A High Speed Asynchronous Multi Input Pipeline for Compaction and Transfer of Parallel SIMD Data

Christoph Hoppe, Jens Döge, Peter Reichel, Patrick Russell, Andreas Reichel and Peter Schneider
Fraunhofer Institute for Integrated Circuits IIS, Germany
E-mail: christoph.hoppe@eas.iis.fraunhofer.de

***Abstract*—Image sensors with programmable, highly parallel signal processing, so called Vision-Systems-on-Chip, perform computationally intensive tasks directly on the sensor itself. Therefore it is possible to limit the amount of output data to relevant image features only. Reading out such features presents a major challenge, since the position and number of features often is not known. Conventional synchronous buses as well as special event-based readout paths are unsuitable for such a system, since both continuous data, e.g. complete images, and sparse data, like feature coordinates, have to be transfered. A readout path based on an asynchronous pipeline is presented, which supports both readout modes with high speed. Furthermore, a method is introduced that, by serialization, allows for arbitrary data word widths without storing any control information within the data stream. The developed circuit components were measured on a proof-of-concept test chip in a 180 nm CMOS technology and were compared with implementations of asynchronous pipelines found in literature. In addition, the use of the pipeline in a Vision-System-on-Chip, which is still in production, is demonstrated.**

***Index Terms*—asynchronous pipeline, SIMD, sparse data, low latency, high speed, Vision-System-on-Chip, VSoC**

I. Introduction

Image sensors with programmable, highly parallel signal processing allow for performing computationally intensive tasks directly on the sensor. Thus, the output is not necessarily an image, but e.g. the positions and distinctive properties of certain features within the image. Typically, the number of features, as well as their position and time of occurrence, are unknown. A variety of sensors have been proposed in the literature, which are optimized specifically for event-based readout and, in most cases, provide parallelism on pixel level [1], [2], [4]. In order to facilitate feature-based readout, the readout path, instead of being restricted to scanning data sequentially, must be capable of reacting to and propagating events. The Vision-System-on-Chip (VSoC) presented by Döge et al. [3] makes use of a conventional synchronous bus to readout column-parallel data from a Single-Instruction-Multiple-Data (SIMD) [5] unit. Operating at a frequency of up to 40 MHz, it allows for sequential data readout of either all columns of the SIMD array, or at least a consecutive area. However, extracting features may result in sparse data as well as continuous data streams with varying word length. This is why such an approach is not suitable for certain applications, especially if only a few of the SIMD unit's processor elements (PE) actually supply any data. Readout schemes proposed in the literature are based on either continuous data streams, or spatially encoded events, but there is no system that may be applied in both scenarios alike. The proposed asynchronous pipeline is not only supposed to enable high-speed sparse data readout, but also allows for transferring continuous data.

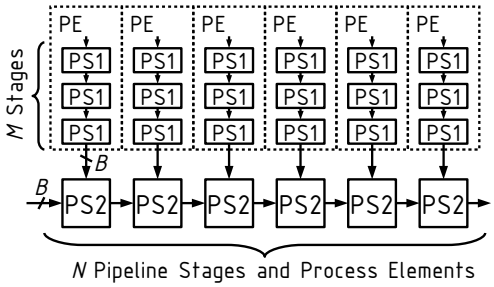


Figure 1: Structure of the Readout Path

The remainder of this paper is structured as follows. In Section II, the structure of the readout path is presented and certain readout modes are discussed. The hardware implementation is discussed in detail in Section III. Results of a test chip are evaluated and compared to the state of the art in Section IV. Finally, Section V provides a conclusion.

II. Structure of the Readout Path

The readout path (Figure 1) is divided into two parts. The first part is a linear pipeline consisting of M stages (PS1) within each PE, serving as a data buffer for the corresponding PE. A second N-stage pipeline runs orthogonally. Each PE is assigned a dedicated pipeline stage with two inputs (PS2), which can take data both from the PE's local data buffer and from the previous PS2. The data bus width B is the same for all elements. One could also imagine a readout path which is organized in a tree structure. This would lead to minimal latency for reading out a single PE. Unfortunately, this requires long distance connections of several millimeters between the pipeline stages and a way more complicated and area consuming routing. This is why this method was not given preference.

In a VSoC, three readout modes are imaginable:

1) Each PE provides data. All data are read out sequentially and in order.
2) An unknown number of PEs provides data, which is read until the pipeline no longer supplies data.

