Pattern Checker" (TPC) transmitter/receiver pair. The TPG provides a known sequence of patterns, while the TPC checks and counts event errors in that sequence and computes the effective event rate received (excluding all control commas). In each FPGA, three TPG/TPC pairs were clocked with the same clock as the Transceiver/Multiplexing core discussed in Figs. 2, 5, and 7. This is a clock running at $f = 75$ MHz, derived from the external 150 MHz Xtal reference oscillator. Each of the 3 synchronous TPGs could therefore provide an event rate of up to 75 Meps (one per clock cycle). The 4th TPG/TPC pair was clocked with the additional 67 MHz clock and interfaced through a pair of async2sync blocks. The LVDS line rate was set at 3.0 Gbps. We tested the setup using the two synchronization schemes discussed in Section II-C and Fig. 4: the faster one requiring on average 6 clock cycles per event transaction, and the slower one requiring on average 12. Table II summarizes the results for the 6-cycle case after testing the setup for 65 hours. None of the

TABLE II
EXPERIMENTAL CHARACTERIZATION TEST FOR FAST 6-CYCLE EVENT TRANSACTION SYNCHRONIZATION SCHEME ON CHANNEL 1

Channel#	Event Rate (Meps)	Link utilization(%)	Error Ratio
Channel1	11.14	14.85	0
Channel2	26.29	35.05	0
Channel3	18.75	25.00	0
Channel4	18.75	25.00	0
Total	74.93	99.90	0

TABLE III
EXPERIMENTAL CHARACTERIZATION TEST FOR SLOWER 12-CYCLE EVENT TRANSACTION SYNCHRONIZATION SCHEME ON CHANNEL 1

Channel#	Event Rate (Meps)	Link utilization(%)	Error Ratio
Channel1	6.43	8.57	0
Channel2	31.00	41.33	0
Channel3	18.75	25.00	0
Channel4	18.75	25.00	0
Total	74.93	99.90	0

channels detected a single error in the transmission. The link bandwidth (75 Meps) was shared by the four Channels as indicated in Table II, covering effective data events plus control commas (for flow control, clock correction, or idle commas). The TPGs clocked at 75 MHz attempted to deliver data at one event per clock cycle, but were slowed down by the corresponding "Fair State Encoder" whenever the SATA link bandwidth was reached. The TPG clocked at 67 MHz (Channel 1) could only deliver events at a much lower rate because of the synchronization delays. This explains why the effective event rate for this link dropped to 11.14 Meps. The rest of the bandwidth was split up between the other Channels as shown in Table II. Note that Channel2 ends up transmitting at a higher rate than Channels 3–4. This is because of the priority endcoder design in Figs. 8 and 9, which assigns to each Channel different priorities every clock cycle (to prevent one pAER transmitter from blocking others). Since Channels 2 to 4 always have data ready to send, they will use 25% of the time (when they have highest priority) to send their data. When highest priority is with Channel1, sometimes there is no data ready to send. In this case Channel2 has second priority and will use this time for its data.

Total data bandwidth was thus 74.93 Meps (99.90% of link bandwidth). The remaining 0.07 Meps bandwidth (0.10%) was used by control commas. Table III shows the same results, but for the case of using the slower 12-cycle synchronization scheme. In this case Channel 1 achieved a lower throughput, but the total link bandwidth was kept the same. Note that Channel1 event rate in Table III is slightly faster than half of that in Table II, which might be surprising because we are expecting the fast scheme to require between 5-to-6 clock cycles and the slow scheme between 10-to-12. When using the fast scheme, throughput is limited by the TX clock at 67 MHz. However, for the slow scheme the delays depend on both the TX and RX clocks, and the effective clock cycle is somewhere between 67 MHz and

