Fast Predictive Handshaking in Synchronous FPGAs for Fully Asynchronous Multisymbol Chip Links: Application to SpiNNaker 2-of-7 Links

Amirreza Yousefzadeh, Luis A. Plana, Steve Temple, Teresa Serrano-Gotarredona, *Member, IEEE*, Steve B. Furber, *Fellow, IEEE*, and Bernabé Linares-Barranco, *Fellow, IEEE*

***Abstract*—Asynchronous handshaken interchip links are very popular among neuromorphic full-custom chips due to their delay-insensitive and high-speed properties. Of special interest are those links that minimize bit-line transitions for power saving, such as the two-phase handshaken non-return-to-zero (NRZ) 2-of-7 protocol used in the SpiNNaker chips. Interfacing such custom chip links to field-programmable gate arrays (FPGAs) is always of great interest, so that additional functionalities can be experimented and exploited for producing more versatile systems. Present-day commercial FPGAs operate typically in synchronous mode, thus making it necessary to incorporate synchronizers when interfacing with asynchronous chips. This introduces extra latencies and precludes pipelining, deteriorating transmission speed, particularly when sending multisymbols per unit communication packet. In this brief, we present a technique that learns to estimate the delay of a symbol transaction, thus allowing a fast pipelining from symbol to symbol. The technique has been tested on links between FPGAs and SpiNNaker chips, achieving the same throughput as fully asynchronous synchronizerless links between SpiNNaker chips. The links have been tested for periods of over one week without any transaction failure. Verilog codes of FPGA circuits are available as additional material for download.**

***Index Terms*—Address event representation (AER), asynchronous links, event-driven links, field-programmable gate arrays (FPGAs), neuromorphic chips, synchronization.**

Manuscript received November 16, 2015; accepted February 8, 2016. Date of publication February 18, 2016; date of current version July 28, 2016. This work was supported by the European Research Council under the European Union's (EU) Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement 320689, by the U.K. Engineering and Physical Sciences Research Council under Grant EP/G015740/1, by the EU "The Human Brain Project" under Grant FP7-604102, by the EU Event-Driven Compressive Vision for Multimodal Interaction Modal Devices (ECOMODE) Project under Grant H2020-644096, and by the Spanish Ministry of Economy and Competitivity through the European Regional Development Fund under Grant TEC2012-37868-C04-01 (BIOSENSE). The work of A. R. Yousefzadeh was supported by a Formación de Personal Investigador Scholarship from the Spanish Ministry of Economy and Competitivity. This brief was recommended by Associate Editor X. Liu.

A. Yousefzadeh, T. Serrano-Gotarredona, and B. Linares-Barranco are with the Instituto de Microelectrónica de Sevilla (IMSE-CNM), Consejo Superior de Investigaciones Científicas (CSIC) and Universidad de Sevilla, 41092 Sevilla, Spain (e-mail: bernabe@imse-cnm.csic.es).

L. A. Plana, S. Temple, and S. B. Furber are with the Advanced Processor Technologies Group, School of Computer Science, The University of Manchester, Manchester M13 9PL, U.K.

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

Digital Object Identifier 10.1109/TCSII.2016.2531092

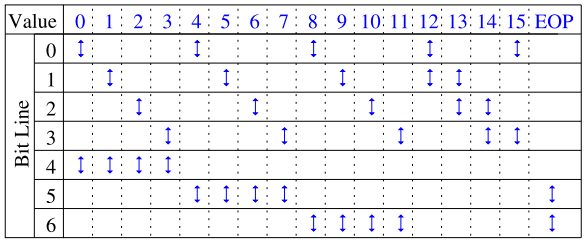

Fig. 1. Encoding of 2-of-7 NRZ protocol transitions to 4-bit symbol values.