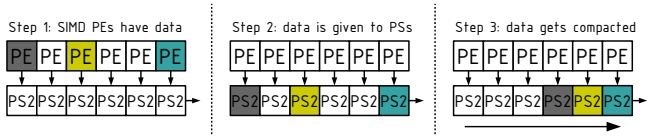


Figure 2: Readout Procedure with Sparse Data

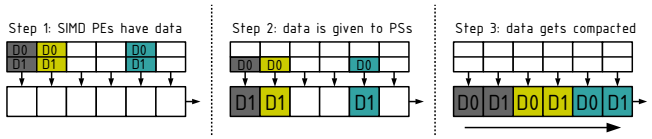


Figure 3: Readout Procedure with Sparse Data of Double Word Length

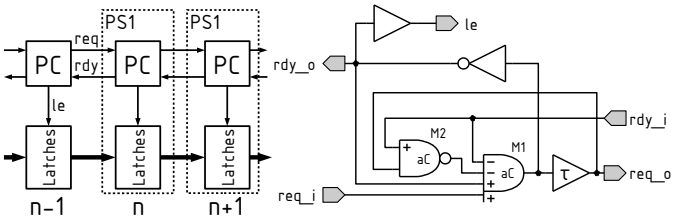
3) An unknown number of PEs provides data, which are continuously readout. Data streams from different processing steps of the PEs can mix.

The first mode is required when complete image information is to be readout. This mode requires the output data to be in order. In the second and third mode, there are sparse data, e.g. coordinates of a detected feature. Figure 2 exemplarily shows the readout process in the second case. Data provided by some PEs in the first step are transferred to the respective PS2s (step 2) and then finally compacted (step 3). The pipeline now holds a continuous data stream of unknown length, which can be readout through a corresponding interface at the end of the pipeline.

If data words of a multiple of the pipeline bit width B are to be output, the trivial solution would simply be to increase the bit width. However, this results in higher area and energy consumption, since more storage elements are required. The bit width B should thus be selected in such a way that minimal data throughput requirements given by the intended application are met. Given a fixed bit width B, larger word widths ($2B$, $3B$, etc.) are serialized before being readout as packets. For this purpose, each PE stores up to M elements in its local buffer, which must then remain contiguous during the transfer to the readout path. Figure 3 exemplarily illustrates the compaction of words of twice the pipeline bit width. In the first step, each PE provides two data elements. In the second step, data are transferred to the respective PS2s. Finally, in the third step, the contiguous data packets are compacted.

The proposed pipeline structure can be implemented both synchronously and asynchronously. However, especially for compaction as well as for readout mode 3, an asynchronous implementation is preferable for several reasons:

- There is no need for a global high speed clock signal (~1GHz). Routing such a signal with low skew requires large buffer trees and considerable power consumption.
- In contrast to a synchronous implementation, the energy requirement is lower in the average case, since it is data-dependent (inherent clock gating).



(a) Linear Pipeline (b) Schematic of the Linear Pipeline Controller

Figure 4: Linear Pipeline and the Corresponding Pipeline Controller

- Current consumption is distributed continuously over time rather than concentrated on a global clock edge.

The last two points are especially important for large pipeline lengths (e.g. $N > 100$), as they occur in a VSoC.

III. Hardware Implementation

Hardware implementation requires all developed components to be placed and routed in a column grid of 8.75 µm. This means that a realization with standard cells is only possible with a large area overhead. Therefore a full-custom implementation is used.

A. Linear Pipeline

The basic structure of an asynchronous pipeline stage consists of a pipeline controller (PC) and an associated data path. A pipelined structure is formed by concatenation of several stages. For example, a three-stage pipeline is shown in Figure 4a. Data transfered through the pipeline are held in latches, which are controlled by the respective PC.

Asynchronous pipelines are characterized by the fact that there is no global synchronization by means of a clock signal, but local synchronization via handshaking signals between successive pipeline stages. Generation and processing of the handshaking signals is carried out by the PC. A lot of implementations are proposed in the literature, with Nowick and Singh [7] giving a good overview. The differences are essentially the choice of the handshaking protocol, the requirement of timing constraints, the choice of data encoding and the logic style used. The pipeline proposed in this paper is based on the so-called *single rail bundled data concept*, due to lower wiring and logic complexity for the data path compared to dual rail encoding. The handshaking protocol is a *4-phase protocol*. The *4-phase protocol* was chosen because it does not require phase conversion when applying control signals or when joining two different pipelines with different phase. In the MOUSETRAP [10] pipeline, for example, additional XOR gates are required to perform phase conversion, which increases latency and area consumption. In the context of the readout path, fast compaction of sparse data is to be made possible, necessitating low forward latency of the pipeline controller. Forward latency is defined by the time required to propagate data from one stage to the next. Furthermore, the storage time within the pipeline is not known in advance,

which is why purely dynamic logic is not applicable. The High Capacity (HC) pipeline, an implementation with very low forward latency, but based on dynamic logic, was presented by Singh and Nowick [11]. The controller presented in this work is, however, designed for static latches. Its function is explained first using a linear controller, before expanding it to a pipeline controller with two inputs.