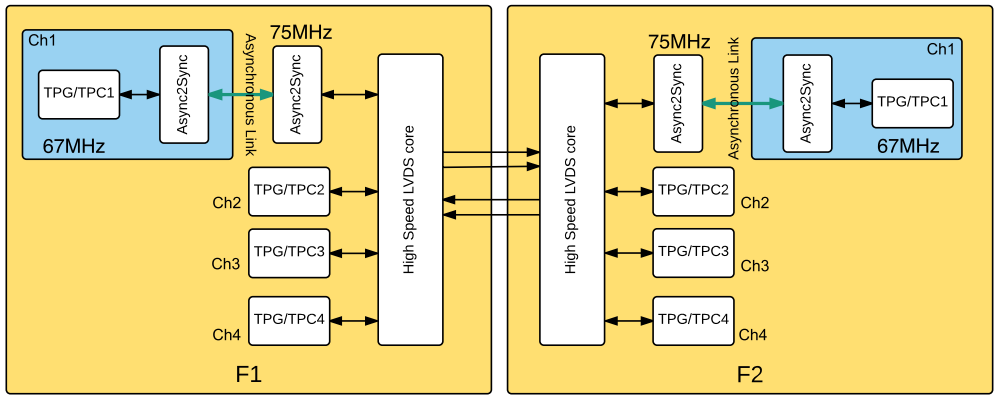


Fig. 14. Simplified diagram illustrating the experimental configuration inside the two Spartan6 FPGAs.

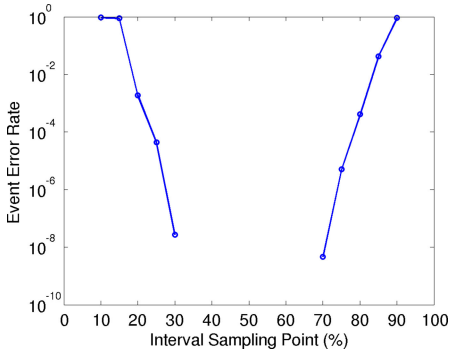


Fig. 15. Measured event error rate while sweeping interval sampling point.

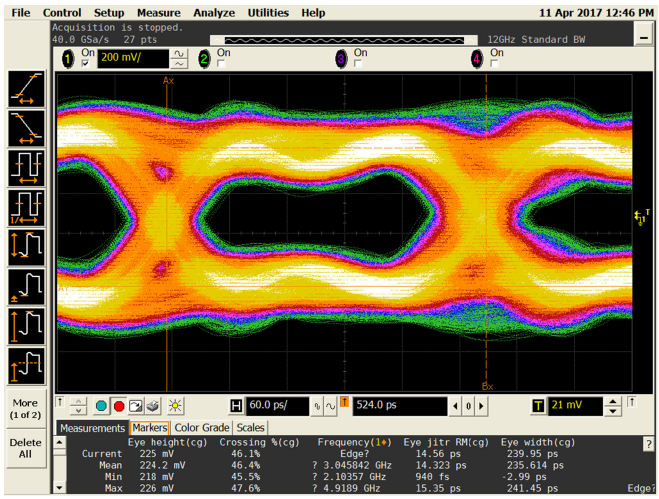


Fig. 16. Measured eye diagram on the LVDS lanes operating at 3.0 Gbps.

75 MHz, resulting in an event transmission rate slightly better than half of the fast one.

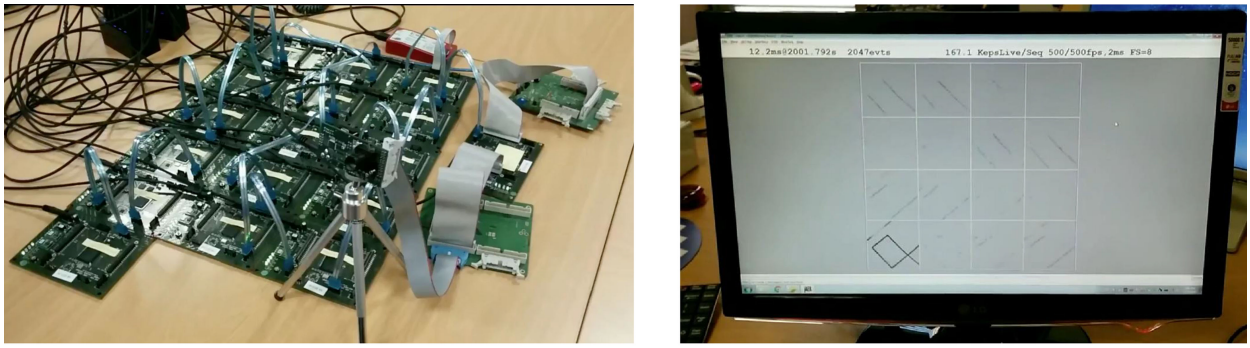
The FPGA GTP receivers made it possible to tune the voltage sampling point of the physical LVDS line to compensate for possible asymmetries. This sampling point is expressed as a percentage of the full range, the central point being 50%. We tested the event error rate of the link as a function of the interval sampling point. The results are shown in Fig. 15, where it can be seen that event error rate was null between sampling points 0.35% and 0.65%. Outside this range, the event error rate increased exponentially.