I. Introduction

NEUROMORPHIC chips and systems use typically the asynchronous four-phase handshaken address event representation (AER) scheme to interchange information in an event-driven manner [1], [2] for vision systems [3], [4] and robotics [5]. The recently available multi-ARM-core SpiNNaker chips [6] (intended for simulating large-scale neuromorphic systems) use a special multisymbol very low-power two-phase-handshaking non-return-to-zero (NRZ) protocol [7], which is called 2-of-7 [8]. Each link is unidirectional and uses eight lines (seven for data and one for Ack, i.e., *Acknowledge*). A symbol is transmitted by changing the state of two data lines only, which signals a *Request* for the handshaking. Although there are 21 possible transitions in two lines out of seven lines, only 17 are used by SpiNNaker (16 data symbols and one "End-of-Packet" symbol). This way, data symbols can be represented by 4-bit nibbles, as illustrated in Fig. 1. SpiNNaker chips can communicate a packet (also called "event") of either short format (44-bits) with 11 4-bit symbols or long format (76-bits) with 19 4-bit symbols.

The structure of a packet/event is 8-bit header, 32-bit data, 32-bit optional payload (extra data for the long format), and "End-of-Packet" symbol. Fig. 2 shows a commercial SpiNN5 board hosting 48 SpiNNaker chips. Each chip connects to six neighbor chips (north, south, east, west, northeast, and southwest), emulating a hexagonal grid [6]. Each chip-to-chip connection contains a pair of 8-bit 2-of-7 lines, one for each direction. Interchip links (which need minimum PCB trace length) can exchange short-format events at a rate of about 6 Meps (mega events per second), which accounts to about 15 ns per symbol transaction. On the top of the board, one can see three Spartan6 field-programmable gate arrays (FPGAs).

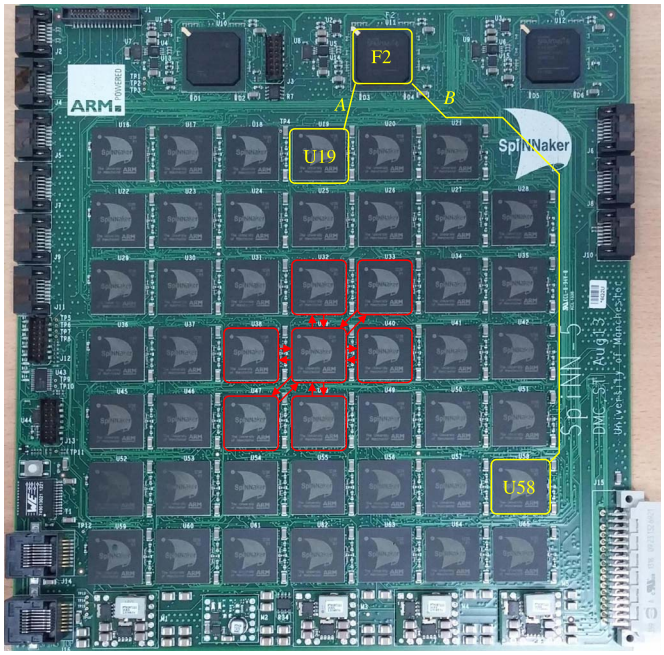

Fig. 2. SpiNN5 board with 48 SpiNNaker chips. Lines in red illustrate inter-chip 2-of-7 links. Lines in yellow show one bidirectional 2-of-7 communication link between a Spartan6 FPGA and one of the SpiNNaker chips.

They connect to some of the SpiNNaker chips through a bidirectional pair of 8-bit 2-of-7 NRZ asynchronous links. On the top in Fig. 2, we highlight two of such links: link A between the central FPGA $F2$ and the SpiNNaker chip $U19$ on the top row, and link B between the same FPGA and the chip $U58$ far away and close to the bottom right edge. The circuitry inside the FPGA is clocked and requires the use of synchronizers to interface properly with external asynchronous links [9], [10]. In the next section, we briefly describe how such a standard link would operate, achieving a maximum average throughput of 3.51 Meps from the FPGA to the SpiNNaker chip. Afterward, in Section III, we present the new proposed approach, which can reach up to 6.89 Meps from the FPGA to the SpiNNaker chip. Our proposed approach could not be used to improve the speed of the reverse direction (from SpiNNaker chip to FPGA) because it would require modifying the SpiNNaker chip itself. Nonetheless, the proposed scheme can be included in future versions of the SpiNNaker chip.

