An Asynchronous Quasi-Delay-Insensitive Bit-Serial Variable-Length Relative Address-Event Codec for Distributed Neuromorphic Systems

Simeon Bamford[1], Ole Richter[2,3], and Chiara Bartolozzi [1]

[1]Event-Driven Perception for Robotics, Italian Institute of Technology, via San Quirico 19D, Genova, Italy
[2]Embedded Systems Engineering, DTU Compute, Technical University of Denmark
[3]Asynchronous VLSI and Architecture Group, School of Engineering and Applied Science, Yale University, USA

***Abstract*—To distribute neuromorphic sensing and spiking computation across different sensing or computing units, there is the need for an event communication protocol that supports the chaining together of arbitrarily many units, so that complex systems can be constructed and modified without fully specifying the design in advance. In embedded applications, such protocol should support serial communication, to minimise the use of wires. We have designed an asynchronous codec and we demonstrate its conformance to quasi-delay-insensitive design principles. The codec as shown here works with address-events which carry a one-bit payload (such as the polarity of a physical change that evokes a sensor event) and could be extensible to different payloads. The address encoding is relative to the sender or receiver, address-events are transmitted in a bit-serial manner, and the length of an address encoding is variable, such that there is no upper limit to the virtual address space. This enables scalable and economical sensory designs in flexible electronics, templated and dense ASIC arrays, and multi-chip composed sensors, by enabling design reuse on all levels.**

I. Introduction

Neuromorphic engineering promises benefits in edge sensing and computing which include low power operation, low latency sensing, and robust real-world perception. Sensorising and controlling robotic bodies or multi-agent systems are promising applications [1].

For building more capable and intelligent systems, scalability is a key factor. Neuromorphic systems consist in large numbers of computational elements often distributed, which communicate by spikes or events similar to biological neural systems [2]–[4]. In contrast to biological systems, where dedicated axons connect neurons and the connectivity does not require multiplexing, the communication infrastructure supporting the flexible routing of spikes is delivered via time-division multiplexing and typically carry information about the identity of the sending elements (addresses) [5]. Given the speed of signals across metal wires with respect to the conduction speed of axons and the time constants of neural activity, the time-multiplexing paradigm can achieve similar capabilities to biological systems on a fixed and static 2-dimensional substrate. Although the spiking activity of the sensing and computing elements of neuromorphic systems is asynchronous, communication requires either clocked [6] or handshaking implementations [7], [8] and the data transmission can come in different implementation flavors, such as parallel [7], semi-parallel [9], [10] or serial [6], [11].

Most protocol implementations embed assumptions about the system architecture; e.g., in parallel implementations [12] the bus width, fixed at design time, limits the number of address bits that can be delivered and with it the maximum system size, and the trade-off between system size and wiring. To achieve distributed neuromorphic sensing (and computation) with both superior design and post fabrication scalability, there is the need for a communication protocol implementation that supports the chaining together of arbitrarily many units, so that complex and flexible systems can be constructed and modified without changes to a unit design.

It is advantageous to minimise the need for synchronisation between communicating units to enable the highest degree of parallel operation possible. We have therefore designed an asynchronous serial address-event encoder and decoder circuits with relative and dynamical addressing. The proposed codec is serial, to limit the wiring towards its integration on robotic platforms and highly space-limited sensory arrays. It supports the inclusion of additional units by inherently adapting the addressing space to the number of connected units without redesign and fabrication. The routing fabric does not therefore impose an address-space constraint as observed in most implementations [2], [3]. It is based on quasi-delay insensitive handshaking, to avoid the need for synchronisation across units while maintaining optimal performance and flexible wire length.

CB acknowledges the financial support of the National Biodiversity Future Center funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.4 - Call for tender No. 3138 of 16 December 2021, rectified by Decree n.3175 of 18 December 2021 of Italian Ministry of University and Research funded by the European Union – NextGenerationEU. CB and SB acknowledge the financial support from PNRR MUR Project PE000013 ”Future Artificial Intelligence Research (hereafter FAIR)”, funded by the European Union – NextGenerationEU. SB acknowledges helpful discussions with Rajit Manohar and Ned Bingham.

A. Background - Asynchronous digital communication and quasi-delay-insensitive logic

In mainstream digital microelectronics, a clock signal is distributed widely across a computational core, or an entire chip, and is used to drive the rhythm of computation. Local circuits each compute a result based on their inputs and then wait until the clock pulse arrives in order to output the result of their computation. This design style greatly simplifies the problem of designing logic circuits to perform complex computation by limiting the speed to the slowest element in the system. When scaling flexible and fine grained parallel systems synchronization becomes a significant design constraint resulting in large circuit overhead or significantly degraded system performance [13].

Asynchronous circuits avoid the above scaling problems by design [14]. By synchronising computation and data transmission only with direct communication partners, a fine-grained, distributed parallel system is formed. Communication between the elements is coordinated by handshaking, which is performed using request and acknowledgement signals [15]. [1].

While synchronous systems need to guarantee timing relative to a clock, asynchronous systems need to ensure the timing order of handshaking signals to guarantee correct operation [16]. In flexible or distributed systems where the signal wiring delay can not be precisely estimated at component design time, a specific encoding 1-of-n or dual-rail encoding can be used. By merging the data and handshaking signal on the same wires, the circuit becomes delay-insensitive and will function under all delay conditions. Historically, to enable practical complex computations [17] the 'isochronous fork assumption' was introduced and became known as Quasi-Delay-Insensitive (QDI) logic [14]. The delay insensitivity comes at a price, the dual-rail encoding introduces significant wire, area and power consumption overhead in standard bit-parallel circuit design. Bit-serial design can mitigate this overhead by fixing the number of wires and gates are used while maintaining the QDI properties.

B. Background - Event-based communication in neuromorphic systems

Event-based communication is a fundamental concept in neuromorphic computing, which is inspired by the way the brain transfers information using spikes. Spikes are generated by individual neurons in response to stimuli, such as changes in illumination triggering cells in the retina, the skin, or the cochlea, and are transmitted between neurons to carry information. While biological neurons communicate spikes along dedicated transmission lines (axons and dendrites), the same can not be implemented in a fixed 2-dimensional substrate with limited I/O ports. To solve this issue, Sivilotti [8] demonstrated spike-generating circuits sending requests to peripheral circuitry, which acknowledged them; events were queued in production order and sent off-chip via a time-multiplexed shared bus using asynchronous digital communication.

These systems, subsequently called Address-Event Representation (AER), output packets that encode the binary address of the sender (neuron or sensory element) of each event. Because microelectronic timescales far exceed biological ionic ones, the propagation delay of the address-events were considered negligible, thus the systems operated on the principle of "time represents itself". To increase throughput these systems are highly pipelined.

Although the first AER implementations were not QDI, later work established this as a defacto standard [18], to avoid manual timing closure. AER now refers to a large class of protocols which move around address-events within the context of arrays of sensing and processing elements ('neurons'). Wiring space may be plentiful inside an ASIC but in dense arrays, multi-ASIC compositions or interchip connections, bus size becomes an important consideration.

A challenge arises between the optimal function of encoding and decoding physical arrays, which are theoretically optimal as a binary or custom tree [11], [19], and the tiling capability of the system, which follows a line, grid or mesh [20]. Making each element as close to identical as possible is particularly important in systems with many end points or leaves. Limited space for wires and repeatable fixed elements or sensor elements require fixed size parallel buses [21] or special serial links. In densely packed one ([2], [22]) and two ([9], [23]–[25]) dimensional arrays, a classic global encoder or decoder (P-AER) [7] comprising of a single pipeline stage, is the most widespread.

Many variants have been developed: Address-events are delivered in two or more words [10]; serialised into single bits [26] to transmit words with variable lengths enabling unbounded address spaces; furthermore arithmetic may be performed on these variable-length serialised addresses [27]. Routing and mapping systems have been developed to transform the transmitted address [28] and events to 'fan out', i.e. to be received by multiple receivers [29], [30]. Addresses of receiving neurons can be treated as relative addresses or routing instructions in a potentially infinite space of neurons [2].

The system described in this manuscript uses incrementation of digital addresses in encoding (and decrementation in decoding). A variable-length bit-serial encoding allows devices to be arbitrarily long without requiring redesign, making it flexible to be deployed in a wide range of sensory and neural processing applications.

II. Methods

A. Target system

This AER implementation has been developed for a one-dimensional array of sensory cells, each of which can at any moment produce a digital event in response to a change in a physical stimulus, such as tactile pressure, light intensity, chemical concentration, etc. As a physical stimulus can either increase or decrease, each event carries a data payload of one bit, the event polarity, that distinguishes the direction of change

[1] These signals are sometimes referred to as 'ready' and 'valid'.

(these polarities are denoted hereafter as 'a' and 'b' [2], and the address-events can be described as 'polarised'). More broadly, the system we propose is agnostic to the type of circuit that generates events, being a neuron in a spiking neural network, or a sensing element in a sensor array. However here we present designs and results only for polarised address-events. It's possible to reuse these designs directly by treating the polarity as the least significant bit of a system of addresses, but whether the events come from a sensory cell or a pair of neurons, the circuits presented assume that the events are produced in a mutually exclusive manner - once an event source has produced an event of a given polarity, by activating one of two request signals, it will neither cancel that request nor activate the other polarity request until the first request has been acknowledged. By following the rule that every change of state must be acknowledged, the event source provides a valid input to a QDI communication system. In this paper, we assume that an ideal delay-insensitive source is available and do not describe its inner workings.

Fig. 1(a) shows a block-level description of the proposed system. Each sensor cell is connected locally to an 'encoder' circuit. The encoder circuits are also connected one to the other in a chain.

The sensor cell outputs two data request wires, one for each polarity. Together with an acknowledge wire which outputs from the encoder block, these form a 'one-of-two' channel from the sensor to the encoder block. 'one-of-two' refers to the behavioural rules for the channel: the transmission of an event starts by switching exactly one of the two data-request wires. This is the first step in a 'four-phase handshake' (fig. 1 (b)). After making a request, the sensor must wait until the encoder block raises the acknowledge wire[3]. After this, the sensor must lower the request, and the encoder block must wait for the request to lower before lowering the acknowledge. Only after this may another request of either polarity be made.

The encoder block is subdivided into two blocks. Firstly, an 'increment' block, which receives a channel from upstream blocks and increments the addresses of address-events which pass through it; secondly a merge block, which passes through address-events from upstream blocks whilst inserting requests from the local sensor cell, assigning these the address '1', and arbitrating between these two sources where necessary. The channels which pass along the chain are one-of-four channels - as motivated in Sec. II-B.