Fig. 16 shows the eye diagram measured using an Agilent DSO81304B oscilloscope with 12GHz bandwidth soldered probes. Eye opening was 235 ps width times 224 mV height, with an average rms jitter of 14.3 ps. This confirms a safe enough margin on the physical design side for the transmission speed of 3 Gbps (333 ps per bit). We can see in the figure that transmission of one bit requires an average of 328 ps, which yields an average transmission frequency of 3.046 GHz. The figure also shows that the differential amplitude of the physical LVDS signal has an average of about 700 mV peak to peak.

C. Application Example

Fig. 17(a) illustrates a multi-FPGA multi-PCB application example of a neuromorphic system that extensively exploits the presented LVDS interface protocol technique. The setup shows 17 Spartan6-based AER-Node Board PCBs [25] which receive real-time events from an event-driven spiking retina Dynamic Vision Sensor [28]. The sensor event flow is distributed to a mesh of event-driven convolution filters [21] that emulate the operation of the vertebrate V1 visual cortex. These convolution filters are implemented on the Spartan6 FPGAs and the event traffic is distributed using the SATA LVDS links. The output event flow of each convolution filter is then routed back through the same SATA LVDS links and concentrated on one of the AER-Node Boards, which converts the input and a selected number of outputs to parallel AER, which is then interfaced to a PC through a USBAERmini2 PCB [10], to monitor in real time the activity of the selected V1 filters on a screen, as shown in Fig. 17(b), using jAER [36].

Typical techniques for mapping generic spiking networks onto modular hardware exploit the mapping of computational architectures to 2D meshes [15], [21], where unit elements are connected to nearest neighbours. Each unit element includes a processing module and a router. Each router contains its own routing table, and the set of all routing tables defines how the original computational architecture has been mapped onto the



(a) (b)

Fig. 17. Example application of neuromorphic event-driven sensing and computing setup including an event-driven Dynamic Vision Sensor and array of event-driven filtering blocks emulating the V1 layer of the vertebrate visual system. (a) Physical setup using 17 Spartan6 AER-Node Boards [25] intercommunicated through our SATA serial protocol, one event-driven vision camera [28], and one USBAERmini2 computer interfacing PCB [10] to monitor computations in real time on a monitor screen, shown in (b).

2D array. In this approach, physical links only exist between neighbours. This may result in congestions if for example two distant unit elements have to maintain a high event data rate between them, because this event traffic would use time of all the routers in the path. However, with the proposed technique of multiplexing multiple AER paths on the same physical SATA wire, it is possible to establish direct paths between distant unit elements without necessarily going through all the routers within the 2D mesh path.

VI. Conclusion

We have presented a method for multiplexing multiple asynchronous and/or synchronous Address-Event-Representation channels over a physical bidirectional inter-FPGA LVDS link. The scheme allows for the separate, independent flow control of each AER channel and includes proper byte alignment control for the serial communication, together with clock correction techniques for compensating clock drifts between the reference clocks of the FPGAs. Experimental results using the LVDS links of two Spartan6 FPGAs on separate boards demonstrated the correct operation of the link. Exhaustive tests were carried out in hardware, demonstrating error free transmissions over extended time periods while communicating events at the full bandwidth of the physical links. Complete vhdl/verilog codes and example setups are made available as supplementary material.