II. Conventional Synchronization Approach

Figs. 3 and 4 show the diagram and the timing between an FPGA transmitter (TX) link side and a SpiNNaker chip receiver (RX) link side,[1] using a conventional two-D-flip-flop synchronization, respectively. Short 32-bit (or long 64-bit) events are provided to a finite-state machine (FSM), which will convert them to the 11 (or 19) 4-bit symbol sequence, loading each into the 4-bit register in Fig. 3. After this, an "Encoder" activates the 2-of-7 bits that need to change, according to Fig. 1, which, after being "XOR-ed" with the previous output, provides the new

[1]The complementary link from SpiNNaker TX to FPGA RX is not shown to save space.

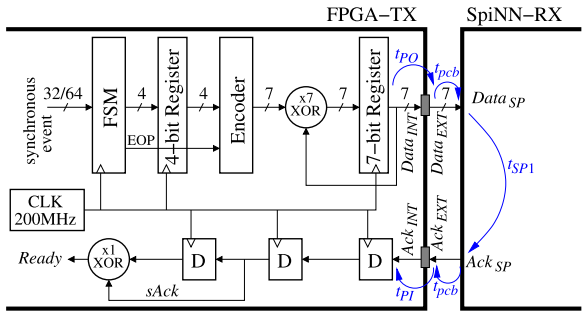

Fig. 3. Simplified block diagram illustration of an FPGA-TX-to-SpiNNaker-RX directional link.

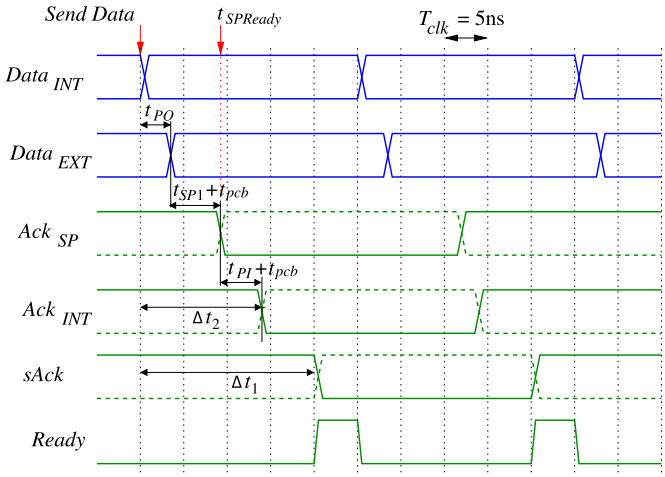

Fig. 4. Timing waveforms illustrating NRZ handshaking and synchronization.

output storing it in an output 7-bit register. This register holds the new 2-of-7 Data-and-Rqst Data_{INT}. Once it is available, it requires a delay due to output pad buffering I/O t_{PO} to go out of the FPGA as Data_{EXT}, plus an interchip PCB trace delay of t_{pcb} to be visible at the SpiNNaker chip input. The SpiNNaker chip RX port requires a delay, which is here called t_{SP1}, between detecting the 2-of-7 Data-and-Rqst until providing its acknowledge signal Ack_{SP}, which after t_{pcb} will make Ack_{EXT} visible at the FPGA external input. The FPGA input pad introduces an additional delay t_{PI} until the asynchronous acknowledge signal Ack_{INT} is visible internally inside the FPGA. After this, the synchronization circuit using a standard two-D-flip-flop delay line, requires two additional clock edges to make a synchronized version of the acknowledge signal sAck available. At this point in time, a new Data_{INT} value can be made available for the next clock edge. From the Spartan6 FPGA manufacturer specifications, we know that $t_{\text{PI}} \approx 1.2$ ns and that t_{PO} may vary between 1.7 and 5.9 ns, depending on output pad settings. For our settings, $t_{\text{PO}} \approx 3.0$ ns. In Fig. 4, we can observe that, if $\Delta t_2 = t_{\text{PO}} + t_{\text{SP1}} + 2t_{\text{pcb}} + t_{\text{PI}}$ is between two and three clock cycles (10–15 ns), then a full symbol transaction can be done in five clock cycles (25 ns). If 10 ns $< \Delta t_2 <$ 15 ns, then it implies that 5.8 ns $< t_{\text{SP1}} + 2t_{\text{pcb}} <$ 10.8 ns. Otherwise, if 10.8 ns $< t_{\text{SP1}} + 2t_{\text{pcb}} <$ 15.8 ns, then a symbol transaction would require six clock cycles (30 ns). Implementing the link in Fig. 4 on link A in Fig. 2 results in five-cycle transactions, while on link B, this results in six