'Exit' is marked after the final block in the chain: at this point the chain should be attached to a receiver, which also implements the four-phase handshake protocol and handles onward processing. The addresses assigned to the sensors are referred to this point in the chain. This is why the left-most sensor in the chain is referred to as 'Sensor 3': the sensor and encoder blocks are all identical and no addresses are preassigned, but rather, once sensor 3 introduces an event with the address 1, it must pass through two increment blocks before arriving at the exit, and so it ends up with '3' as its address.

B. *Address-event encoding*

Addresses are encoded as binary words, but each bit is transmitted in series, and the sequence has a variable length. The least-significant bit (LSB) is transmitted first, up to the most-significant bit (MSB). As the address-events have a one-bit data payload, the transmission of the polarity bit is used to indicate that the address sequence has finished. There are therefore four possible tokens, of which only one will be transmitted at a time: 'address 0', 'address 1', 'polarity a' and 'polarity b'; this is the reason for using a one-of-four protocol. As every binary number apart from 0 has 1 as its MSB, the address numbering can start from 1 sparing the transmission of the MSB. Table I demonstrates the encoding. It also shows a fixed-width binary encoding for comparison: No matter which width is chosen (16 bits in the example), unnecessary bits are transmitted in the case of lower addresses, whilst higher addresses eventually exceed the capacity of the address space.

Address in decimal	Address in 16-bit binary (MSB to LSB)	Our encoding
0	0000000000000000	not used
1	0000000000000001	P
2	0000000000000010	0 P
3	0000000000000011	1 P
4	0000000000000100	0 0 P
5	0000000000000101	1 0 P
6	0000000000000110	0 1 P
7	0000000000000111	1 1 P
8	0000000000001000	0 0 0 P
9	0000000000001001	1 0 0 P
99999	overflow (17 bits)	1 1 1 1 1 0 0 1 0 1 1 0 0 0 0 1 P

TABLE I
REPRESENTATION OF ADDRESSES IN OUR CHOSEN ENCODING. THE SEQUENCES TO THE RIGHT ARE SENT IN ORDER FROM LEFT (LSB) TO RIGHT (MSB). P IS A TOKEN WHICH INDICATES THE POLARITY OF THE ADDRESS-EVENT AND WHICH REPLACES THE MSB. DETAILED DESCRIPTION IN THE TEXT.

C. *Increment block*

Fig. 2 shows how a regular digital incrementer works. Fig. 2(a) gives a symbolic representation of a circuit whose role is to handle a single bit of an incrementation. It has two input ports, labelled 'bit in' and 'carry in'. Assuming these ports hold valid data, the look-up table (fig. 2(b)) shows what outputs it will produce at its two output ports 'bit out' and 'carry out'. Fig. 2(c) shows how three of these blocks can be combined to increment a 3-bit number. The example number '3' is presented to the 'bit in' ports of the three blocks, whilst a '1' is presented to the 'carry in' port of the block for the least-significant bit. The least significant bit is calculated first, and its carry-out port provides the value '1' to the carry-in port of the next bit, allowing that in turn to be calculated. Referring to the look-up table, it can be seen that the number '4' is produced as a 3-bit binary output. The 'carry out' bit of the block for the most-significant bit could be ignored, or else

[2] We avoid terms such as 'ON'/'OFF', '0'/'1', 'inc'/'dec', some of which appear in the literature of event-based sensing, partly to avoid possible confusion with other concepts in logic design and partly to introduce a conceptual separation between the events themselves and the physical triggers of these events, which may be different in different sensor types.

[3] In practise it is often more convenient to implement an acknowledgement as an active-low signal, which is then referred to as an 'enable' signal; however we talk about acknowledgement because it may be conceptually easier.

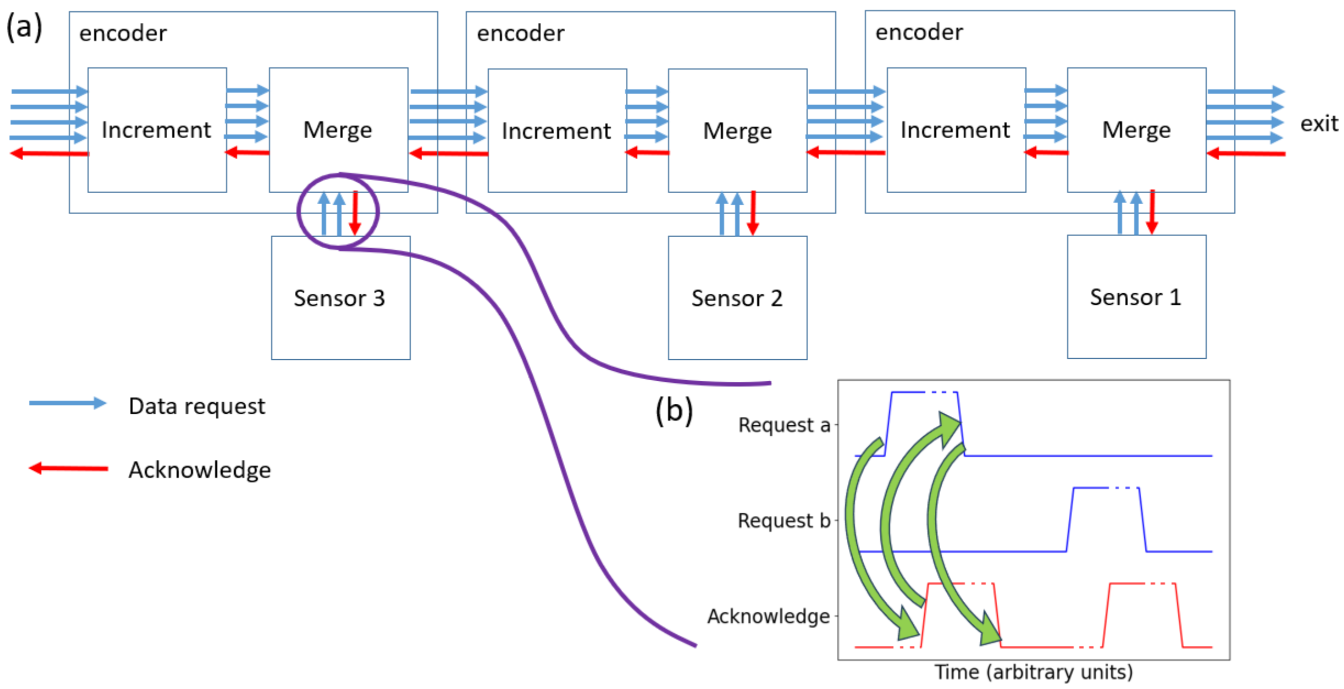
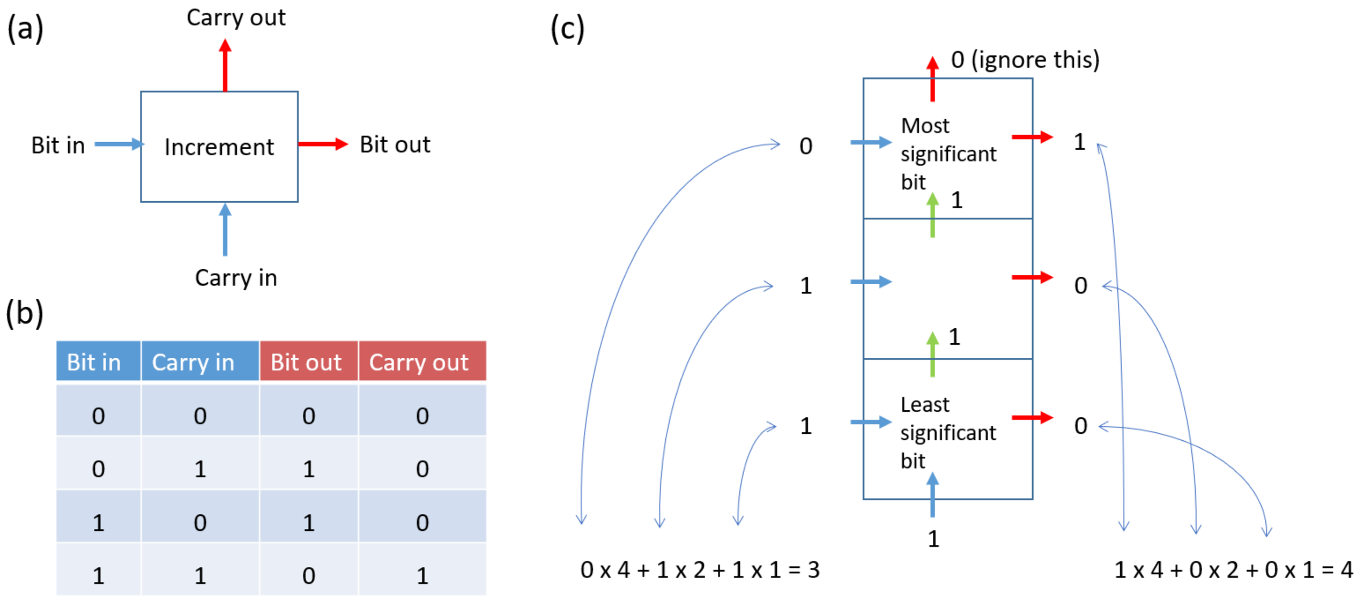


Fig. 1. (a) A one-dimensional array of sensor cells, each connected to its own encoder block, which are then chained together. Each arrow represents a wire; a bundle of data-request wires together with a single acknowledge wire create a channel. (b) Timing of the four-phase handshakes of a one-of-two channel between one of the sensor cells and its encoder module. The sensor cell raises request ‘a’, and this triggers the encoder to raise the ‘ack’ signal, where the causality is shown by the green arrow. The encoder responds once it is internally free to do so - dotted lines show the periods such as this where an arbitrary delay could be introduced. Once ‘ack’ is raised, the request can be lowered. Once the request has been lowered the acknowledge can be lowered. The channel is then free for new tokens to be transmitted - a second transmission is shown, this time from request ‘b’, but any sequence is possible, e.g. ‘a’, ‘a’, etc.



Bit in	Carry in	Bit out	Carry out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 2. (a) Symbolic representation of a bit incrementer, showing the two input ports in blue, and the two output ports in red; (b) look-up table showing the behaviour of the bit incrementer, i.e. which outputs it will produce for each combination of inputs; (c) A worked example for a 3-bit incrementer. Detailed description in the main text.

used to tell if the incrementation operation had overflowed by producing a 4-bit number.

In order to perform incrementation on an asynchronously transmitted variable-length code, a single increment block plays the role of the bit-incrementer for every bit in turn. To achieve this, one can think of creating a channel from the ’carry out’ port to the ’carry in’ port, whose token is to be consumed in the following cycle. Fig. 3(a) shows the channel connections for the increment block. In practise though, we do not have an explicit implementation of the carry channel, but rather maintain the carry state with two state variables, and check that these have made the required transitions before advancing to the completion of the handshaking cycle, i.e. four-phase handshakes with the input and output channels, whose sequencing is described below.