The PC required for the operation of linear pipelines, see Figure 4b, consists of three gates, two asymmetrical C-elements *M1* and *M2*, as well as an inverter. A C-element [9] is a common gate in asynchronous circuits, e.g. for state encoding, similar to a flip-flop in synchronous implementations. The output of the C-element is 1, if all positive inputs and unlabeled inputs are in state 1 and it is 0, if all negative inputs and unlabeled inputs are in state 0. In all other cases, the output remains unchanged.

The request signal *req_i* (Figure 4b) indicates the arrival of new data, with state 1 meaning new data from the previous pipeline stage has arrived, whereas state 0 indicates that no new data is available. The signal *req_i* of stage n must not change into state 1, until all data of the $n-1$-th stage are stable at the latches of the n-th stage. This condition is, in the case of a pipeline without combinatorial logic inside the data path, i.e. in the case of a FIFO, usually fulfilled by default, while in the case of pipelines with higher data path delays, it is achieved by introducing an additional delay element τ into the request signal. The request signal *req_o* signals the presence of new data to the $n+1$-th stage. It is set when the n-th stage has received and accepted new data, i.e. if *req_i* has been set before and *rdy_o* is in state 1. The signal is generated by the C-element *M1*. The request of the n-th stage remains active, until it is acknowledged by the $n+1$-th stage with a falling edge of *rdy_i*. The C-element *M2* ensures that the *rdy_i* signal was in state 1 between two transfers, i.e. that indeed the falling edge is evaluated. After being acknowledged, the request signal *req_o* changes again to state 0 and the n-th stage becomes ready for a new transfer.

For the pipeline controller to operate correctly, several – fortunately easily satisfiable – timing constraints have to be met. The first constraint (1) ensures that the data path is faster than the control path, with t_{M1}, t_{delay}, t_{skew} and t_{DQLatch} being the delay of C-element *M1*, the delay of delay element τ, the skew of the latch-enable buffers and the data delay of a transparent latch, respectively.

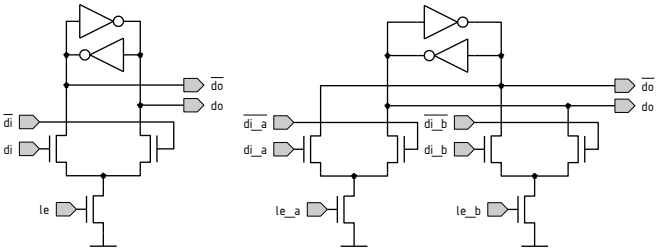
$$t_{\text{M1}} + t_{\text{delay}} + t_{\text{skew}} > t_{\text{DQLatch}} \tag{1}$$

The second constraint (2) defines a minimum pulse length at the gate input of the latches t_{GLatch}, which ensures correct sampling of the latches, with t_{inv} being the delay of the inverter.

$$t_{\text{M1}} + t_{\text{inv}} \geq t_{\text{GLatch}} \tag{2}$$

The third and final constraint (3) ensures that C-element *M2* must have recognized that *rdy_i* had been in state 1 before it returned to state 0.

$$t_{\text{M1}} + t_{\text{inv}} > t_{\text{M2}\downarrow} \tag{3}$$



(a) Latch with One Data and Select Input (b) Latch with Two Data and Select Inputs

Figure 5: Latch Implementations for the Data Path

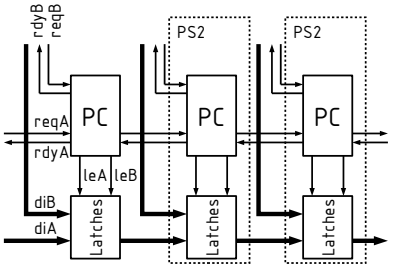


Figure 6: Multi Input Pipeline

In (1) it is specified that the delay time of the latches must be shorter than the one of the control path. Since the control path only has a delay equivalent to one C-element, this condition can not be realized with conventional C²MOS latches. Furthermore, in order to avoid additional multiplexers, latch implementations that are easily expandable by a second input, should be given preference. Figure 5a shows such a latch implementation consisting of a differential pair and two feedback inverters. By adding a second differential pair, this implementation can be easily modified into a latch with two inputs, as shown in Figure 5b. The delay time of the latches corresponds to that of two inverters and is, if sized appropriately, faster than a C-element's delay time of at least two inverter stages.