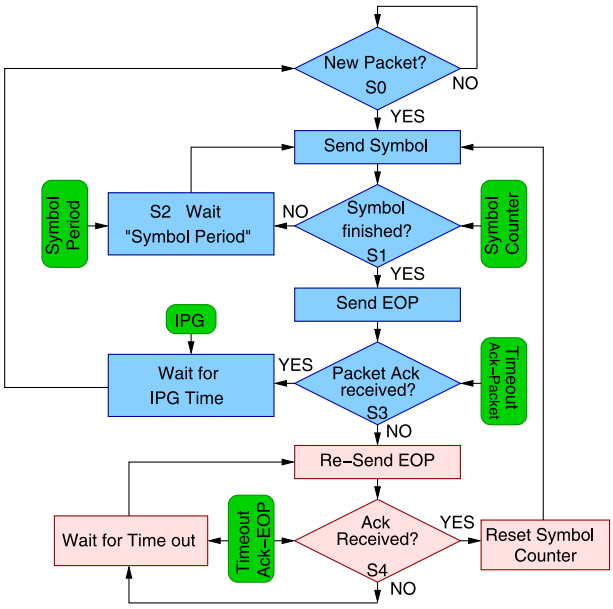

Fig. 5. Simplified flow diagram of new transmitter FSM. Blue (medium gray) boxes correspond to failure-free packet transmission and pink (light gray) boxes to failure handling and packet retransmission, and green (dark gray) boxes indicate parameters.

cycles. In summary, for the 48-chip PCB in Fig. 2, a symbol transaction varies between five and six clock cycles, depending on the length of PCB traces. In the next section, we propose a method to reduce this time down to two clock cycles for both links. It requires changing the FSM in Fig. 3, i.e., the sender of the link. Therefore, this means that we could only test it by changing the FSM at the FPGA (the sender side). Consequently, we present results only for the case of transmitting data from the FPGA to the SpiNNaker chip.

Time t_{SP1} is typically quite stable for each SpiNNaker link, except for the cases when the interchip circuitry is sending back pressure (i.e., delaying Ack) because of internal traffic saturation.

III. Proposed Predictive Synchronization Scheme

The herein proposed new synchronization scheme is based on the following observation in Fig. 4. The SpiNNaker RX side of the link is, in principle, ready to receive a new 2-of-7 Data-and-Rqst, as soon as it has provided acknowledge signal $\mathrm{Ack_{SP}}$ at time t_{SPReady}. However, due to the synchronization with two D-flip-flops on the FPGA side, the FPGA cannot provide a new $\mathrm{Data_{INT}}$ until four clock edges later. Here, we propose a scheme where the FPGA "learns" to forecast, reliably, the minimum number of clock cycles required to send a new symbol (without waiting to receive each symbol's synchronized acknowledge signal sACK). Nonetheless, during the multisymbol transmission of a packet, an independent process in parallel would count the total number of actual acknowledge signals received during the full packet to make sure that the full packet transaction was completed successfully.