Although the lowest data transmission layers are well known and widely used (Xilinx IP wrapper), our contribution in the field of biology-inspired computing is the setup of heterogeneous architectures able to combine various custom processing elements (like concurrent ARM-based Spinnaker platform, massively parallel FPGA-based emulators of spiking neurons) connected to each other via various asynchronous interfaces (2-of-7 parallel bus, pAER 16-bit parallel bus, bi-directional LVDS serial links) and driven by multiple event-based neuromorphic sources of real sensory signals (such as artificial Retinas or Cochleas). Even though computing elements (CPU's and FPGA's) and transmission links operate asynchronously and at different speeds, we have proven that they are able to cooperate and the transmission can be error-free even when reaching physical data rate limits. One special concern and contribution in this work was sharing the bandwidth of single LVDS channels by various agents (multiple transmitters and receivers). We demonstrate error-free transmission running at almost maximum possible speed sharing the bandwidth of single bit-serial channels among multiple synchronous and asynchronous elements running at different rates. Priority encoding and flow-control is critical for avoiding buffer overflow when control and data events appear concurrently. Illustrations of multiple and heterogenous neuromorphic setups are provided.

References

[1] M. Sivilotti, "Wiring considerations in analog VLSI systems with application to field-programmable networks," Ph.D. thesis, Comput. Neural Syst., Caltech, Pasadena, CA, USA, 1991.

[2] M. Mahowald, "VLSI analogs of neuronal visual processing: A synthesis of form and function," Ph.D. thesis, Comput. Neural Syst., Caltech, Pasadena, CA, USA, 19'92.

[3] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuts Syst. II*, vol. 47, no. 5, pp. 416–434, May 2000.

[4] V. Chan, C. Jin, and A. van Schaik, "An address-event vision sensor for multiple transient object detection," *IEEE Trans. Biomed. Circuits Syst.*, vol. 1, no. 4, pp. 278–288, Dec. 2007.

[5] Z. Fu, T. Delbrck, P. Lichsteiner, and E. Culurciello, "An address-event fall detector for assisted living applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 2, no. 2, pp. 88–96, Jun. 2008.

[6] B. Wen and K. Boahen, "A silicon cochlea with active coupling," *IEEE Trans. Biomed. Circuits Syst.*, vol. 3, no. 6, pp. 444–455, Dec. 2009.

[7] S. Mitra, S. Fusi, and G. Indiveri, "Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI," *IEEE Trans. Biomed. Circuits Syst.*, vol. 3, no. 1, pp. 32–42, Feb. 2009.

[8] R. Mill, S. Sheik, G. Indiveri, and S. L. Denham, "A model of stimulus-specific adaptation in neuromorphic analog VLSI," *IEEE Trans. Biomed. Circuits Syst.*, vol. 5, no. 5, pp. 413–419, Oct. 2011.

[9] D. G. Chen, D. Matolin, A. Bermak, and C. Posch, "Pulse-modulation imaging—Review and performance analysis," *IEEE Trans. Biomed. Circuits Syst.*, vol. 5, no. 1, pp. 64–82, Feb. 2011.

[10] R. Serrano-Gotarredona *et al.*, "CAVIAR: A 45k neuron, 5M synapse, 12G connect/s AER hardware sensory-processing-learning-actuating system for high speed visual object recognition and tracking," *IEEE Trans. Neural Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[11] M. Kahn *et al.*, "SpiNNaker: Mapping neural networks onto a massively-parallel chip multi-processor," in *Proc. IEEE Int. Joint. Conf. Neural Netw.*, Jun. 2008, pp. 2849–2856.

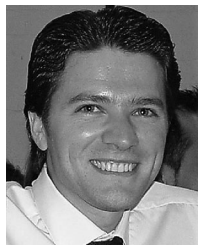
[12] J. Fieres, J. Schemmel, and K. Meier, "Realizing biological spiking network models in a configurable wafer-scale hardware system," in *Proc. IEEE Int. Joint. Conf. Neural Netw.*, Jun. 2008, pp. 969–976.