Since the number of tokens in the sequence may need to increase, for example in the transition between address 7, with its 3 tokens ('1', '1', 'P'), and address '8', with its 4 tokens ('0', 0', '0', 'P'), the block must be capable of inserting a new token into the sequence, which is achieved when necessary by performing the four-phase handshake to the right twice before completing a single handshake to the left. Fig. 3(b) shows how the basic incrementer look-up table is extended to achieve the desired functionality. In particular, when a polarity token arrives, this indicates the end of the address; if at this point the carry bit is 1, then a new token must be inserted. A '0' token is therefore output along both the address-event channel and the carry value is also set to '0'. Meanwhile, the handshake to the incoming address-event channel does not complete. The new conditions, i.e. a polarity token coupled with a '0' carry in, allow the polarity bit to be transmitted to the address-event output. Only then is the handshake to the address-event input channel completed. Meanwhile, the carry-out channel is set to '1', creating the initial conditions for a new address-event to be incremented.

D. Merge block

The channel connections for the block are shown in fig. 4. Its role is to pass tokens from the address-events-in channel to the address-events-out channel whilst merging in new address-events created by tokens which arrive from the sensor-in channel. Because events may arrive from the address-events-in or the sensor-in source, the block includes an arbiter, whose role is to decide which source came first in case of conflict, and to guarantee exclusive access to that source until communication is complete. A complexity is that an incoming address-event will be composed of multiple tokens in a sequence, and once a sequence starts, it must not be interrupted until it is completed by the arrival of a polarity token. This complexity is handled by a chain of internal states which can block the sensor-in channel's access to the arbiter during an address-event. When a sensor polarity token is accepted, it is transmitted out as a single polarity token, and this represents the address '1'.

E. Decoder function

The decoder block can be chained together in order to decode address-events and deliver them to a one-dimensional array of receiver elements. As the address-events have a one-bit polarity payload, the decoder uses a one-of-four channel as previously. The decoder block performs a decrement function and a split function. The decrement function reverses the logic of incrementation, meaning that it sometimes needs to remove tokens from a sequence rather than insert them, and therefore it may handshake twice to the input before completing a handshake with the output. The decoder block includes the ability to recognise the address '1' and to split off the polarity payload in order to deliver it to a local receiver. By contrast to the merge function, the split function is deterministic - there is no need for the equivalent of an arbiter. The overall logic is simpler to the point that the decrement and split functions are not chained together but rather implemented in a single stage. Fig. 5 shows the channel connections for the decoder block. The equivalent of 'carry' in the incrementer is called 'borrow' here, due to the need for less-significant bits which switch from '0' to '1' to borrow value from a more significant bit later in the sequence.

F. Design process

We briefly outline the design process that leads to the code. In the syntax of Communicating Hardware Processes [31], an input-driven asynchronous process may be simply described as:

```
*[L?x; R!f(x)]
```

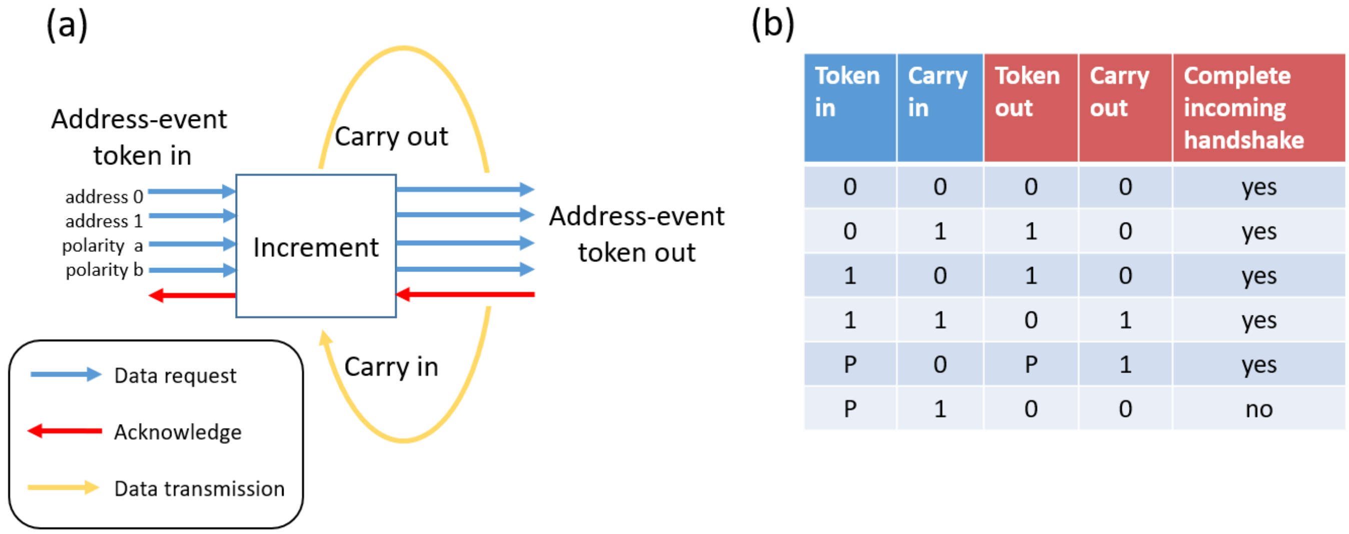
where the process waits to receive an input x from a input (or L=Left) channel, and transmits some function f of that input towards the output (o R=Right) channel, before repeating indefinitely.

When a block has to handshake both to the left and the right, the order in which these handshakes advance makes a difference to circuit complexity, and to pipelining, i.e. the ability of successive blocks to work on separate tokens at the same time. [32] provided an analysis of this, deriving a series of templates, amongst which, the 'pre-charge half-buffer' template, which the blocks presented here are based on. Half-buffers take their name from the fact that if they are in a chain, the chain can contain at most one token for every two half-buffers (another way of saying this is that each buffer contributes half a token of slack). The property of a half-buffer arises in a handshaking expansion (HSE) of the above CHP, so for a simple buffer (i.e. f = identity) on a one-of-two input, the HSE is:

```
*[  // repeat forever
   [(L0|L1)&˜RA];  // wait until input is valid and right is ready
   (L0→R0↑);  (L1→R1↑);  // evaluate and drive outputs
   LA↑;  // acknowledge left
   [RA];  // wait for right to acknowledge
   R0↓;  R1↓;  // pre−charge outputs
   [˜(L0|L1)];  // wait for left to return to quiescence
   LA↓;  // drop left ack
   [˜RA]  // wait for right to drop ack
]
```

We have modified the above template to implement the necessary logic for each block. We revert to CHP here to summarise the logic. For an incrementer block with one-of-three (i.e. ignoring event polarity) can be described in CHP thus:

```
*[
   c:=1;  // a variable representing carry−in is preset to 1
   do  // Consume bits until end token, emitting result bits on the way
      [] L0? → if c=0 → R0!; c:=0 [] c=1 → R1!; c:=0
       // input bit b = 0
      [] L1? → if c=0 → R1! ; c:=0 [] c=1 → R0!; c
      :=1  // input bit b = 1
      [] L2? → break  // end of input number
```



Token in	Carry in	Token out	Carry out	Complete incoming handshake
0	0	0	0	yes
0	1	1	0	yes
1	0	1	0	yes
1	1	0	1	yes
P	0	P	1	yes
P	1	0	0	no

Fig. 3. (a) Channel connections for incrementer block, showing data-request wires in blue and acknowledge wires in red. One-of-four channels bring through the successive tokens of a serially encoded address-event. The diagram conceptualises the carry as a channel which transmits data from the 'carry out' port to the 'carry in' port; our implementation achieves the passage of carry data to the next cycle without explicit channel machinery, but rather by manipulating internal states. (b) Look-up table showing the behaviour of the incrementer, 'P' indicates either of the polarity tokens, which is then mirrored to the output. Detailed description in the main text.

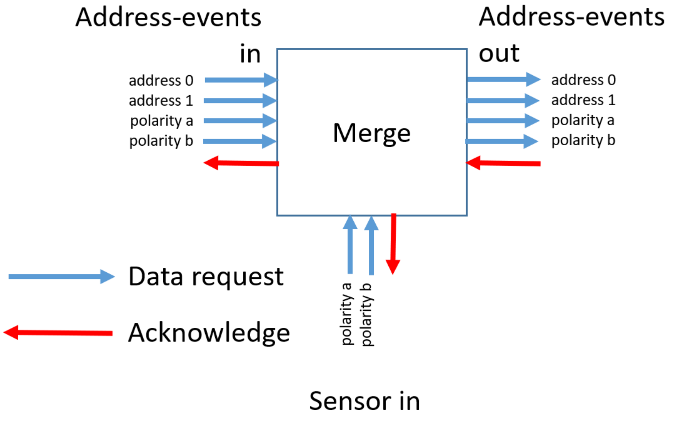


Fig. 4. Channel connections for merge block, showing data-request wires in blue and acknowledge wires in red. One-of-four channels bring through the successive tokens of a serially encoded address-event, whilst tokens from the local sensor modules result in new address-events being inserted and passed out. Detailed description in the main text.

```
  if c=1 → R1! [] c=0 → skip // If carry still 1, emit one
     more '1' bit
  R2! // End token for the output number
]
```

Meanwhile, CHP for the merge logic between the chain (L) and a local event source (D=Down), again ignoring polarity, is:

```
*[
  [] D?  → R2! // D causes end token (representing the MSB for the
     number 1)
  [] L0? → R0! ; ForwardRestFromL // start forwarding an L−
     sequence
  [] L1? → R1! ; ForwardRestFromL
  [] L2? → R2! // empty sequence shortcut
]
subprocess ForwardRestFromL:
```

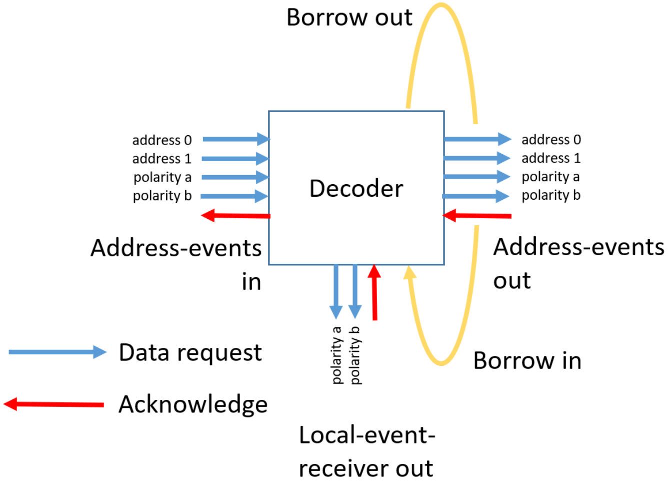


Fig. 5. Channel connections for decoder block, showing data-request wires in blue and acknowledge wires in red. One-of-four channels bring through the successive tokens of a serially encoded address-event. An incoming address of 1 is split off and delivered to the local event receiver. A 'borrow' value, necessary for the decrementation of the serially encoded address is envisaged here as a channel of communication from the processing of the previous to the next token in an address-event, and in practice, forms part of the internal state which is necessary for the full implementation of block functionality.

```
  do
    [] L0? → R0!
    [] L1? → R1!
    [] L2? → R2! ; break
```

The above logic is then expanded for example by handling different polarities etc. Synthesis then proceeds by implementing a Production Rule Set (PRS) [14]. This describes a series of nodes, each with a defined pull-up and pull-down network, where these networks are logical combinations of the state of some other nodes in the circuit. Channels can also be defined, consisting of sets of nodes playing defined roles such as data and acknowledge signals, and over which certain rules of communication, such as mutual exclusion, can be assumed. Synthesis involves adding a series of guards which ensure that signal transitions occur only in orders allowed by the template. In this process, additional signals may be added if necessary either to ensure that a CMOS implementation is possible (so called 'bubble reshuffling') or to simplify the design.