The new proposed algorithm for the transmitter FSM in Fig. 3 is shown in Fig. 5. This FSM will send out the 11 (or 19) event/packet symbols without waiting for individual Acks from the receiver side. It will simply wait for a "Symbol Period" time (i.e., number of clock cycles) before sending the next symbol. A parallel process (not shown in the figure) will be counting the number of Ack signals received and generating an internal "Packet Ack" signal once all of them are received. The operation of the FSM in Fig. 5 is as follows. The first state S0 waits for a new (32- or 64-bit) event/packet. After this, the corresponding sequence of symbols must be sent. For this, an extra state S2 is included, which waits for a given number of clock cycles ("Symbol Period") before sending the next symbol. After sending all header and data symbols, the "End-of-Packet" (EOP) symbol is also sent. After this, the FSM enters state S3, where it waits for the internal "Packet Ack" signal. This signal is triggered only if the receiver has acknowledged all symbols sent. Once "Packet Ack" is received, an optional "Inter-Packet Gap" (IPG) wait time can be included to allow the receiver some extra time for event/packet processing. In case not all symbol Acks have been received within a given "Time-out" period, state S3 will branch out through its "Packet NOT sent" output, indicating there has been a failure in the event/packet transmission. In this case, it will try to resend the event/packet. For this, it will first send an EOP symbol to the receiver and wait for the corresponding Ack, through state S4. If this Ack is not received, then it will wait for some time to let the receiver recover and, after this, retry the transmission of an EOP symbol. This situation may occur in case the SpiNNaker internal event handling circuitry is sending back pressure (because of event traffic saturation), or there is a transient fault/disconnection in the transmission line.

At startup, there is a "learning process" in which the FSM adjusts its parameter "Symbol Period." Initially, this period is set to "1" (one clock cycle), and it will be increased progressively until reaching a stable communication. For each "Symbol Period" value, two weights are defined. The first weight is the rate of failure, and the second is the rate of success. Every time state S3 leaves through its "Yes" output, the success weight is increased. If state S3 leaves through its "NO" output, the failure counter is increased. During learning, the "Time-out" parameter values are reduced to speed up learning. The rate of convergence of this startup learning process is relatively fast, although it also depends on weight granularity and initial state. In our case, we used 8-bit weights, and both of them (success and failure) were initially set to "0." Convergence time was on the order of 500 μs. After this, no more failures were detected, even when running the links for over one week. If there are transient faults during the startup learning process, it will converge to very conservative Ack/Rqst intervals. Therefore, during startup, the system and all physical connections should be in optimum conditions.

So far, we have discussed the situation of sending events from the FPGA (as TX) to the SpiNNaker chip (as RX). In this case, it is the TX who learns to forecast the intersymbol delay, and also who detects whenever an event/packet has not been sent. In order to implement this forecasting/acceleration capability for the reverse direction without changing the SpiNNaker chip, we would need to change the RX side in the FPGA. For this, the receiver in the FPGA would need to send out Acks before obtaining the synchronized versions of the 2-of-7 Data-and-Rqst

