Asynchronous Interface FIFO Design on FPGA for High-throughput NRZ Synchronisation

DOI:
10.23919/FPL.2017.8056801

Document Version
Accepted author manuscript

Published in:
2017 27th International Conference on Field Programmable Logic and Applications (FPL)

Asynchronous Interface FIFO Design on FPGA for High-throughput NRZ Synchronisation

Gengting Liu, James Garside, Steve Furber, Luis A. Plana, Dirk Koch
School of Computer Science, University of Manchester
Manchester, United Kingdom, M13 9PL
Email: {gengting.liu, james.garside, steve.furber, luis.plana, dirk.koch}@manchester.ac.uk

***Abstract*—Networks-on-chip (NoCs) have become a new chip design paradigm as the size of transistors continues to shrink. Globally-asynchronous locally-synchronous (GALS) on-chip networks are proposed for solving issues such as large clock tree distribution and signal delay variations. More interestingly, for the GALS networks using m-of-n delay-insensitive interconnect, the asynchronous interconnect not only can be used for on-chip interconnection, but also provides a simple, direct and power-saving solution for off-chip interconnection.**

This paper presents an asynchronous interface FIFO design to improve throughput over asynchronous inter-chip links using 2-of-7 Non-Return-to-Zero (NRZ) encoding in an existing many-core system. The proposed design is suitable for implementation on commodity FPGAs without using the limited global clock buffer resources, but involves using the FPGA to implement asynchronous circuits. The interface FIFO is constructed from the transition detectors themselves rather than by employing a separate buffer in the more conventional fashion. The proposed solution has been demonstrated in an existing system and is suitable for adaptation to other asynchronous m-of-n NRZ coding protocols for high-throughput communication.

I. Introduction

Packet switched network-on-chip (NoC) [1] [2] architectures are proposed to replace bus-based networks for the integration of a large number of design blocks. These blocks are often confined to different clock domains for easy timing closure; thus passing the signals between different clock domains has become a normal design practice.

Globally-asynchronous locally-synchronous (GALS) networks [3] are proposed to solve the problems of large clock tree distribution, delay variations and dynamic power consumption. The delay-insensitive m-of-n asynchronous protocol [4] can be used in GALS systems to simplify the implementation of on-chip network interconnect, as well as inter-chip connection. However, chip-to-chip communication is a critical factor and the more important objectives are latency, throughput and power consumption [5]. In this paper, we investigate the delay-insensitive asynchronous communication scheme in an existing many-core GALS system [6]. Each chip in the system has 6 asynchronous links that can be used to connect multiple chips in a hexagonal mesh. The asynchronous link is bidirectional and comprises two independent channels, a transmitter (Tx) and a receiver (Rx), as shown in Figure 1.

The channels are implemented with a 2-of-7 non-return-to-zero (NRZ) encoding [7] to minimise the number of transitions required; a single NRZ 'acknowledge' wire completes the handshake cycle. The 2-of-7 protocol was chosen for the implementation because it has higher bit-transfer rate per wire than the traditional dual-rail and 1-of-4 encodings. Therefore, each link has 16 wires in total for both channels. These links can be connected between the custom chips across a PCB without timing concerns.

FPGAs can be used to interface multiple asynchronous links and accumulate the communication speed. However, commodity FPGAs are optimised for synchronous designs. It is therefore important to capture the NRZ asynchronous signals and convert them into a more tractable form as soon as possible. A second conversion, from the synchronous domain of a receiving FPGA back to an asynchronous link, is also performed. The following descriptions apply primarily to the buffer leading from the asynchronous domain to the synchronous transmitter. An analogous process, in practice less complex because the data translation is easier, can be used between the synchronous receiver and the asynchronous links on the destination PCB.