G. Circuit design and simulation

To simulate the circuitry, we have used the open-source Asynchronous Circuit/Compiler Tools (ACT) software developed by Manohar and colleagues at Yale [33]. This software allows pull-up and pull-down networks, once logically enabled, to take highly random amounts of time to force a transition of the state of the node to which they are attached. Repeated simulation allows cases of event timing to be tested which are much more extreme than would occur in practice in a CMOS implementation, so that any conditions which would cause the circuit to fail in practice, such as interference between a pull-up and a pull-down network simultaneously active on the same node, can be caught quickly. The delays are drawn from a $1/(1) = x$ distribution with $x = [0, 65536)$ for an extra wide std-deviation of the samples $\sigma \approx \frac{65536}{\sqrt{2}}$ compared to the mean $\overline{x} = \frac{65536}{\ln{(65536+1)}}$. This type of simulation gives a good level of assurance that circuits actually conform to QDI assumptions. For example, the simulation reported in section III-A2 below involved around 235000 gate switching events and therefore 235000 random delays used.

Once a production rule set is defined, there is a simple deterministic process to convert this into a digital CMOS design. The X-Fab 0.18 μm CMOS design kit has been used within Cadence to manually construct these corresponding schematics [4].

The encoder has been laid out (see figure 6) in a $60 \times 90 \mu m$ block, transistor sizings based on standard rules of thumb for complementary logic, including keepers with 1:4 drive strength ratio, and manual layout not optimised for area [5].

III. Results

The core production rule sets for the 3 blocks (*increment*, *merge* and *decrement*) can be found in the supplementary materials. The complete design within the ACT framework can be found in this GitHub repository. The repo includes test benches, input data, output data and data analysis and visualisation scripts which replicate the results shown in the following section.

For each signal in the design (for example r1) there are a pull-down (→ *r1-*) and a pull-up rule (→ *r1+*), where the pull-down rule is subsequently constructed as a network of n-type transistors and the pull-up rule with p-type transistors. The right half of each pull-down rule contains the logical complement of the left half of the pull-up rule, and vice versa. This design pattern completes the rule set so that no weak feedback is necessary in order to stabilise the r1 node. This pattern is more verbose than one relying on weak feedback, leading to a larger design with almost twice as many transistors. This design pattern has been followed as a step towards subsequent conversion to a single-polarity design for candidate flexible electronic technologies (not presented in this paper). The resulting design contains 151 transistors in the incrementer, 299 transistors in the merge (therefore 450 transistors in the encoder) and 569 transistors in the decoder.

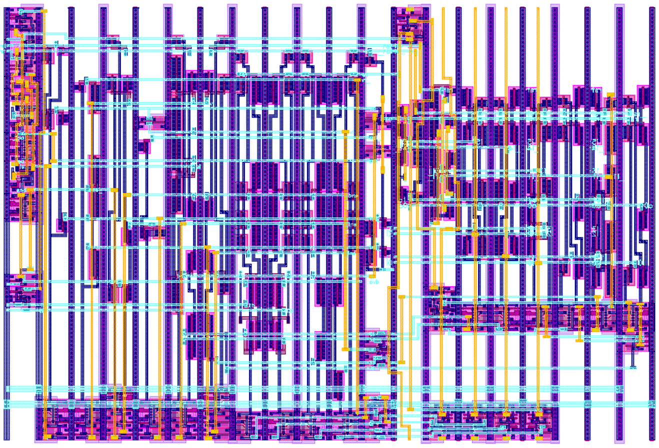


Fig. 6. Manual layout for encoder in X-Fab 0.18 μm.

The repository also contains a non-complementary version of the encoder, which relies on keepers, and this design has been used in the ams/spectre simulations below. This design contains 368 transistors c.f. 450 for the complementary design.

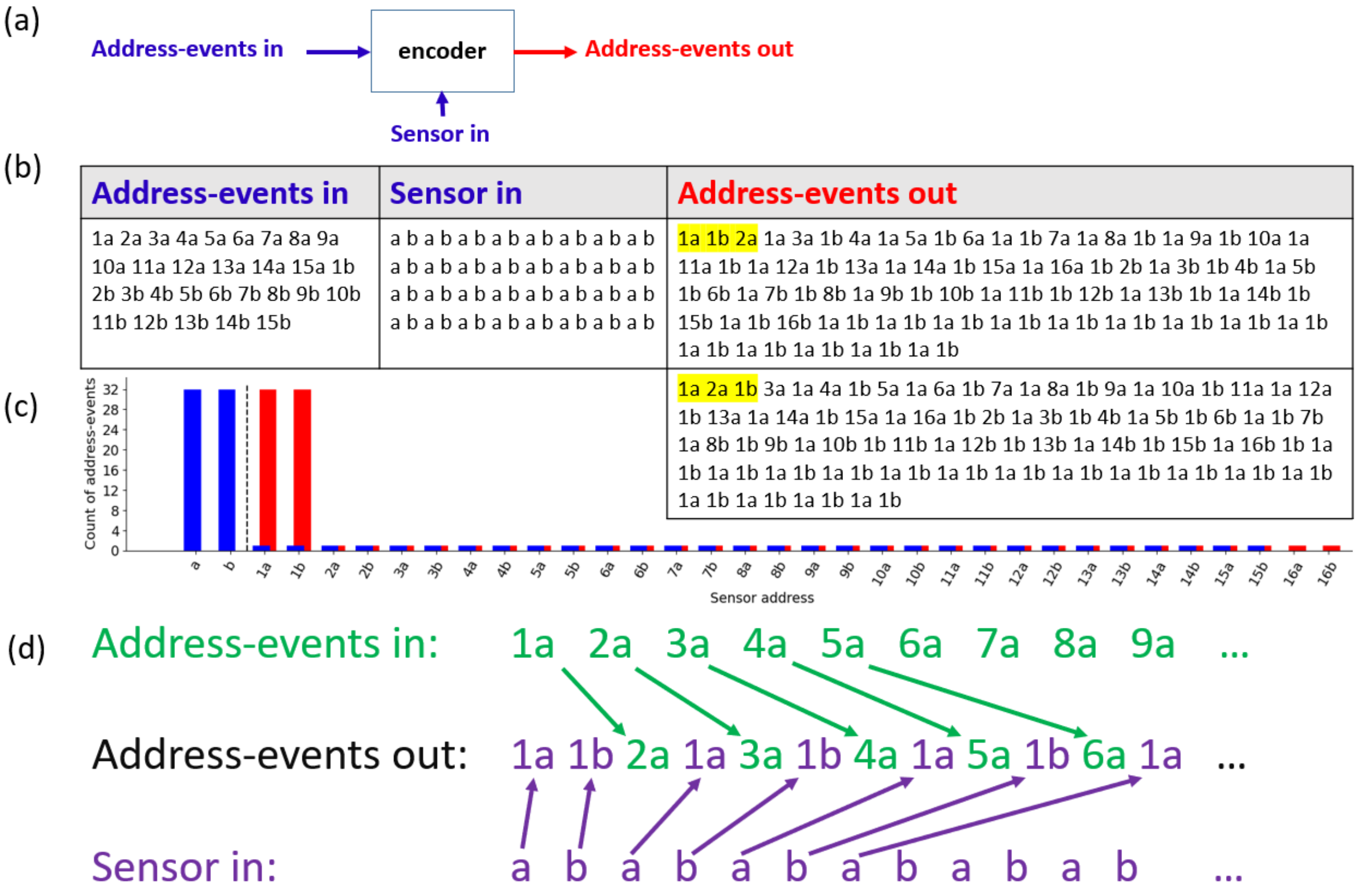
A. ACT simulation results

1) Single encoder: We present results from three test scenarios. Fig. 7(a) shows the first scenario, in which address-events from previous sensing element along the chain and local sensor events come into the encoder block from their respective channels. Fig. 7(b) shows the actual sequences of events which were input and received at the output [6]. (c) is a histogram which gives counts of the events both in and out, making it easy to see that every event input has been incremented by 1 and output. Two different output sequences are given for the same set of inputs, which result from using a different random seed in the simulator. It can be seen that all the same tokens are output (thus, the histogram in (c) is identical) but that they come out in different orders; for example the first sequence starts '1a, 1b, 2a ...' but the second sequence starts '1a, 2a, 1b ...' (highlighted in yellow in the raw data). Diverse merging order for identical sender behaviour is always possible but is made more likely by the extreme random timing of the simulator; a given physical

[4] The prs2net tool in the same repo could be used to automate the production of spice netlists for simulation, but this was not used here; some minor manual optimisations were made between PRS and schematic.