[13] P. O. Pouliquen and A. G. Andreou, "Bit-serial address-event representation," in *Proc. 33rd Annu. Conf. Inform. Sci. Syst.*, Baltimore MD, USA, Mar. 1999, pp. 893–896.
[14] K. Boahen, "A burst-mode word-serial address-event link I, II, III," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 7, pp. 1269–1300, Jul. 2004.
[15] S. Furber, F. Galluppi, S. Temple, and L. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 2, no. 5, pp. 652–665, May 2014.
[16] L. A. Plana *et al.*, "A GALS infrastructure for a massively parallel multiprocessor," *IEEE Des. Test Comput.*, vol. 24, no. 5, pp. 454–463, Sep./Oct. 2007.
[17] *LVDS Owner's Manual*, 4th ed., Nat. Semiconductors, Santa Clara, CA, USA, 2008.
[18] L. M.-A. *et al.*, "A LVDS serial AER link," in *Proc. 13th IEEE Int. Conf. Circuits Syst.*, 2006, pp. 938–941.
[19] H. Berge and P. Hfliger, "High-speed serial AER on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2007, pp. 857–860.
[20] D. B. Fasnacht, A. M. Whatley, and G. Indiveri, "A serial communication infrastructure for multi-chip address event systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 648–651.
[21] C. Zamarreo-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "Multi-casting mesh AER: A scalable assembly approach for eeconfigurable neuromorphic structured AER systems. application to ConvNets," *IEEE Trans. Biomed. Circuits Syst.*, vol. 7, no. 1, pp. 82–102, Feb. 2013.
[22] C. Zamarreo-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 0.35 um Sub-ns wake-up time ON-OFF switchable LVDS driver-receiver chip I/O pad pair for rate-dependent power saving in AER bit-serial links," *IEEE Trans. Biomed. Circuits Syst.*, vol. 6, no. 5, pp. 486–497, Oct. 2012.
[23] C. Zamarreo-Ramos, R. Kulkarni, J. Silva-Martnez, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 1.5 ns OFF/ON switching-time voltage-mode LVDS driver/receiver pair for asynchronous AER bit-serial chip grid links with up to 40 times event-rate dependent power savings," *IEEE Trans. Biomed. Circuits Syst.*, vol. 7, no. 5, pp. 722–731, Oct. 2013.
[24] C. Zamarreo-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "An instant-startup jitter-tolerant manchester-encoding serializer/deserializar scheme for event-driven bit-serial LVDS inter-chip AER links," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 11, pp. 2647–2660, Nov. 2011.
[25] T. Iakymchuk *et al.*, "An AER handshake-less modular infrastructure PCB with x8 2.5Gbps LVDS serial links," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2014, pp. 1556–1559.
[26] L. Plana *et al.*, "spI/O: A library of FPGA designs and reusable modules for I/O in SpiNNaker systems," 2014. [Online]. Available: http://dx.doi.org/10.5281/zenodo.51476
[27] S. Furber *et al.*, "Overview of the SpiNNaker system architecture," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2454–2467, Dec. 2013.
[28] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 × 128 1.5% contrast sensitivity 0.9% FPN 3us latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance amplifiers," *IEEE J. Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, Mar. 2013.
[29] P. Popescu, A. Solheim, and M. Wight, "Experimental monolithic high speed transceiver for Manchester encoded data," in *Proc. 1995 Bipolar/CMOS Circuits Technol. Meeting*, Oct. 1995, pp. 110–113.
[30] P. A. Franaszek and A. X. Widmer, "Byte oriented DC balanced (0,4) 8b/10b partitioned block transmission code," U.S. Patent 4 486 738, Dec. 4, 1984.
[31] M. Jablonski, T. Serrano-Gotarredona, and B. Linares-Barranco, "High-speed serial interfaces for event-driven neuromorphic systems," in *Proc. 2015 Int. Conf. Event-Based Control, Commun. Signal Process.*, Jun. 2015, pp. 1–4.
[32] R. Dobkin and R. Ginosar, "Zero latency synchronizers using four and two phase protocols," VLSI Syst. Research Center, Technion—Israel Inst. Technol., Haifa, Israel, Internal Rep., Oct. 2007.
[33] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Proc. 9th Int. Symp. Asynchronous Circuits Syst.*, 2003, pp. 89–96.
[34] *Spartan-6 FPGA GTP Transceiver, Advanced Product Specifications. UG386 (v2.2)*, Xilinx, San Jose, CA, USA, Apr. 30, 2010.
[35] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE J. Solid-State Circ.*, vol. 46, no. 1, pp. 259–275, Jan. 2011.
[36] T. Delbruck, "jAER Open Source Project," 2007. [Online]. Available: http: //jaer. wiki. sourceforge. net