B. Multi Input Pipeline

Starting from the linear pipeline controller, as it is used inside the PEs to provide contiguous packets, it is now expanded by arbitration to support multiple inputs. The structure of the resulting pipeline is shown in Figure 6. It consists of $N+1$ inputs and one output for N stages (cf. Figure 1). An arbiter selects either input A or input B, depending on which of the two inputs provides new data. The channel selection is not changed, until no successive data are available for a given duration, i.e. a continuous transfer was interrupted. This allows for the realization of contiguous packets without having to evaluate special control information within the data words.

The arbiter, see Figure 7, consists of an element enforcing mutual exclusion (MUTEX), which, in case of competing requests, selects one of the two channels in a glitch-free manner.

The MUTEX consists of a NAND RS flip-flop followed by a metastability filter – a well-established implementation, that is also frequently used in other applications [6], [13]. The additional OR gates at the input hold the actual channel selection for a certain time. The induced slowdown is insignificant, since the NAND and OR gate can be combined into a single-stage CMOS gate. The hold time is set using delay elements and should be chosen somewhat larger than the cycle time of a pipeline stage. If the selected delay elements do not propagate rising edges considerably faster than falling ones, incoming request signals may be too short to be recognized at the output of the delay element. This can be accomplished using a chain of inverters combined with NAND respectively NOR gates. As long as no active channel has been selected, the delay time of the arbiter has a direct influence on the forward latency of the pipeline controller. However, this cannot be avoided due to unknown temporal relations between the incoming request signals.

Figure 8 depicts the pipeline controller with two inputs. The outgoing request signal *req_o* is once again generated by the C-element *M1*. The additional logic at *M1*'s positive input makes sure that the outgoing request is only set, if the condition $(req_{\mathrm{a}} \wedge sel_{\mathrm{a}} \wedge rdy_{\mathrm{a}}) \vee (req_{\mathrm{b}} \wedge sel_{\mathrm{b}} \wedge rdy_{\mathrm{b}})$ is fulfilled. This means that a request from channel A is accepted only if the arbiter has selected channel A and the latches in the data path sample data from channel A. The same conditions hold for channel B, in a similar fashion. As this constraint may easily be integrated into the NMOS paths of the C-element, there are in fact no additional gates required, as suggested by the principle circuit in Figure 8. After setting the signal *req_o* in the n-th stage, as in the case of the linear controller, the signal *rdy_i* is set to 0 by the $n+1$-th stage. Following that, the signal *req_o* in the n-th stage is set to 0 as well. The latch enable signals *le_a* or *le_b* are logically linked by AND gates with the signals *sel_a* or *sel_b*, respectively. The signals *rdy_a* and *rdy_b* are generated in a similar manner. The channel select signals *sel_a* or *sel_b* once again decide whether *rdy_a* or *rdy_b* must be set. C-elements must be used at this point, because the ready signals must not change their state during channel switch. Otherwise, the previous pipeline stage may enter an invalid state.

For the controller to operate correctly, timing constraints have to be taken into account, as well. The constraints given in (1) to (3) for linear controllers apply to multi input pipeline controllers in a similar fashion, with the term $t_{\mathrm{M1}} + t_{\mathrm{inv}}$ now being replaced by t_{M1}. This is explained by the fact that, for the linear controller, all latches are transparent if the pipeline stage does not hold valid data. However, in the case of the extended pipeline controller, the latches are not transparent because the channel to be used is not selected yet. This is why the constraints given in (2) and (3) are somewhat more difficult to meet. No further timing constraints are necessary.

C. Multi Input Pipeline without MUTEX

If the readout mode 3, i.e. the continuous readout of data from different processing steps of the PEs, is not required,