[5] At the time of writing we know that minor changes of transistor sizing are necessary w.r.t. this layout to avoid glitches due to charging pumping. This layout includes muxes which allow the cell to be disabled, passing data directly through in the L-R chain - 60 extra transistors

[6] The addresses have been converted from the serial encoding; the actual token sequences are at https://github.com/event-driven-robotics/snowball/tree/main/outputs, the conversions are obtained with the script https://github.com/event-driven-robotics/snowball/blob/main/data_conversion.py



Address-events in	Sensor in	Address-events out
1a 2a 3a 4a 5a 6a 7a 8a 9a 10a 11a 12a 13a 14a 15a 1b 2b 3b 4b 5b 6b 7b 8b 9b 10b 11b 12b 13b 14b 15b	a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b a b	1a 1b 2a 1a 3a 1b 4a 1a 5a 1b 6a 1a 1b 7a 1a 8a 1b 1a 9a 1b 10a 1a 11a 1b 1a 12a 1b 13a 1a 14a 1b 15a 1a 16a 1b 2b 1a 3b 1b 4b 1a 5b 1b 6b 1a 7b 1b 8b 1a 9b 1b 10b 1a 11b 1b 12b 1a 13b 1b 1a 14b 1b 15b 1a 1b 16b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b
		1a 2a 1b 3a 1a 4a 1b 5a 1a 6a 1b 7a 1a 8a 1b 9a 1a 10a 1b 11a 1a 12a 1b 13a 1a 14a 1b 15a 1a 16a 1b 2b 1a 3b 1b 4b 1a 5b 1b 6b 1a 1b 7b 1a 8b 1b 9b 1a 10b 1b 11b 1a 12b 1b 13b 1a 14b 1b 15b 1a 16b 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b 1a 1b

Fig. 7. (a) Test scenario, with inputs in blue and outputs in red. (b) Raw event sequences for each input and output. The suffixes 'a' and 'b' are shorthand for the two event polarities, and 'a' and 'b' without a numeric address prefix are used to indicate the events from the sensor. Two output sequences are shown one above the other for the same inputs, each resulting from a different random seed for the prsim simulator. (c) Histogram of events in (blue) and events out (red) by sensor address. The events to the left of the dashed line are those from the local sensor. (d) Detail of how the two event streams have been merged, for the start of the first output sequence from (b).

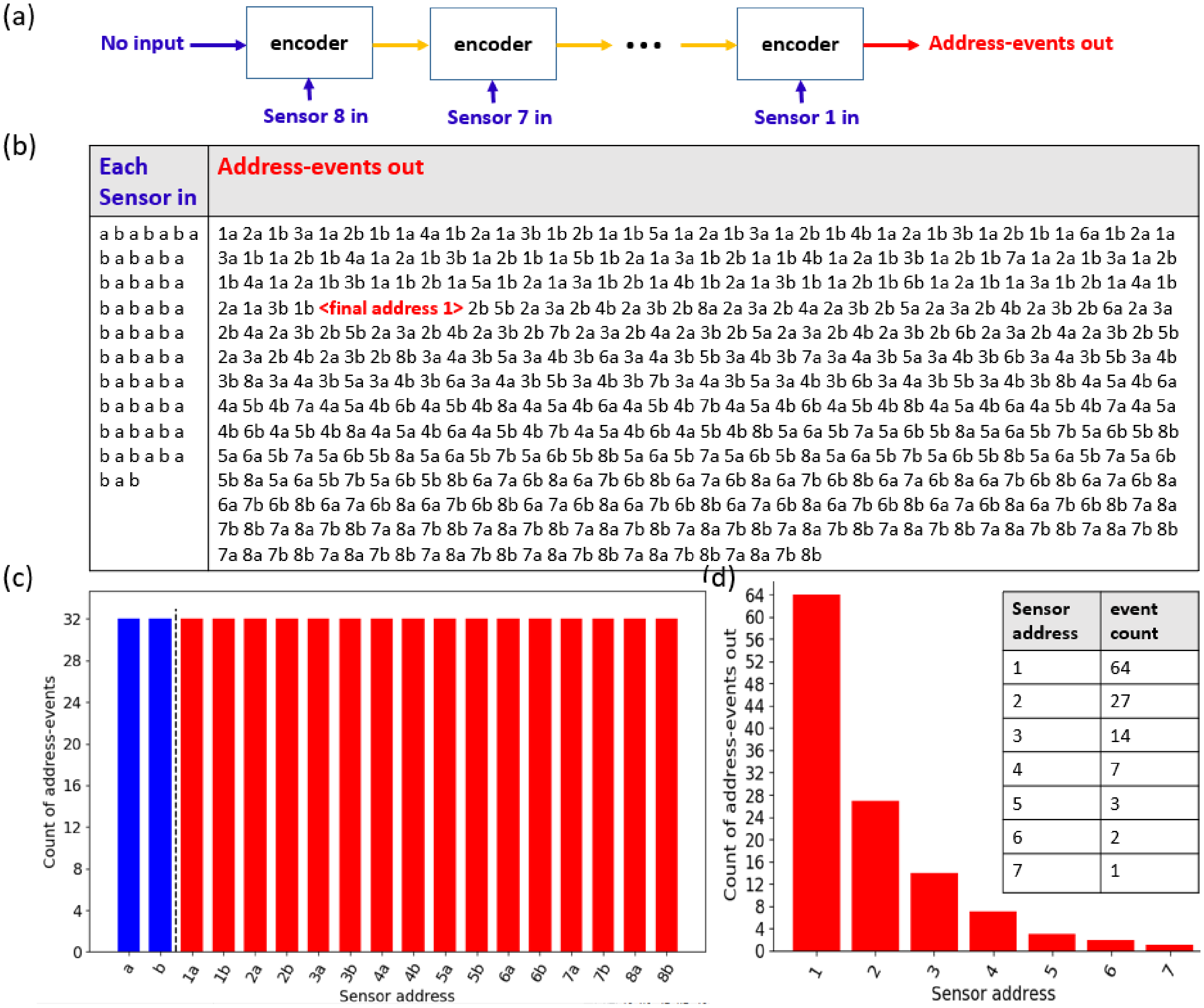
implementation of this system is likely to handle arbitration in a highly repeatable way. (d) Illustrates how the first section of the first output sequence arises from the merging and incrementing of the two address sources.

2) Encoder array: Fig. 8(a) gives the scenario for which the application is envisioned, in which all events arise from a 1D array of sensors, and accumulate at the output, having had individual addresses assigned through progressive incrementation. Fig. 8(b) gives the actual event sequence which was input at each sensor, and the sequence of address-events which came out. These are summarised in the histogram in fig. 8(c), which shows that the sensor sequences were all assigned incrementing addresses. We will discuss the ordering of the output sequence in section IV-B below.

3) Decoder: Fig. 9(a) shows how we have tested the decoder block through the streaming in of address-events from a single source. Fig. 9(c) summaries the counts of events at the input and received at each of the outputs, making it clear that each address has been decremented, and that events which entered with address 1 were sent to the local event receiver.

B. Spectre (Schematic) simulation results

A PRS design which is CMOS implementable can be translated straightforwardly into a working CMOS design. In 10 we present the results of a brief simulation of such a translation, for which we have used the X-Fab 0.18μm process. The simulated system is a single encoder block receiving simulated inputs from a channel l and a local event source d, and producing outputs on channel r. The r enable signal is produced by a nor gate on the outputs of channel r. We have handcrafted inputs (red traces) in order to highlight aspects of the performance of the system. The outputs of the encoder are shown in green. The 'r' channel enable (*re*) is shown in blue as this is produced by a 4-input nor gate on the r channel outputs (this enable signal is active-low, as opposed to the active-high acknowledge signal shown in fig. 1. Each address-event is shown by a linked pair of black balloons and these are numbered. For example (1) a single token on l2 (i.e. address: 1a) produces two tokens in output - r0 followed by r2 (i.e. address: 2a). (3) a token t0 (i.e. polarity a) produces token r2 (i.e. address 1a), but although it presents itself at 500 ns, it must wait until multi-token address-event (2) has completed before it gets inserted into the r channel.



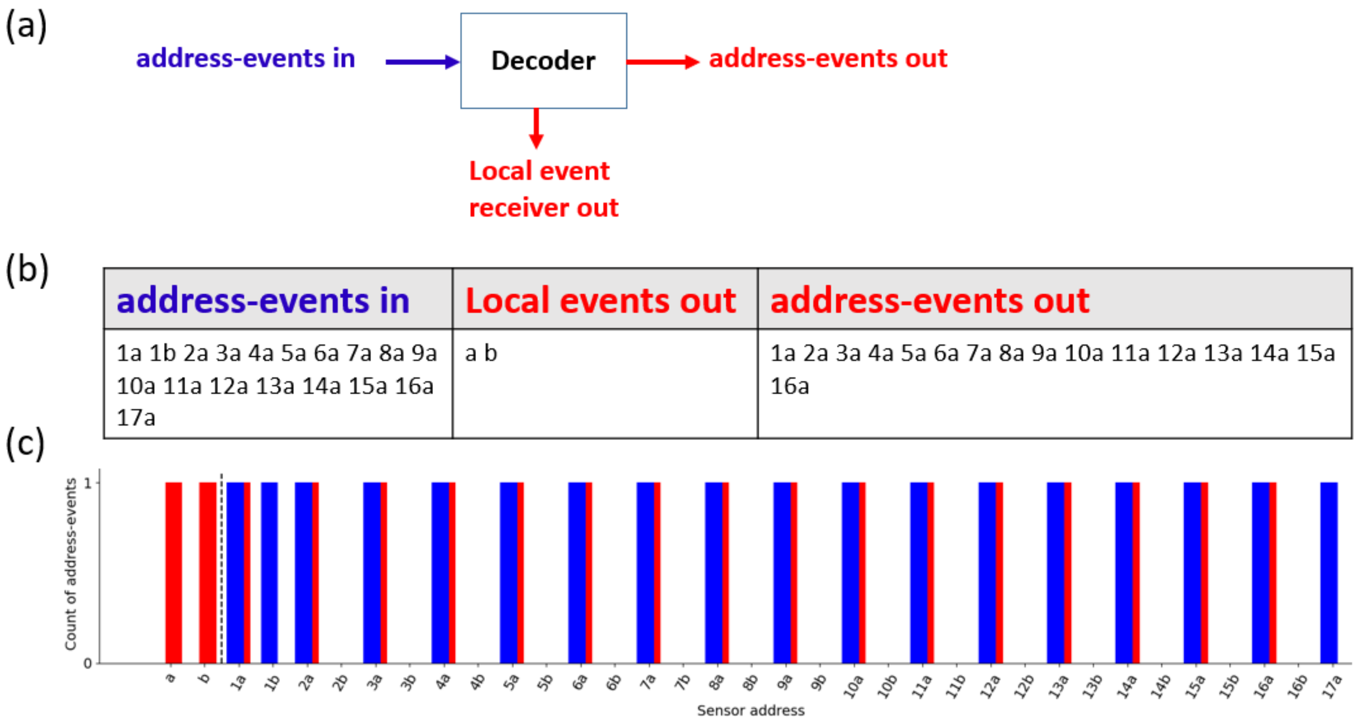
Each Sensor in	Address-events out
a b a b a b a	1a 2a 1b 3a 1a 2b 1b 1a 4a 1b 2a 1a 3b 1b 2b 1a 1b 5a 1a 2a 1b 3a 1a 2b 1b 4b 1a 2a 1b 3b 1a 2b 1b 1a 6a 1b 2a 1a
b a b a b a	3a 1b 1a 2b 1b 4a 1a 2a 1b 3b 1a 2b 1b 1a 5b 1b 2a 1a 3a 1b 2b 1a 1b 4b 1a 2a 1b 3b 1a 2b 1b 7a 1a 2a 1b 3a 1a 2b
b a b a b a	1b 4a 1a 2a 1b 3b 1a 1b 2b 1a 5a 1b 2a 1a 3a 1b 2b 1a 4b 1b 2a 1a 3b 1b 1a 2b 1b 6b 1a 2a 1b 1a 3a 1b 2b 1a 4a 1b
b a b a b a	2a 1a 3b 1b <final address 1> 2b 5b 2a 3a 2b 4b 2a 3b 2b 8a 2a 3a 2b 4a 2a 3b 2b 5a 2a 3a 2b 4b 2a 3b 2b 6a 2a 3a
b a b a b a	2b 4a 2a 3b 2b 5b 2a 3a 2b 4b 2a 3b 2b 7b 2a 3a 2b 4a 2a 3b 2b 5a 2a 3a 2b 4b 2a 3b 2b 6b 2a 3a 2b 4a 2a 3b 2b 5b
b a b a b a	2a 3a 2b 4b 2a 3b 2b 8b 3a 4a 3b 5a 3a 4b 3b 6a 3a 4a 3b 5b 3a 4b 3b 7a 3a 4a 3b 5a 3a 4b 3b 6b 3a 4a 3b 5b 3a 4b
b a b a b a	3b 8a 3a 4a 3b 5a 3a 4b 3b 6a 3a 4a 3b 5b 3a 4b 3b 7b 3a 4a 3b 5a 3a 4b 3b 6b 3a 4a 3b 5b 3a 4b 3b 8b 4a 5a 4b 6a
b a b a b a	4a 5b 4b 7a 4a 5a 4b 6b 4a 5b 4b 8a 4a 5a 4b 6a 4a 5b 4b 7b 4a 5a 4b 6b 4a 5b 4b 8b 4a 5a 4b 6a 4a 5b 4b 7a 4a 5a
b a b a b a	4b 6b 4a 5b 4b 8a 4a 5a 4b 6a 4a 5b 4b 7b 4a 5a 4b 6b 4a 5b 4b 8b 5a 6a 5b 7a 5a 6b 5b 8a 5a 6a 5b 7b 5a 6b 5b 8b
b a b a b a	5a 6a 5b 7a 5a 6b 5b 8a 5a 6a 5b 7b 5a 6b 5b 8b 5a 6a 5b 7a 5a 6b 5b 8a 5a 6a 5b 7b 5a 6b 5b 8b 5a 6a 5b 7a 5a 6b
b a b	5b 8a 5a 6a 5b 7b 5a 6b 5b 8b 6a 7a 6b 8a 6a 7b 6b 8b 6a 7a 6b 8a 6a 7b 6b 8b 6a 7a 6b 8a 6a 7b 6b 8b 6a 7a 6b 8a
	6a 7b 6b 8b 6a 7a 6b 8a 6a 7b 6b 8b 6a 7a 6b 8a 6a 7b 6b 8b 6a 7a 6b 8a 6a 7b 6b 8b 6a 7a 6b 8a 6a 7b 6b 8b 7a 8a
	7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b
	7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b 7a 8a 7b 8b