Amirreza Yousefzadeh (M'11) was born in Tehran, Iran, in 1988. He received the dual B.Sc. degrees in electronic and communication engineering from Amirkabir University of Technology, Tehran Polytechnic, Tehran, Iran, in 2010 and the M.Sc. degree in digital-electronics from Sharif University of Technology, Tehran, in 2013. He is currently working toward the Ph.D. degree at the Instituto de Microelectronica de Sevilla (IMSE-CNM), CSIC University of Seville, Seville, Spain. His research interests include design and VLSI implementation of bioinspired vision processing algorithms.

Mirosław Jabłoński received the M.Sc.Eng. degree in electronics and telecommunication and the Ph.D. degree (Hons.) in automatics and robotics, both from the Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, AGH-University of Science and Technology (UST), Kraków, Poland, in 2001 and 2009, respectively. He worked as a Research and Teaching Assistant in the Department of Automatic Control (2001–2009) and since 2010 as an Assistant Professor in the Department of Automatic Control and Bioengineering, AGH-UST. He was awarded a Postdoc position (2014–2015) at the Sevilla Microelectronics Institute (IMSE-CNM-CSIC). His research interests include parallel implementations of machine vision algorithms and biologically inspired computation by means of FPGAs and GPUs.

Taras Iakymchuk received the M.Sc. diploma degree from the Wroclaw University of Technology, Wroclaw, Poland, in 2011. He is currently working toward the Ph.D. degree at the University of Valencia, Valencia, Spain, in the Digital Signal Processing Group. He was collaborating with research groups from Sevilla, Manchester, and the Institute of Neuroinformatics in Zurich. His main research interests include embedded systems, neural networks, hardware learning, and bioinspired computation.



Alejandro Linares-Barranco (M'04–SM'17) received the B.S., M.S., and Ph.D. degrees in computer science from the University of Sevilla, Sevilla, Spain, in 1998, 2002, and 2003, respectively. From January 1998 to June 1998, he was a Second Lieutenant in the Spanish Air Force working as a System Administrator and a Software Developer. From 1998 to 2000, he was a Member of the Technical Staff at the Sevilla Microelectronics Institute (IMSE-CNM-CSIC). From 2000 to 2001, he was a Development Engineer with the Research and Development Department, at ABENGOA Group, Sevilla. From 2001 to 2006, he was an Assistant Professor in the Computer Architecture and Technology Department, University of Sevilla. In 2006, he was promoted to an Associate Professor. He has been serving as the Director of the Department since 2017. His recent research interests include VLSI for FPGA digital design, neuro-inspired event-based processing, motor control and computer interfaces. In 2013, he became the Chair of the Neural Systems and Applications Technical Committee of the IEEE Circuits and Systems Society. He was a Visiting Professor with the University of Zurich, Zurich, Switzerland, in 2014.

Alfredo Rosado received the M.Sc. and Ph.D. degrees in physics from the University of Valencia, Valencia, Spain, in 1994 and 2000, respectively. He is currently an Associate Professor in the Department of Electronic Engineering, University of Valencia. His work is related to automation systems, digital hardware design (embedded systems) for digital signal processing and control systems, especially targeted for biomedical engineering, and bioinspired systems. He is a member of International Federation of Automatic Control.

Luis A. Plana (M'97–SM'07) received the Ingeniero Electrnico degree from Universidad Simn Bolivar, Caracas, Venezuela, the M.Sc. degree in electrical engineering from Stanford University, Stanford, CA, USA, and the Ph.D. degree in computer science from Columbia University, New York, NY, USA. He was with Universidad Politecnica, Venezuela, for over 20 years, where he was a Professor of Electronic Engineering. He is currently a Research Fellow in the School of Computer Science, University of Manchester, Manchester, U.K.