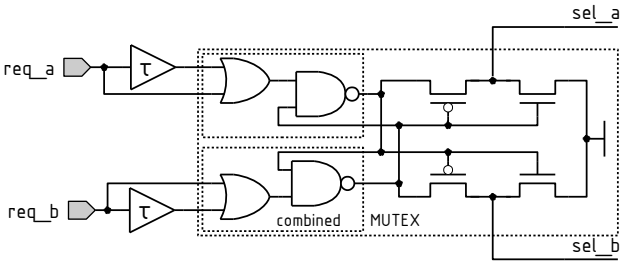


Figure 7: Arbiter for Channel Selection with Channel Hold Function

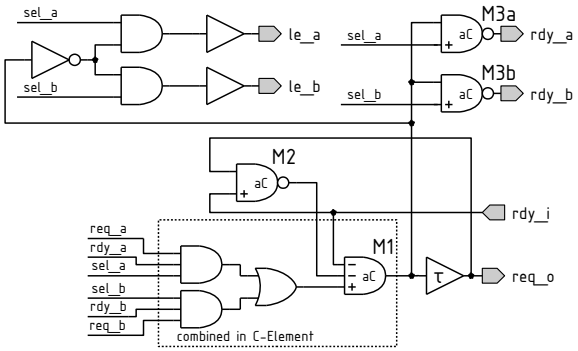


Figure 8: Schematic of the Two-Input Pipeline Controller

it is possible to omit the MUTEX. This is only needed if a priori is not known when which of the two inputs must be selected. In readout modes 1 and 2, however, it is clear that first data has to be read from the local pipelines of the PEs and then only data from the previous horizontal pipeline stage. This can be used by first forcing channel B at the start of the readout process and keeping it selected until the PE's local pipeline no longer contains any data. Then there is a change to channel A which will be kept selected for the rest of the reading process. This variant offers the advantage that the additional delay introduced by the MUTEX when a channel is changed is eliminated and instead a maximum of one channel selection per readout process takes place. To implement this function, a token-based channel selection is used instead of the MUTEX, as shown in Figure 9. The flip-flop stores the current channel selection and can be set to channel B (*select_local* = 1) by a rising edge of the *start* signal. However, this happens only as long as at least one of the local request signals ($req_0, \ldots, req_{\mathrm{M\text{-}1}}$) is active. As soon as none of the request signals and thus *local_available* is no longer set and the PS2 pipeline controller no longer carries out a transfer (*req_o* = 0), the flip-flop is reset and channel A is selected. This selection remains until the next *start* edge. In addition to the *start* signal, a reset of the channel selection can be forced by using $\overline{reset}$. Care must be taken with the generation of the *local_available* signal. The method shown here is only valid if no new data is generated in the

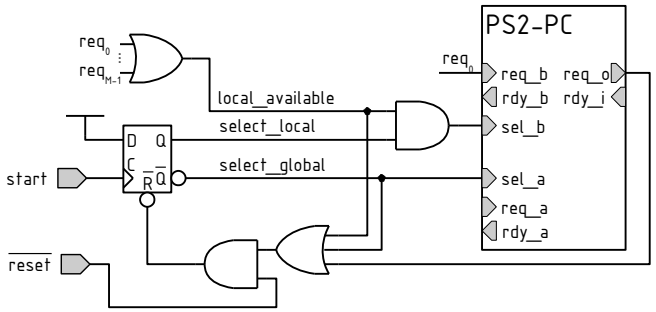


Figure 9: Schematic of an Alternative Channel Selector

PE while the pipeline is readout. Otherwise, there could be a race-condition when new data arrives in the PE while the PS2 stage is sampling *local_available*.

D. Initialization

So far, two pipeline controllers, the corresponding data path latches and two possible channel arbitration schemes have been presented. The very important aspect of initializing the pipeline, however, has not been covered at all. Similar to synchronous circuits, a reset signal is used for this purpose. This signal, which resets all C-elements at once, may be realized by expanding each C-element by just a single transistor. Controlling data transfer from the PEs' data buffers to the output path is another important aspect. This necessitates an additional AND gate controlled by a global start signal to be added to the request signals between PEs and PS2s. Otherwise, data elements from the PEs would be immediately fed into the readout pipeline. In this way, it would not be possible to readout the pipeline and build up new data in the PEs' data buffers at the same time. For the Multi Input Pipeline without MUTEX, this data flow control is already included and does not need to be provided separately. In case of a long pipeline, the start and reset signals must be fed in either via a buffer tree or in the opposite direction to the pipeline data flow. This is comparable to the supply of a clock signal with long shift registers.

IV. Test Results and Comparison

A. Proof-of-Concept

The presented pipeline controllers were demonstrated to operate correctly using a test chip in a 180 nm low-power CMOS technology. The test chip includes a 14-stage asynchronous pipeline with an 8-bit data path. The inputs and outputs can be read or written via synchronous interfaces. Two of the 14 pipeline controllers have been replaced by controllers with arbitration, i.e. the pipeline has a total of three inputs. The two additional inputs are equipped with one buffer stage each. Furthermore, the pipeline can be connected as a ring structure, in order to circulate a defined number of data elements. Handshaking was monitored by using strong buffers to output selected request and ready signals to bond pads. A schematic representation of the structure is given in Figure 10.