Sensor address	event count
1	64
2	27
3	14
4	7
5	3
6	2
7	1

Fig. 8. (a) Test scenario, with inputs in blue, outputs in red, and intermediate channels in yellow. The leftmost input is unused. (b) Event sequences for each input and output. The suffixes 'a' and 'b' are shorthand for the two event polarities, and 'a' and 'b' without a numeric address prefix are used to indicate the events from a sensor. In the output sequence, we have marked the point at which the final address-event from sensor 1 was received. (c) Histogram of events in (blue) and events out (red) by sensor address. The events to the left of the dashed line are those from each local sensor; note that there are 8 identical sets of these, one from each sensor. (d) Histogram of events out only up to the point that the final address-event from sensor 1 was received (the rest of the events go on to arrive after this snapshot, until 64 events per sensor have been received). These events are not separated by polarity, so for example addresses '1a' and '1b' are both counted as address 1. The inset table shows the actual count of address-events received.

C. Benchmarking

We compare the encoder to a parallel AER (P-AER) encoder, in terms of speed, area, power and number of pins. Specifically we compare to a one-dimensional encoder: although some neuromorphic chips take advantage of a two-dimensional array of spike sources to simplify encoding hardware, it is also common for neurons to be arranged in one-dimension and for the second dimension of the chip area to host a synaptic cross-bar array, as in [34]. We explore a wide range of numbers of inputs (5 points spaced exponentially over 3 orders of magnitude) to investigate how various quantities scale. Inputs are perfect, polarised (or paired) event sources, so one input maps to either one encoder from this work with a local one-of-two input or to two adjacent inputs to P-AER, and the experiment measures the effects of one spike from each of these pairs, all delivered simultaneously plus a small amount of temporal jitter (uniform in range $5\,ns * num\_input\_pairs$ from a common seed). The P-AER encoder is constructed from the templates of https://github.com/async-ic/actlib-neurosynaptic-perifery, (specifically the "sadc_encoder" unit test, which has been deployed in the spiking ADC of the TEXEL chip [35]. It arbitrates, encodes and then converts from QDI to bundled data output; this is a typical approach to reducing the number of pins necessary for inter-chip communication, at the expense of additional stages and the introduction of timing assumptions. We do not perform



address-events in	Local events out	address-events out
1a 1b 2a 3a 4a 5a 6a 7a 8a 9a 10a 11a 12a 13a 14a 15a 16a 17a	a b	1a 2a 3a 4a 5a 6a 7a 8a 9a 10a 11a 12a 13a 14a 15a 16a

Fig. 9. (a) Test scenario, with inputs in blue and outputs in red. (b) Event sequences for each input and output. The suffixes ’a’ and ’b’ are shorthand for the two event polarities, and ’a’ and ’b’ without a numeric address prefix are used to indicate the events sent to the local event receiver. (c) Histogram of events in (blue) and events out (red) by sensor address. The events to the left of the dashed line are those received by the local event receiver.

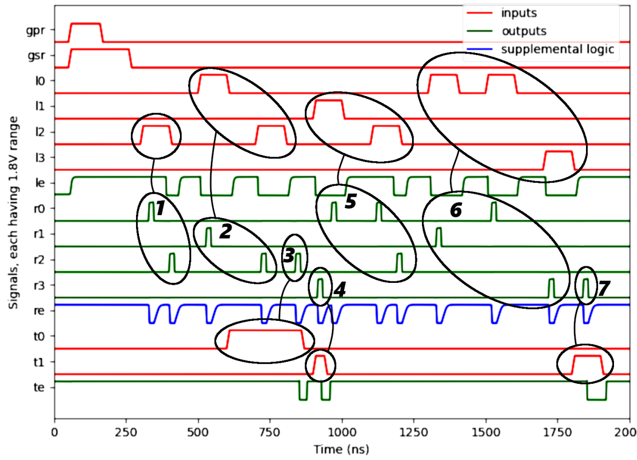


Fig. 10. Results of a transistor-level Spectre simulation of an encoder merging inputs from both upstream addresses and from a local event producer. Detailed description in the text.

bundled data conversion for this work because the number of wires is neither large nor expanding. The design incorporates programmable delay stages to ensure data validity; we have nominally included one such stage but have not used this delay for these comparisons. Knowing that each encoder block introduces 1 token of slack, we have calculated the mean number of serial address-events from a spiking source that can queue ahead of it before exiting the chain (each requiring a variable and growing number of tokens)[7], and we have introduced sufficient half-buffer stages in the P-AER encoder (between the arbiter and the final bundled data converter) to introduce the same amount of slack; this design choice might be reasonable for P-AER when for example significant bursts of spikes from multiple sources are expected or when the receiver is expected to consume events at a non-constant rate; however this choice comes at the expense of an increased single-event latency. We have also compared against P-AER with only a nominal addition of 2 half-buffers (slack=1) to decouple the encoder from the converter to bundled data. Both this p-AER implementation and our work are simulated prior to layout in the X-Fab 0.18μm process. The results are in table II. The number of transistors in each design can serve as a proxy for both quiescent power and layout area. Energy per address-event is given; additional power necessary to transmit off-chip is not considered, since this cost is highly dependent on the nature of downstream circuitry. We do not compare to an LVDS-based design such as [36] for the same reason. Mean latency is given, which should be understood as queuing time given 50% load (one input per input pair) almost simultaneously. Figure 11 summarises the results, showing best-fit lines on a log–log plot. The reported gradients correspond to power-law exponents, assuming that scaling is in the relation $y \propto inputs^{grad}$.

[7] https://github.com/event-driven-robotics/snowball/blob/main/calculate_slack_by_address.py

Num inputs	Num pins output		Mean queue capacity (address-events)			Num transistors			Mean latency (ns)			Mean energy (J/event)		
Inputs (pairs)	This work	P-AER	This work	P-AER same cap.	P-AER slack=1	This work	P-AER same cap.	P-AER slack=1	This work	P-AER same cap.	P-AER slack=1	This work	P-AER same cap.	P-AER slack=1
10 (5)	5	6	0.6	0.5	1	1840	1580	1780	29.4	12.1	12.2	9.54	8.38	10.4
32 (16)	5	7	2.2	2	1	5890	3890	3420	143	41.5	41.2	43.4	17.4	12.8
100 (50)	5	9	5.6	5.5	1	18400	14000	11200	614	131	130	187	47.8	18.9
316 (158)	5	11	14	14	1	58100	55800	45170	2620	611	607	777	131	23.5
1000 (500)	5	12	35	35	1	184000	144000	113000	10700	2260	2240	3080	343	27

TABLE II
COMPARISON BETWEEN P-AER AND THIS WORK. ALL VALUES ROUNDED TO 3 SIGNIFICANT FIGURES. SIMULATIONS ARE PRE-LAYOUT, USING THE X-FAB 0.18μM PROCESS.