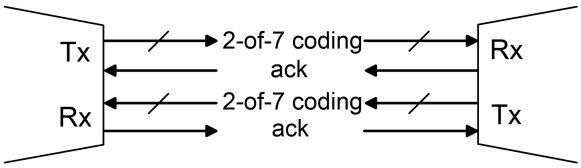

Fig. 1: A 2-of-7 asynchronous link

For digital designs, synchronisation is required to handle the metastability problem [8] [9] from external asynchronous signals and prevent the synchronous circuit entering a metastable state. Using a pair (or more) of flip-flops in series is a simple approach to synchronisation. Figure 2 shows two-stage synchronisers inserted in an asynchronous communication circuit forming a handshaking loop. This type of synchroniser imposes a delay which is large enough to be unacceptable in the cycle time of each flit. If a 4-phase hand-shake protocol is applied, the cycle time is doubled because an extra loop is required to finish the return-to-zero handshaking phases.

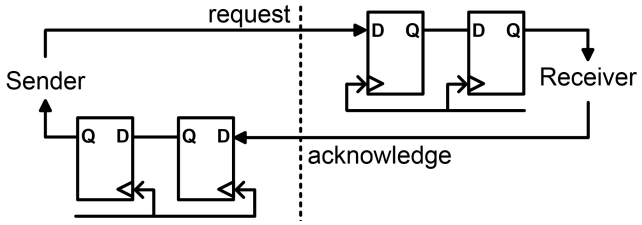

Fig. 2: An asynchronous communication handshaking loop

II. Related Work

The performance issue of interfacing between different circuit domains has become one of the main problems to overcome in various GALS networks. A number of researchers have investigated the designs of reliable high-speed GALS network interfaces. Dolkin et al. analyse the synchronisation issues [10] in GALS systems. A bi-synchronous interface FIFO for two clocked domains in a GALS system is proposed by Panades et al. [11]. Asynchronous-to-synchronous and synchronous-to-asynchronous interfaces using conventional FIFOs [12] for Return-to-Zero (RZ) synchronisation were developed by Beigne et al. [13] [14]. A fast transmitter from synchronous to asynchronous domains by employing a predictive sending scheme is also investigated by Yousefzadeh et al [15].

This paper presents a complete interface FIFO design between asynchronous and synchronous domains for high-throughput NRZ synchronisation. In contrast to previous work, the proposed design works for the NRZ protocol and is suitable for the implementation on commodity FPGAs.

III. Potential Synchroniser Designs

Figure 3 shows the base design, which is completely synchronous. This solution synchronises the NRZ data at the input. A synchronous level-sensitive edge detection circuit is implemented in the subsequent module. When a valid flit is detected, the circuit enables the memory block to latch the data and acknowledge the circuit.

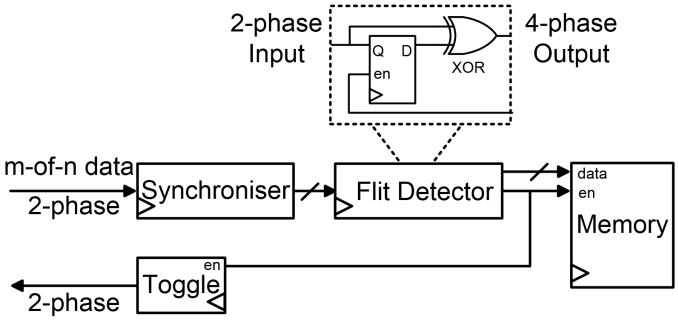

Fig. 3: Immediate synchronisation

Practice reveals that the throughput of this design is limited. This is because the round-trip latency is impacted by the synchroniser. Let's consider an example with an FPGA running at 300 MHz, a 2-of-7 code that transfers 4 bits data per transaction and a latency of 8 ns on both sides, then the transfer rate would be not more than $1/(2 * (3.33\ ns * 2 + 8\ ns))$. The asynchronous link functions correctly but becomes a network bottleneck.

A more promising approach is to use a FIFO buffer allowing asynchronous insertion (and acknowledgement) of each flit with the synchronisation latency 'concealed' between this and the synchronous read process. This removes the synchronisation penalty from the cycle at peak throughput; the response of the asynchronous controller is still the critical timing factor which can be minimised.