Pipeline Design	Cycle Time in ns	Throughput in Giga Items per Second	Logic Style	Process Node
PS0 [12]	1.98	0.51	static	0.18µ HP
LPsr2/2 [12]	0.76	1.31	dynamic	0.18µ HP
LPsr2/2 [12]	0.65	1.55	dynamic	0.18µ HP
HC [11]	0.57	1.75	dynamic	0.18µ HP
this work	1.1	0.91	static	0.18µ LP

Table I: Comparison of Different Asynchronous Pipelines

In total, there are four test modes available:

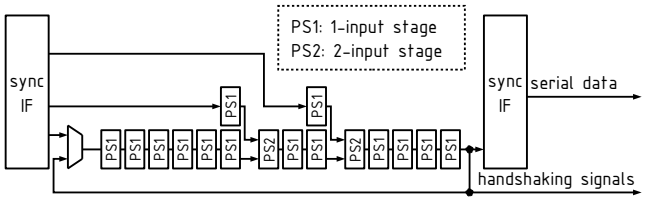
- In test mode 1, all pipeline elements are filled in a defined order and these are subsequently read out. Arbitration and correctness of the readout data can be checked.
- In test mode 2, data are written only to the additional pipeline inputs. This makes it possible to check whether the compaction delivers correct data at the output.
- In test modes 3 and 4, the pipeline is connected as a ring, i.e. all items that were inserted in the first place, keep circulating within the pipeline. In this mode, the pipeline controllers' forward latency as well as their cycle time can be measured. The forward latency is determined by inserting exactly one element. In the case of 14 pipeline elements and assuming that all of them are approximately equal in speed, the forward latency results from the rotation time of the data item according to (4).

$$t_{\text{Forward}} \approx t_{\text{Ring1}}/14 \tag{4}$$

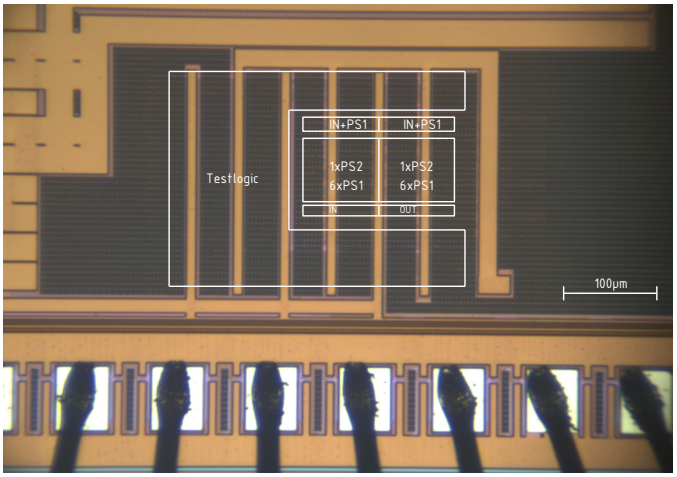
$$t_{\text{Cycle}} \approx t_{\text{Ring13}}/14 \tag{5}$$

Similarly, the cycle time of the pipeline controllers can be determined by inserting 13 elements into the pipeline according to (5). In this case, there is exactly one pipeline stage without a valid data item. In Figure 11, the measured forward latency as well as the cycle time depending on the supply voltage are shown. As expected, the controllers' speed increases with higher supply voltage. Even at a supply voltage of only 0.7 volts, some of the test chips are still functional. In addition, the power consumption was measured for the case that only one element circulates in the ring (Figure 12).

Table I shows the delay of the presented pipeline controllers compared to variants in the literature. Compared to the HC pipeline [11], the implementation presented here is only about half as fast, which is, on the one hand, due to the use of static logic in the data path, on the other hand due to different process technologies. Furthermore, the values given in the literature only apply to linear pipelines. Because of the delay induced by the arbiter, the multi-input pipeline presented is inevitably a little slower. The measured forward latency is 520 ps at a nominal operating voltage of 1.8 V. This corresponds to an equivalent clock frequency of 2 GHz for a synchronous implementation, which is not feasible with the chosen 180 nm technology. Under the same test conditions, a cycle time of 1.1 ns was measured. This corresponds to the throughput of the pipeline when all stages hold a data element.



(a) Circuit Structure



(b) Chip Photograph

Figure 10: Proof-of-Concept Test Chip

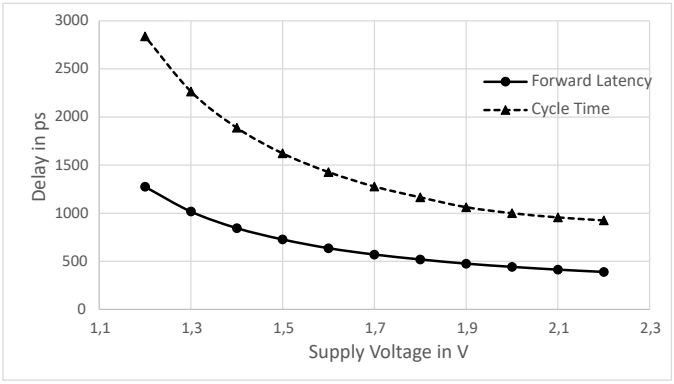


Figure 11: Measured Forward Latency and Cycle Time as a Function of Supply Voltage