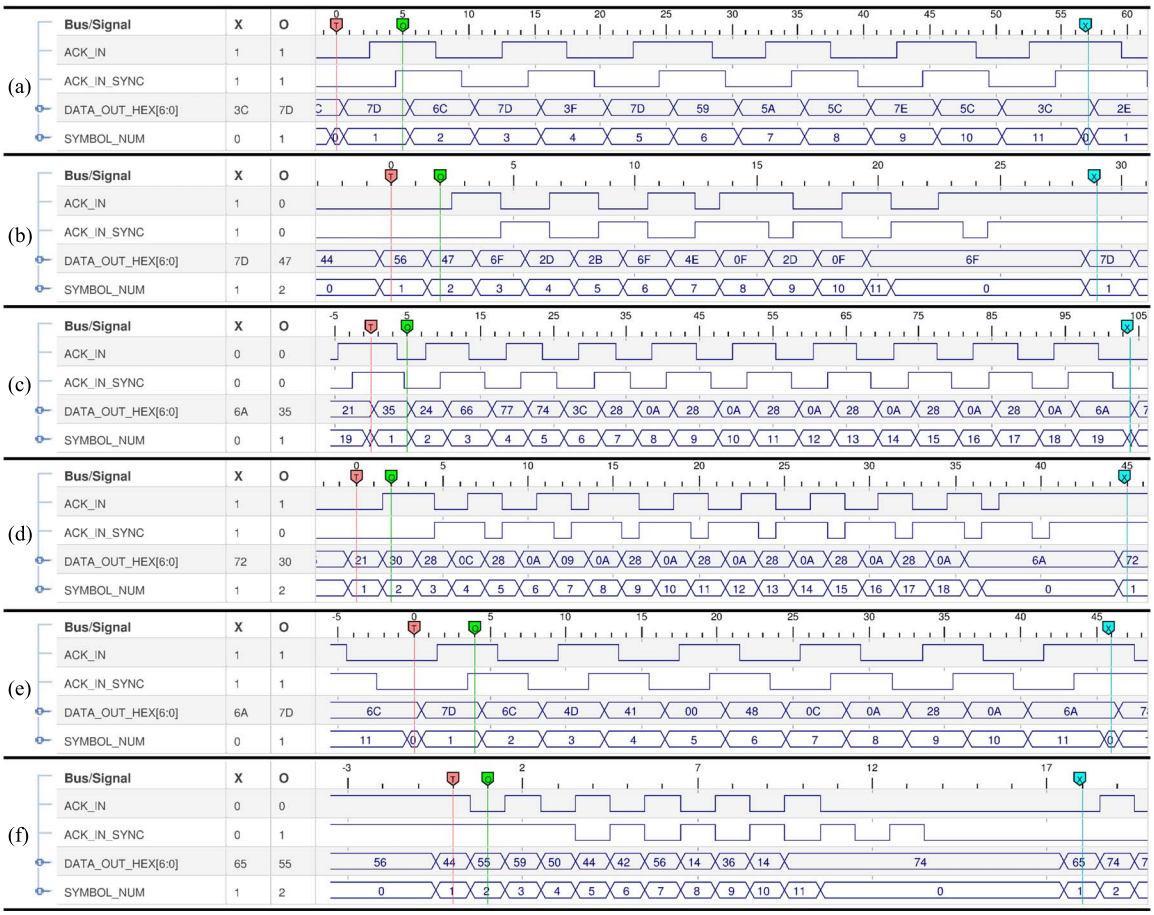

Fig. 6. ChipScope measurements for link A with FPGA pads set to SLOW and 6-mA power per pad. (a) Normal synchronization; short packet; 200-MHz clock. (b) Fast scheme; short packet; 200-MHz clock. (c) Normal synchronization; long packet; 200-MHz clock. (d) Fast scheme; long packet; 200-MHz clock. (e) Normal synchronization; short packet; 100-MHz clock. (f) Fast scheme; short packet; 100-MHz clock. Note that, in (b), ACK_IN and ACK_IN_SYNC are not exact replicas with two-clock-cycle delay. This is because ACK_IN was captured before the synchronizer and its edge must have been very close to that of the ChipScope clock.

transitions. In case of failure, only the RX circuit in the FPGA would be aware of it, and the TX in the SpiNNaker chip would not be able to resend the event/packet. There are three obvious approaches for solving this. First, add an extra 2-of-7 command to the table in Fig. 1, so that the FPGA can request the retransmission of an event/packet. Second, implement this new algorithm inside the SpiNNaker chip in its TX ports. Or third, implement a slower upper layer in software to detect event/packet loss and request a new retransmission. The first two options require a redesign of the SpiNNaker chip, and we leave this as suggestions for future versions. The third solution is beyond the scope of this brief. In the next section, we provide experimental results for the link direction from the FPGA (TX) to the SpiNNaker chip (RX).