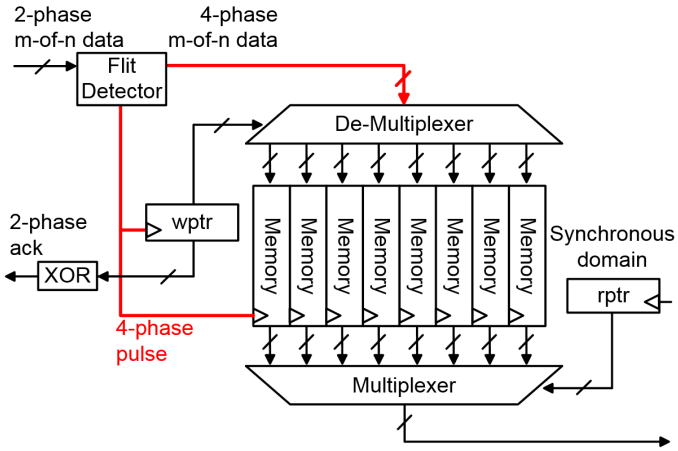

Fig. 4: Synchronising through a conventional FIFO

A speedup solution using a FIFO is shown in Figure 4. Here the flit insertion into the FIFO is asynchronous and the flit removal from the FIFO is synchronous; the synchronisation (not explicitly shown) is done between the two pointers to indicate the status of the FIFO, such as whether the buffer is empty or not.

This approach requires the construction of self-timed circuits on the FPGA. In the cycle described above the incoming signal triggers a series of sequential steps.

1) Two transitions arrive (asynchronously, independently) on input wires. Each active input signal is translated into a level using an edge detector (more details in Section III-A).
2) A completion detector (Section III-B) identifies that a complete flit has been received, synchronising the two input signals.
3) The input code is copied into a conventional asynchronous FIFO
4) The appropriate FIFO pointer is incremented.
5) The acknowledge signal is toggled; The edge detector is reset in parallel – in this case using a self-timed pulse.

A. Transition-sensitive asynchronous Edge Detector

A custom fault-tolerant edge detection circuit was proposed by Shi et al. [7]. A functionally equivalent implementation can be constructed using D-type flip-flops (Figure 5), of which the FPGA has an abundant supply. In this circuit, inputs are connected to logic one and the circuit is driven by the signal's transition. The upper D-type flip-flop is used to detect a rising edge of the transition signal and the bottom D-type flip-flop detects a falling edge. Once the first valid transition is detected, the output is asserted. After the circuit is reset, it will be ready for detecting the next transition. However, the output signal will not be de-asserted until the circuit is reset, therefore any glitches between the first asserted output and the circuit reset will be tolerated, giving the circuit the fault-tolerant feature. This edge-detector also performs the function of converting the 2-phase input into a 4-phase output. The circuit needs to be reset by a 4-phase signal because the reset of a D-type flip-flop is level sensitive.

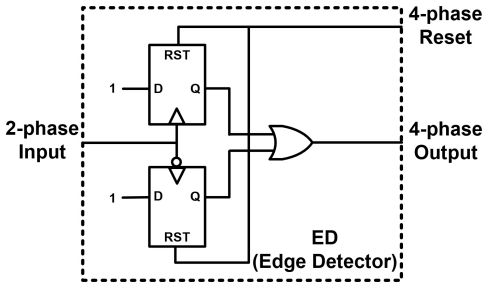

Fig. 5: One-bit transition-sensitive edge detector on FPGA

B. Two-of-Seven Flit Detector and Coding Protocol

The flit detector combines the outputs from seven edge detectors with a 'completion detector' circuit to validate the arrival of a single flit. Each edge detector also contains a reset signal which is not explicitly shown in the diagram. In the 2-of-7 coding protocol, there are 21 possible symbols. From among these symbols, 16 are chosen to encode 4-bit values and a further one is used for the EoP (End of Packet) signal. Table I shows the 2-of-7 coding table which is designed to simplify the logic needed to detect a complete flit, and the corresponding completion detector is shown in Figure 6.

C. Non-Return-to-Zero Acknowledge signal

To finish the communication cycle, a Gray [16] encoded pointer is designed to generate the 2-phase acknowledge signal. The signal can be generated from the parity (exclusive-OR tree) of the pointer and the Gray encoded pointer can be used for synchronisation.