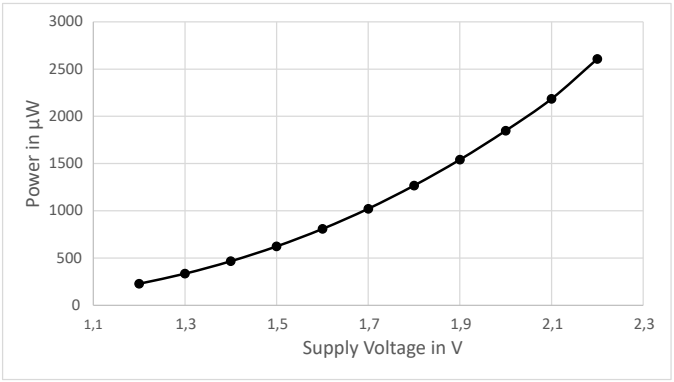


Figure 12: Measured Power Consumption as a Function of Supply Voltage

Parameter	Synchronous Bus in [3]	Asynchronous Pipeline in this work
PE interface bitwidth	2x8	8
bus bitwidth	32	8+1
local data buffers	2x8 Bit	4x8 Bit
possible packet length	8, 16	8, 16, 24, 32
maximum operating frequency at output	40 MHz	114 MHz*
maximum throughput continuous data	1.3 GBit/s	7.3 GBit/s*
maximum latency for a single element read	6.4 µs (8 Bit) 12.8 µs (16 Bit)	0.5 µs*
possibility to compact sparse data	no	yes
area PE interface	480 µm²	590 µm²
area local data buffer	280 µm²	300 µm²
total area	1040 µm²	1790 µm² (4 buffers) 1190 µm² (2 buffers)
maximum energy per transfered byte	500 pJ/byte	690 pJ/byte*
energy per complete row (1024 Byte)	512 nJ	356 nJ*

*) based on measurements of proof-of-concept test chip

Table II: Comparison Synchronous Bus vs. Asynchronous Pipeline in a VSoC

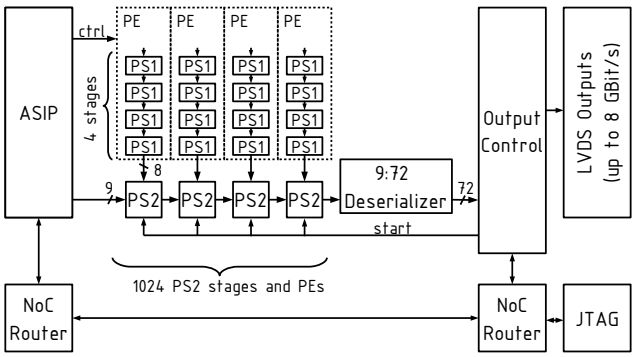


Figure 13: Multi Input Pipeline in a VSoC

B. Implementation in a VSoC

After validating the functionality in a proof-of-concept design, the pipeline was implemented on full scale in a VSoC. It consists of $N = 1024$ PS2 stages and $M = 4$ PS1 stages in the local PEs. The bit width is $B = 9$ for the PS2 stages and $B = 8$ for the local PS1 stages. By default, the MSB is set to '0' by the local pipelines and can only be set at the beginning of the horizontal pipeline. It is used to mark the end of a line. In contrast to the proof-of-concept, PS2 stages without MUTEX are used in this design. In addition, the pipeline is embedded in a complete system consisting of an application-specific instruction-set processor (ASIP), a SIMD unit, a LVDS (low voltage differential signaling) output and a network on chip (NoC). In contrast to the proof-of-concept, this allows considerably more extensive tests, which can be completely defined in software. In the VSoC, data output from the pipeline are combined at the output into 72 bit packets for subsequent processing in a synchronous design with a moderate clock frequency of up to 125 MHz. Combining the data is also asynchronous and uses the featured pipeline controllers and latches. The aggregated data can either be routed outwards via a parallel LVDS interface or read back by the ASIP via an asynchronous network on chip [8]. The start of readout, the packet length used and the output data are controlled by the ASIP, which has access to the PEs of the SIMD unit. It can also control the readout of the pipeline via the NoC integrated in the VSoC. Figure 13 shows the embedding of the pipeline and the peripheral components required for the tests in the VSoC. Compared to a previous implementation as a synchronous bus by Döge et al. [3], the asynchronous readout path yields a number of advantages, which are summarized in Table II. Since the VSoC is not physically available yet, the measurement results were approximated from the proof-of-concept test chip. The comparison shows that the asynchronous pipeline achieves continuously six times more data throughput, although the bus width is only a quarter of the synchronous implementation. There is a clear difference in the maximum latency to read a single data element. In the case of 16 bit data with 0.5 µs latency this is 25-times lower than in the synchronous version. The area requirement for the same data buffer size is about 15 % higher for the asynchronous implementation. The maximum energy requirement per transmitted byte is slightly higher with the asynchronous implementation with 690 pJ than with the synchronous variant with 500 pJ. However, for a complete readout of an entire row, the asynchronous variant is about 30 % more energy efficient and requires 356 nJ.