IV. Experimental Results

Exhaustive tests have been performed on the 48-chip SpiNNaker PCB shown in Fig. 2, to test the performance of packet/event communication from an FPGA to a SpiNNaker chip. The results shown here focus on two of such links: "Link-A" between FPGA "F2" and SpiNNaker chip "U19," which is one of the shortest links on the PCB, and "Link-B" between the same FPGA and chip "U58," which is one of the longest links. Experimental characterizations were performed by generating sequences of numbers with a counter on one end and checking the sequence on the other end. Failure-free transmissions were obtained after a few hundred microseconds of training, which would stay failure free for long periods (we tested for over one week). Experimental measurements were done through the use of Xilinx's built-in logic analyzer module "ChipScope." This tool allows monitoring FPGA internal signals with reference to its internal clock. For our experiments, we have set this internal clock to either 200 MHz (5-ns period) or 100 MHz (10-ns period). Fig. 6 shows ChipScope screen captures for different measurements. For each measurement, we show the same four signals: signal ACK_IN, which corresponds to Ack_{INT} in Fig. 4; ACK_IN_SYNC, which is sAck in Fig. 4; DATA_OUT_HEX, which is Data_{INT} in Fig. 4; and SYMBOL_NUM (not shown in Fig. 4), which counts the symbol number within the packet/event. On the top of each subfigure, the ticks indicate clock cycle number.

Fig. 6(a) illustrates the case of using the conventional synchronization approach on link A for a short package with a 200 MHz clock. As can be seen, to transmit all 11 symbols, 57 clock cycles are needed, which corresponds to 3.51 Meps

		Link A								Link B							
		Short Packet				Long Packet				Short Packet				Long Packet			
		200 MHz		100MHz		200 MHz		100MHz		200 MHz		100MHz		200 MHz		100MHz	
		Normal	Fast	Normal	Fast	Normal	Fast	Normal	Fast	Normal	Fast	Normal	Fast	Normal	Fast	Normal	Fast
fast	Pck Rt	3.51	6.89	2.17	5.55	2.06	4.44	1.28	3.7	2.72	6.66	2.17	5.55	1.72	4.34	1.28	3.7
	Nc Pck	57	29	46	18	97	45	78	27	73	30	46	18	116	46	78	27
	Nc Sym	5	2	4	1	5	2	4	1	6	2	4	1	6	2	4	1
Slow	Pck Rt	3.51	6.89	2.17	5.55	1.96	4.44	1.28	3.7	2.53	6.66	2.17	5.55	1.62	4.34	1.2	3.7
	Nc Pck	57	29	46	18	102	45	78	27	79	30	46	18	125	46	83	27
	Nc Sym	5	2	4	1	5	2	4	1	6	2	4	1	6	2	4	1
quiet	Pck Rt	2.94	6.66	2.17	5.55	1.72	4.34	1.28	3.7	2.19	4.76	1.75	3.44	1.34	3.07	1.03	2.22
	Nc Pck	68	30	46	18	116	46	78	27	91	42	57	29	149	65	97	45
	Nc Sym	6	2	4	1	6	2	4	1	7	3	5	2	7	3	5	2

Fig. 7. Measured parameters are "Pck Rt" packet rate (in mega events per second), "Nc Pck" number of clock cycles per packet, and "Nc Sym" number of clock cycles per symbol. Vertical columns show measurements for links A and B, for short packets (11 symbols) and long packets (19 symbols), at 200-MHz and 100-MHz FPGA clock frequencies, for "Normal" (conventional) synchronization scheme and for "Fast" predictive handshaking scheme. Horizontal rows are repeated for three different FPGA output pad bias settings ("current" and "slew-rate"): "fast" is 12 mA per pad with nominal 1.71-ns delay, "slow" is 6 mA per pad (recommended) with nominal 3.00-ns delay, and "quiet" is 2 mA per pad with 5.47-ns delay.