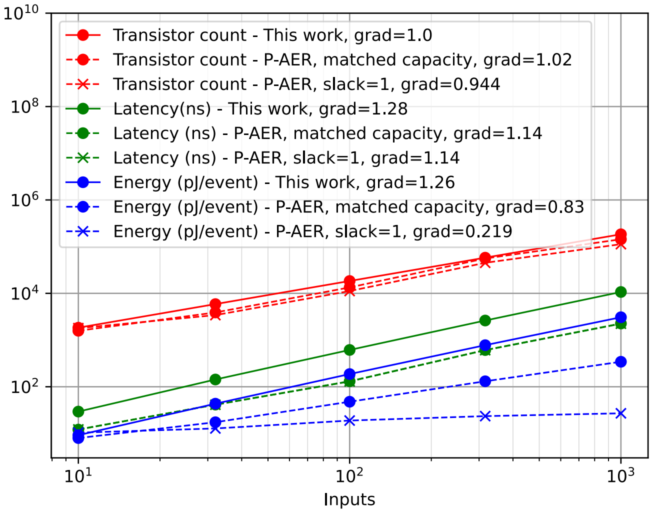


Fig. 11. Summary of results from table II

IV. DISCUSSION

A. Comparison to parallel AER

Compared to the number of inputs, the transistor count scales linearly, and is broadly similar to P-AER. Latency scales super-linearly in all designs and more steeply in this work. Power scales super-linearly for this work but sub-linearly for P-AER, particularly in the design with only nominal slack. Overall, this work does not excel in any of the quantitative metrics investigated; the advantages are instead in the freedom and simplicity of system design that it allows, especially for distributed systems, where e.g. one can play with trade-offs with respect to the number of elements in a sensor cell (expanding the payload) and the number of sensor cells. Moreover, we have presented mean performance by input, however this belies great differences in behaviour as discussed in the following section.

B. Sensor cell priority

In the encoder circuit, an arbiter serves to merge events which come from a local sensor circuit together with events which come from an upstream encoder circuit and thereby from other sensor circuits. Whilst this arbiter is not guaranteed to be fair, it performs fairly in practice, because while it is servicing a request from one of the two input paths, if a request arises from the other input path, then the other input path is very likely to be serviced next[8]. Nonetheless, the overall priorities that sensor cells are given within a chain of encoders is highly unfair. In case of sensory event production which exceeds the overall bandwidth of the channel, queuing will occur. In this queue, an event from a sensor cell attached to the final encoder in the chain needs to wait for just one event ahead of it before being processed by the encoder. For the sensor cell attached to the last-but-one encoder, it must wait for one event before being processed by the encoder, but then it must wait for the sensor cell ahead of it to have a turn before moving on to the final encoder. For the next one back, the two sensor cells ahead of it will be processed a total of three times before its turn, and so on, with relative priority approximately halving at each step backwards in the chain.

We demonstrate this here with a reanalysis of the results from the multiple sender experiment of section III-A2. Fig. 8(d) shows a histogram of the number of events which were received from each address (ignoring polarity) up to the point at which the final event from sensor 1 (i.e. the sensor closest to the exit) is received. The approximate exponential decay in the number of events received from sensors backward along the chain reflects the reducing priority each sensor has been given. (Note that this snapshot was taken before all events arrived - eventually an equal number of events from all sources went on to be delivered, since the simulated event sources themselves queued their events for as long as was necessary.)

Locally fair arbitration has been used to achieve globally fair arbitration across two-dimensional arrays since [7], giving all

[8] The deviations from this fair behaviour which can be seen in the ACT experiments presented are likely exaggerated due to the highly random simulation timing strategy.

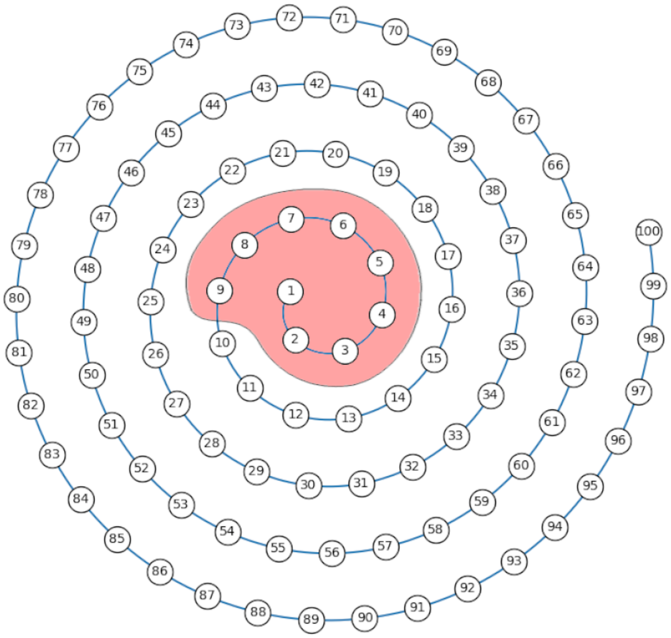


Fig. 12. The circles represent sensor cells arranged on a two-dimensional surface, perhaps taxels on a fingertip, with the number inside each one representing the address assigned, given their arrangement in a one-dimensional array (the blue line) with data flowing towards the centre. The pink region encloses the taxels which would be able to deliver events at 1 kHz simultaneously in the case that all of them were trying to do so, and assuming a 10 kHz total bandwidth for the array.

event-producing cells in the array equal priority. This seems like the most general strategy for arbitration. In the context of event-based vision, for example, this should mean that no part of the visual field is neglected for too long even in the case of high activity in another part of the visual field. We argue, however, that this strategy may not always be the best one. Fig. 12 represents taxels on the surface of a fingertip connected together in a spiral pattern. The connections represent links between encoder circuits, and the flow of data is inwards towards the centre of the spiral. The numbers of the taxels therefore represent the addresses that the taxels will be assigned by the encoder chain. A tactile array with performance at least equivalent to a human fingertip should have (rounding up to the nearest order of magnitude w.r.t. references) 1000 taxels [37], and a fast subset of these should achieve individual event frequencies of 1 KHz [38]; to achieve this the event bandwidth at the output of at least the final encoder circuit may need to be up to 1 MHz in order not to exceed bus capacity, an event rate which may not be achievable with new candidate transistor technologies for flexible electronics, such as IGZOs, OFETs, OECTs etc.

The figure shows a simplified scenario in which there are 100 taxels (which may be realistic given the number of taxels that are currently integrated on robotic platforms). Let's consider the effect of a system-wide event bandwidth of 10 kHz (i.e. the maximum address-event rate achievable at the output of cell 1, ignoring the fact that addresses with different numbers of tokens will have different delivery duration). Let's consider the effect of a vibrating stimulus which evokes an event rate of 1 kHz from every taxel for a brief period. If all of the taxels were serviced fairly, then each would produce events at only 100 Hz (i.e. 10 kHz total capacity divided equally between 100 cells), from which the nature and particularly the frequency of the stimulus could not easily be inferred. Instead what happens is that the centremost 9 taxels are serviced with high priority, each producing a 1 kHz signal. Thereafter backwards in the chain, taxels produce increasingly infrequent signals, with most being effectively ignored completely. For example, taxel 10 will produce at approx 500 Hz, taxel 11 at 250 Hz, and so on; if the situation continued indefinitely, taxel 35 would get serviced only once per year!

There is therefore a central region of the fingertip (as shown on the diagram) which has guaranteed high temporal sensitivity. Other regions of the fingertip also offer high temporal sensitivity in isolation, providing only that the bus is not overloaded. A prosthesis user or robot with such a fingertip could use this knowledge to adjust their active haptic exploration or handling strategy in order to prioritise the use of the highly sensitive region.

Similar arguments might be made for other sensory modalities, including vision; wherever a sensor is intended to be used for active exploration rather than in a passive reactive way, this prioritisation strategy might be preferable. Foveated vision sensors mimic the human eye in prioritising spatial acuity in their central region [39]; this protocol could deliver an alternative form of foveation offering better temporal acuity.

The passage of events through each link in the chain of encoders adds latency, which in the above example of frequency sensing will translate to a phase shift per taxel. Such phase shift would be somewhat predictable and might be calibrated against, were it to cause a problem for a particular application. In more detail, the sender-specific delay consists of a fixed time per transmission stage through which it must pass, plus a jitter, which is different for each transmission. The more stages an event must pass through the more timing uncertainty is added by the accumulation of jitters.

The multiple buffer stages and incrementation calculations also imply an energy cost. Mitigating this, all communications are local and there is never a need for a signal from one cell to drive the capacitance of multiple other driver stages, as is the case in parallel multiplexing. Detailed energy and timing analyses should be delayed until a target technology has been chosen.

In the above example of a high-frequency stimulus, after an abrupt end to this stimulus, the chain will flush out an old event from every cell within a 10 ms period; a receiver would need to interpret such a spatiotemporal pattern of events accordingly, perhaps by discarding those events.

C. Extensions and generalisations

If a polarity payload for each event is not required, for example if events are generated by neurons, the polarity token could be replaced straightforwardly with a token indicating the end of an address-event, and this would result in a one-of-three protocol. Such solutions have been designed in PRS and tested in ACT; however we do not present the results here, since they are straightforward simplifications of the work we have presented.

It should be possible to add more dimensions to the address space by inserting tokens in the sequence which represent a change of address dimension; however we haven't developed these ideas.

An implication of the decision to implement the decoder from a single half-buffer template is that, for the design presented, the decoder offers half a unit of slack per neural destination whereas the encoder offers one unit of slack per spiking source. In other words, an encoder chain would be able to store up twice as many events internally as a decode chain of the same length. Naturally, such decisions could be revisited in case of clearer system-level requirements.

The prioritisation strategy described in the previous section has been explored as a consequence of the possibility to print circuitry using a rolling process [40]. If unfair arbitration is undesirable, it could be partially mitigated by printing or otherwise producing repeating cells containing multiple sensory units, with fair arbitration performed between all sensors in a single cell, before address-events enter a serial encoder. For example, if a rolling printing process allowed 128 sensory cells to be printed in a single block, then the first 128 sensors in an array would share equal priority with the second group of 128 sensors and so on. Such a sharing of the cells developed here might reduce their effective transistor count per neural unit. We have not created such a design, which would require additional complexity for the transmission of local address bits. Fig. 13 offers a motivating example for such a development. It shows decoder and encoder chains delivering actuator control events and the output of a local 'ganglion' - i.e. a recurrent spiking neural network whose role is perception based on locally available information. This same example could also benefit from a two-dimensional variable-length serial protocol as mentioned above, for added generality. Just as evolution has explored animal designs with highly variable numbers of repeating segments, we perceive an advantage to robot designers in having freedom to chain together arbitrary numbers of local processing or sensorimotor units, for which an octopus-inspired robotic arm could be one example.

V. Conclusion

We have designed an asynchronous, bit-serial, variable-length address-event codec with relative addressing for distributed neuromorphic sensing and computation. This codec works with address-events carrying a one-bit payload, a common design motif in event-based sensors. We plan to use it to communicate bidirectional changes in pressure or other physical quantities. The codec consists in two circuits: an encoder, which increments address-events it receives and merges these with new events triggered by a local event source; and a decoder, which decrements address-events which it received and splits off events with a specific address to deliver to a local event receiver. The QDI design method, paired with the bit serial encoding, makes this implementation dynamically tileable, robust and suitable for deployment on flexible electronics [40], modular sensory platforms and dense on-chip array implementations, all without a circuit design complexity overhead.