V. Summary

This work proposes a readout path for a Vision-System-on-Chip, which is capable of serially outputting parallel data of a column-parallel SIMD unit. The readout path is implemented as an asynchronous pipeline and therefore enables both very fast compaction of sparse data and readout of continuous data streams. Through clever arbitration, it is possible to transmit contiguous data packets, and thus variable word widths, without additional control information. The circuit components presented were successfully tested on a proof-of-concept chip in a 180 nm CMOS technology. With 520 ps measured forward latency at nominal operating voltage, the proposed asynchronous pipeline has been found to equal a corresponding synchronous implementation clocked at 2 GHz for the compacting operation. The measured cycle time of 1.1 ns equals a clock frequency of 910 MHz for the readout of compacted data. The readout path shows a 25-times improvement in readout latency compared to a previous implementation as a synchronous bus while taking 15 % more area and 30 % less energy consumption when a whole image row is readout.

Acknowledgement

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the Innovation Initiative "Entrepreneurial Regions", project consortium 3Dsensation, project cSoC-3D, grant number 03ZZ0427E. The authors of this paper are solely responsible for its content.

References

[1] Raphael Berner, Christian Brandli, Minhao Yang, S-C Liu, and Tobi Delbruck. A 240x180 120db 10mw 12us-latency sparse output vision sensor for mobile applications. In *IEEE International Image Sensor Workshop*, pages 41–44, 2013.

[2] Gaozhan Cai, Bart Dierickx, Bert Luyssaert, Nick Witvrouwen, and Gerlinde Ruttens. Imaging sparse events at high speed. In *IEEE International Image Sensor Workshop*, 2015.

[3] Jens Doege, Christoph Hoppe, Peter Reichel, and Nico Peter. A 1 Megapixel HDR Image Sensor SoC with Highly Parallel Mixed-Signal Processing. In *IEEE International Image Sensor Workshop*, 2015.

[4] A Dupret, B Dupont, M Vasiliu, B Dierickx, and A Defernez. CMOS image sensor architecture for high-speed sparse image content readout. In *IEEE International Image Sensor Workshop*, pages 26–28, 2009.

[5] Michael J Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966.

[6] Yu Liu, Xuguang Guan, Yang Yang, and Yintang Yang. An asynchronous low latency ordered arbiter for network on chips. In *2010 Sixth International Conference on Natural Computation*, volume 2, pages 962–966. IEEE, 2010.

[7] Steven M Nowick and Montek Singh. High-performance asynchronous pipelines: an overview. *IEEE Design & Test of Computers*, 28(5):8–22, 2011.

[8] Patrick Russell, Jens Doege, Christoph Hoppe, Thomas B Preusser, Peter Reichel, and Peter Schneider. Implementation of an asynchronous bundled-data router for a GALS NoC in the context of a VSoC. In *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2017 IEEE 20th International Symposium on*, pages 195–200. IEEE, 2017.

[9] Montek Singh and Steven M Nowick. High-throughput asynchronous pipelines for fine-grain dynamic datapaths. In *Advanced Research in Asynchronous Circuits and Systems, 2000.(ASYNC 2000) Proceedings. Sixth International Symposium on*, pages 198–209. IEEE, 2000.

[10] Montek Singh and Steven M. Nowick. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines. In *Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference On*, pages 9–17. IEEE, 2001.

[11] Montek Singh and Steven M Nowick. The design of high-performance dynamic asynchronous pipelines: high-capacity style. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(11):1270–1283, 2007.

[12] Montek Singh and Steven M Nowick. The design of high-performance dynamic asynchronous pipelines: lookahead style. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(11):1256–1269, 2007.

[13] J Sparso. Asynchronous Circuit Design - A Tutorial. *Chapters 1-8 in Principles of asynchronous circuit design-A systems Perspective'*, pages 1–152, 2006.