Steve Temple received the B.A. degree in computer science and the Ph.D. degree in research into local area networks from the University of Cambridge, Cambridge, U.K., in 1980 and 1984, respectively. He was subsequently a Research Fellow with the University of Cambridge, Computer Laboratory. He was a self-employed Computer Consultant from 1986 to 1993, when he took up his current post of Research Fellow in the School of Computer Science, University of Manchester, Manchester, U.K.

Teresa Serrano-Gotarredona (M'07) received the B.S. degree in electronic physics and the Ph.D. degree in VLSI neural categorizers from the University of Sevilla, Sevilla, Spain, in 1992 and 1996, respectively, and the M.S. degree in electrical and computer engineering from The Johns Hopkins University, Baltimore, MD, USA, in 1997. She was an Assistant Professor in the Electronics and Electromagnetism Department, University of Sevilla, from 1998 to September 2000. Since September 2000, she has been a Tenured Scientist at the Sevilla Microelectronics Institute (IMSE-CNM-CSIC), Sevilla, and in July 2008 she was promoted to a Tenured Researcher. She has been a Visitng Professor/Fellow at Texas A&M University College-Station, College-Station, TX, USA, The University of Manchester, Manchester, U.K., and the University of Lincoln, Lincoln, U.K. Her recent research interests include AER vision chips, Real-time vision sensing and processing chips, and nanoscale memristor-type AER systems. She received two IEEE Transactions Best Paper Awards, and has been an Associate Editor of the *PLoS ONE Journal* and the IEEE TRANSACTIONS IN CIRCUITS AND SYSTEMS PART II. She is currently an Associate Editor the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-I. From 2013 to 2015, he was the Chair of the IEEE Circuits and Systems Society Spain Chapter.

Steve B. Furber (F'05) was born in Manchester, U.K., in 1953. He received the B.A. degree in mathematics and the Ph.D. degree in aerodynamics from the University of Cambridge, Cambridge, U.K., in 1974 and 1980, respectively, and the Honorary Doctorate degrees from Edinburgh University, Edinburgh, U.K., in 2010, and Anglia Ruskin University, Cambridge, U.K., in 2012. From 1978 to 1981, he was a Rolls Royce Research Fellow in Aerodynamics at Emmanuel College, Cambridge, U.K., and from 1981 to 1990, he was at Acorn Computers Ltd., Cambridge, U.K., where he was a Principal Architect of the BBC Microcomputer, which introduced computing into most U.K. schools, and the ARM 32-bit RISC microprocessor, over 100 billion of which have been shipped by ARM Ltd.'s partners. In 1990, he moved to the ICL Chair in Computer Engineering at the University of Manchester, Manchester, U.K. His research interests include asynchronous digital design, low-power systems on chip, and neural systems engineering. He is a Fellow of the Royal Society, the Royal Academy of Engineering, the British Computer Society, the Institution of Engineering and Technology, and the Computer History Museum (Mountain View, CA). He was a Millennium Technology Prize Laureate (2010) and holds an IEEE Computer Society Computer Pioneer Award (2013).

Bernabé Linares-Barranco (M'94–F'10) received the B.S. degree in electronic physics, the M.S. degree in microelectronics, and the first Ph.D. degree from the University of Sevilla, Sevilla, Spain, in 1986, 1987, and 1990, respectively, and a second Ph.D. degree from Texas A&M University, College Station, TX, USA, in 1991. Since 1991, he has been with the Sevilla Microelectronics Institute (IMSE-CNM), from the Spanish Research Council (CSIC) of Spain, where he currently is a Full Professor of Research. He has been a Visiting Professor/Fellow at The Johns Hopkins University, Baltimore, MD, USA, Texas A&M University, College Station, The University of Manchester, Manchester, U.K., and the University of Lincoln, Lincoln, U.K. His recent interests include address-event-representation VLSI, real-time AER vision sensing and processing chips, memristor circuits, and extending AER to the nanoscale. He received two IEEE Transactions Best Paper Awards, and has been an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-II, the IEEE TRANSACTIONS ON NEURAL NETWORKS, and *Frontiers in Neuromorphic Engineering*. From 2011 to 2013, he was the Chair of the IEEE Circuits and Systems Society Spain Chapter.