(mega events per second). Transmission of one symbol requires five clock cycles. Fig. 6(b) shows the same case, but when implementing the predictive handshaking approach. As can be seen, one 11-symbol packet needs now only 29 clock cycles, which corresponds to 6.89 Meps. Each symbol can be reliably transmitted with only two clock cycles (see signals DATA_OUT_HEX and SYMBOL_NUM), although now there is an extra seven-cycle overhead after transmitting all symbols. Interestingly, the parallel independent process in charge of counting the transitions at ACK_IN normally needs two clock cycles, although sometimes it needs one or three (see signals ACK_IN and ACK_IN_SYNC). Fig. 6(c) and (d) illustrates the same setup as Fig. 6(a) and (b) but for a long 19-symbol packet. Similarly, Fig. 6(e) and (f) shows the same as Fig. 6(a) and (b) but setting the clock to 100 MHz. This is to illustrate the situation for a slower FPGA. As can be seen, for the conventional synchronization approach, four clock cycles per symbol are required (instead of five) and 46 per packet (instead of 57). This is because the fixed delay Δt_2 in Fig. 4 is framed into less clock cycles. However, for the same reason, in the predictive handshaking approach, one symbol can be transmitted now in just one clock cycle. On the other hand, the overhead requires the same eight clock cycles; thus, the overall delay for a short-packet transaction is 18 cycles, resulting in a speed improvement factor of 2.55.

Fig. 7 shows the measured packet/event rate (Pck Rt) and the number of clock cycles per symbol (Nc Sym) and per packet/event (Nc Pck) for all experimental setups: for links A and B, for 100-MHz and 200-MHz clock frequencies, for short and long packets, and also for three different settings of the FPGA output pads (setting SLOW with 6 mA per pad, which corresponds to all cases shown in Fig. 6; setting FAST with 12 mA per pad, and setting QUIET with 2 mA per pad). The packet transaction speed improvement varies between a factor of about 2 (1.96 for link A, short packet, 200 MHz) up to a factor of almost 3 (2.89 for link A, long packet, 100 MHz). The Verilog codes used for these setups are provided as additional material for download.

V. Conclusion

A scheme for accelerating asynchronous handshaken multisymbol packet transmissions between an asynchronous module and a synchronous one has been proposed and successfully tested on a 48-chip SpiNNaker board. The scheme exploits the fact that, within the same packet, the transaction delay per symbol remains stable and can be "learned" by the sending circuit. Symbol Acks are counted by a separate process in parallel to verify correct packet transmission. In case of failure, the packet is resent. Exhaustive tests have been performed on the 48-chip SpiNNaker board for different PCB trace lengths, packet sizes, clock frequencies, and pad delays. Once trained, the transmission stays stable and failure free. The proposed scheme can help to improve the traffic bottleneck between SpiNNaker and PCBs, as this bandwidth is limited by the throughput between FGPA and SpiNNaker chips on board.

References

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

[2] F. Stefanini, E. Neftci, S. Sheik, and G. Indiveri, "PyNCS: A microkernel for high-level definition and configuration of neuromorphic electronic systems," *Frontiers Neuroinform.*, vol. 8, no. 73, pp. 1–14, Aug. 2014.

[3] X. Lagorce, S.-H. Ieng, X. Clady, M. Pfeiffer, and R. Benosman, "Spatiotemporal features for asynchronous event-based data," *Front. Neurosci.*, vol. 9, no. 46, Feb. 2015.

[4] G. Orchard *et al.*, "HFirst: A temporal approach to object recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 10, pp. 2028–2040, Oct. 2015.

[5] J. Conradt, F. Galluppi, and T. C. Stewart, "Trainable sensorimotor mapping in a neuromorphic robot," *Robot. Auton. Syst.*, vol. 71, pp. 60–68, Sep. 2015.

[6] S. Furber *et al.*, "Overview of the SpiNNaker system architecture," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2454–2467, Dec. 2013.

[7] I. Sutherland, "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, Jun. 1989.

[8] L. Plana *et al.*, "A GALS infrastructure for a massively parallel multiprocessor," *IEEE Des. Test Comput.*, vol. 24, no. 5, pp. 454–463, Sep. 2007.

[9] Z. Zhang, "Performance analysis of synchronization circuits," Ph.D. dissertation, School Comput. Sci., Univ. Manchester, Manchester, U.K., 2010.

[10] Y. Li, B. Nelson, and M. Wirthlin, "Synchronization techniques for crossing multiple clock domains in FPGA-based TMR circuits," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 6, pp. 3506–3514, Dec. 2010.