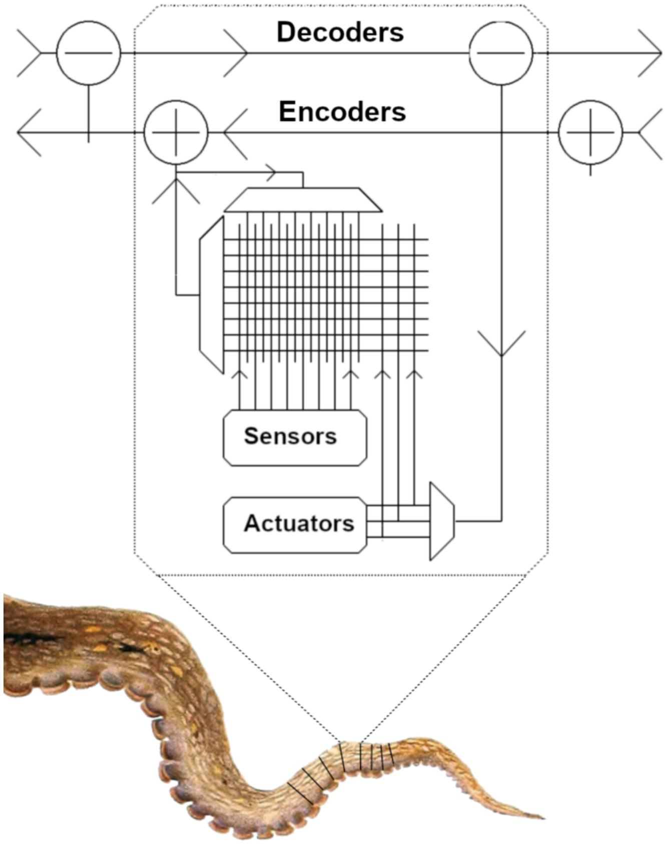


Fig. 13. A possible repeating circuit motif for a segment of a long robotic arm. A serial decoder chain delivers events whose payload is the local address of an actuator. These events join events produced by local sensors to innervate a local recurrent spiking neural network, whose output events undergo local fair hierarchical arbitration to become the payload of events inserted into a serial encoder chain.

Acknowledgment

CB acknowledges the financial support of the National Biodiversity Future Center funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.4 - Call for tender No. 3138 of 16 December 2021, rectified by Decree n.3175 of 18 December 2021 of Italian Ministry of University and Research funded by the European Union – NextGenerationEU. SB acknowledges the financial support from PNRR MUR Project PE000013 ”Future Artificial Intelligence Research (hereafter FAIR)”, funded by the European Union – NextGenerationEU.

References

[1] C. Bartolozzi, G. Indiveri, and E. Donati, “Embodied neuromorphic intelligence,” *Nature communications*, vol. 13, no. 1, p. 1024, 2022.

[2] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[3] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8259423/

[4] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, “Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model,” *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, Jan. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8591981/

[5] P. Purohit and R. Manohar, “Field-programmable encoding for address-event representation,” *Frontiers in Neuroscience*, vol. 16, p. 1018166, Dec. 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2022.1018166/full

[6] P. M. Ros, M. Crepaldi, C. Bartolozzi, and D. Demarchi, “Asynchronous dc-free serial protocol for event-based aer systems,” in *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, 2015, pp. 248–251.

[7] K. A. Boahen, “Point-to-point connectivity between neuromorphic chips using address events,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000.

[8] M. Sivilotti, “Wiring considerations in analog vlsi systems, with application to field-programmable networks,” Ph.D. dissertation, CalTech, 1991.

[9] O. Richter, C. Wu, A. M. Whatley, G. Köstinger, C. Nielsen, N. Qiao, and G. Indiveri, “DYNAP-SE2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor,” *Neuromorphic Computing and Engineering*, vol. 4, Jan. 2024. [Online]. Available: https://iopscience.iop.org/article/10.1088/2634-4386/ad1cd7

[10] K. A. Boahen, “A burst-mode word-serial address-event link-i: Transmitter design,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 7, pp. 1269–1280, 2004.

[11] S. Fok and K. Boahen, “A Serial H-Tree Router for Two-Dimensional Arrays,” in *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. Vienna: IEEE, May 2018, pp. 78–85. [Online]. Available: https://ieeexplore.ieee.org/document/8589987/

[12] “SCX AER protocol,” https://tilde.ini.uzh.ch/∼amw/scx/scx.html.

[13] C. Li, N. Imam, and R. Manohar, “A deterministic neuromorphic architecture with scalable time synchronization,” *Nature Communications*, vol. 16, no. 1, p. 10329, Nov. 2025. [Online]. Available: https://www.nature.com/articles/s41467-025-65268-z

[14] A. J. Martin and M. Nystrom, “Asynchronous techniques for system-on-chip design,” *Proceedings of the IEEE*, vol. 94, no. 6, pp. 1089–1120, 2006.

[15] J. Sparsø, *Introduction to asynchronous circuit design*. Kongens Lyngby, Denmark: DTU Compute, Technical University of Denmark, 2020, oCLC: 1199326564.

[16] W. Hua, Y.-S. Lu, K. Pingali, and R. Manohar, “Cyclone: A Static Timing and Power Engine for Asynchronous Circuits,” in *2020 26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. Salt Lake City, UT, USA: IEEE, May 2020, pp. 11–19. [Online]. Available: https://ieeexplore.ieee.org/document/9179365/

[17] R. Manohar and Y. Moses, “The Eventual C-Element Theorem for Delay-Insensitive Asynchronous Circuits,” in *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. San Diego, CA: IEEE, May 2017, pp. 102–109. [Online]. Available: http://ieeexplore.ieee.org/document/8097392/

[18] N. Imam, F. Akopyan, J. Arthur, P. Merolla, R. Manohar, and D. S. Modha, “A digital neurosynaptic core using event-driven qdi circuits,” in *2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems*. IEEE, 2012, pp. 25–32.

[19] L. Liu and K. Boahen, “Hierarchical Event Readout with Asynchronous Pipelined Opportunistic Merges,” in *2025 29th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. Portland, OR, USA: IEEE, May 2025, pp. 108–117. [Online]. Available: https://ieeexplore.ieee.org/document/11021121/

[20] P. Purohit and R. Manohar, “Asynchronous, event-driven readout for large-scale imaging devices,” in *2025 29th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. Portland, OR, USA: IEEE, May 2025, pp. 118–125. [Online]. Available: https://ieeexplore.ieee.org/document/11021114/

[21] A. Ben Abdallah and K. N. Dang, “Toward robust cognitive 3d brain-inspired cross-paradigm system,” *Frontiers in Neuroscience*, vol. 15, p. 690208, 2021.

[22] S.-C. Liu, A. Van Schaik, B. A. Minch, and T. Delbruck, “Asynchronous Binaural Spatial Audition Sensor With 2x64x4 Channel Output,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 4, pp. 453–464, Aug. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6658899/

[23] P. Lichtsteiner, C. Posch, and T. Delbruck, “A 128 ×128 120 db 15μs latency asynchronous temporal contrast vision sensor,” *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[24] 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 Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan. 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5648367/

[25] M. Yao, O. Richter, G. Zhao, N. Qiao, Y. Xing, D. Wang, T. Hu, W. Fang, T. Demirci, M. De Marchi, L. Deng, T. Yan, C. Nielsen, S. Sheik, C. Wu, Y. Tian, B. Xu, and G. Li, “Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip,” *Nature Communications*, vol. 15, no. 1, p. 4464, May 2024. [Online]. Available: https://www.nature.com/articles/s41467-024-47811-6

[26] G. Rovere, C. Bartolozzi, N. Imam, and R. Manohar, “Design of a qdi asynchronous aer serializer/deserializer link in 180nm for event-based sensors for robotic applications,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 2712–2715.

[27] N. Bingham and R. Manohar, “Self-timed adaptive digit-serial addition,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 9, pp. 2131–2141, 2019.

[28] S. Moradi, N. Imam, R. Manohar, and G. Indiveri, “A memory-efficient routing method for large-scale spiking neural networks,” in *2013 European Conference on Circuit Theory and Design (ECCTD)*. IEEE, 2013, pp. 1–4.

[29] S. A. Bamford, A. F. Murray, and D. J. Willshaw, “Large developing receptive fields using a distributed and locally reprogrammable address–event receiver,” *IEEE transactions on neural networks*, vol. 21, no. 2, pp. 286–304, 2010.

[30] P. Merolla, J. Arthur, R. Alvarez, J.-M. Bussat, and K. Boahen, “A multicast tree router for multichip neuromorphic systems,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, pp. 820–833, 2013.

[31] A. J. Martin, “Compiling communicating processes into delay-insensitive vlsi circuits,” *Distributed computing*, vol. 1, pp. 226–234, 1986.

[32] A. M. Lines *et al.*, “Pipelined asynchronous circuits,” *Master’s thesis, California institute of Technology*, 1995.

[33] S. Ataei, W. Hua, Y. Yang, R. Manohar, Y.-S. Lu, J. He, S. Maleki, and K. Pingali, “An open-source eda flow for asynchronous logic,” *IEEE Design and Test*, vol. 38, no. 2, pp. 27–37, 2021.

[34] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, “A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses,” *Frontiers in neuroscience*, vol. 9, p. 141, 2015.

[35] H. Greatorex, O. Richter, M. Mastella, M. Cotteret, P. Klein, M. Fabre, A. Rubino, W. Soares Girão, J. Chen, M. Ziegler *et al.*, “A neuromorphic processor with on-chip learning for beyond-cmos device integration,” *Nature Communications*, vol. 16, no. 1, p. 6424, 2025.

[36] C. Bartolozzi, P. M. Ros, F. Diotalevi, N. Jamali, L. Natale, M. Crepaldi, and D. Demarchi, “Event-driven encoding of off-the-shelf tactile sensors for compression and latency optimisation for robotic skin,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 166–173.

[37] G. Corniani and H. P. Saal, “Tactile innervation densities across the whole body,” *Journal of Neurophysiology*, vol. 124, no. 4, pp. 1229–1240, 2020.

[38] R. S. Johansson, U. Landstro, R. Lundstro *et al.*, “Responses of mechanoreceptive afferent units in the glabrous skin of the human hand to sinusoidal skin displacements,” *Brain research*, vol. 244, no. 1, pp. 17–25, 1982.

[39] M. Yeasin and R. Sharma, *Foveated Vision Sensor and Image Processing – A Review*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 57–98. [Online]. Available: https://doi.org/10.1007/11504634_2

[40] S. A. Bamford, E. Janotte, and C. Bartolozzi, “Flexible printed circuit sensor,” 2022 WO2023139478A1.