TABLE I: 2-of-7 coding table

	Value																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	EoP
6th	-	-	-	-	-	-	-	-	T	T	T	T	-	-	-	-	T
5th	-	-	-	-	T	T	T	T	-	-	-	-	-	-	-	-	T
4th	T	T	T	T	-	-	-	-	-	-	-	-	-	-	-	-	-
3rd	-	-	-	T	-	-	-	T	-	-	-	T	-	-	T	T	-
1st	-	T	-	-	-	T	-	-	-	T	-	-	T	T	-	-	-
2nd	-	-	T	-	-	-	T	-	-	-	T	-	-	T	T	-	-
0th	T	-	-	-	T	-	-	-	T	-	-	-	T	-	-	T	-

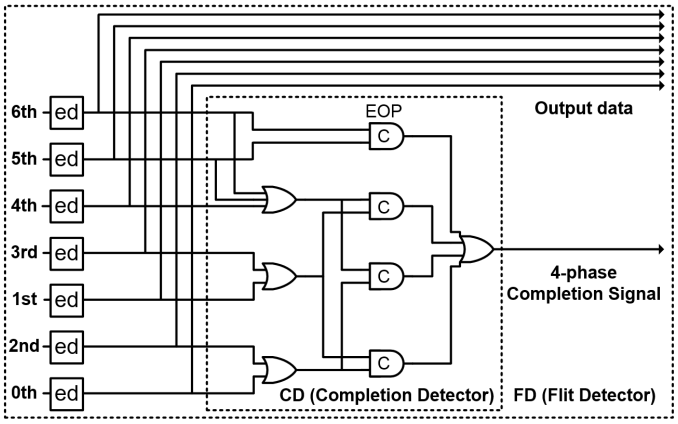

Fig. 6: Two-of-seven flit detector

D. Design concerns of using a conventional asynchronous FIFO

Employing a FIFO (Figure 4) increases the communication throughput because the asynchronous circuits can acknowledge the asynchronous link without stalling for synchronisation. The asynchronous circuit is designed with as few components as possible to minimise the delay impact in the critical path. This relieves the bottleneck between the asynchronous link and the synchronous circuit.

However, there are a number of concerns in implementing such circuits in an FPGA typically intended for synchronous designs, and supported by tools which rely on clock assumptions. Manual intervention can be necessary to alleviate several types of potential problems with the FPGA placement.

Timing constraints need to be satisfied, such as flip-flop set-up and hold times, without introducing inordinate delays. For example, the incoming code is held in the set of edge detectors and its validity is indicated by the completion detector; the data must reach the FIFO and be set up before the completion-detected edge arrives, otherwise the set-up time is violated. When the edge detectors are reset, the pulse must reach all flip-flops intact – and be removed before a subsequent input can arrive.

One way of alleviating the 'clocking' of the input registers could be to use the clock distribution networks of the FPGA. However the *latency* of these would be inordinately high, and there would not be enough of these networks for interfacing multiple asynchronous links.

Glitch control is also a potential problem. Synchronous designs can afford to neglect the possibility of glitches; asynchronous designs cannot always do so and it is important to prevent their possibility in control circuits. These can be alleviated by appropriate design choices, such as using Gray codes, but race hazards should still be considered in the design.

IV. Improved Interface FIFO Design

The FIFO synchroniser can hide the synchronisation latency. However, using the conventional FIFO in the asynchronous design imposes some timing assumptions on the storage elements, which are hard to control in layout. Therefore, an improved interface FIFO is presented in Figure 7. The storage elements are constructed using the transition detectors, which removes the above timing assumption. In addition, the acknowledging arc is shortened in the following 4 sequential actions. Rather than *copying* the recovered flit into a FIFO the edge/flit detector can *become* a stage in the FIFO. This means that the acknowledgement can be transmitted as soon as the completion detector has verified the flit and the pointer has moved to ensure the subsequent flit is directed elsewhere. Now the series of sequential actions shortens as follows:

1) Two transitions arrive (asynchronously, independently) on input wires; each active input signal is translated into a level using an edge detector.
2) A flit detector identifies that a complete flit has been received.
3) The appropriate asynchronous FIFO pointer is incremented and directs the next input to the next flit detector.
4) The acknowledge signal is toggled and the edge detector is reset.

The improved synchroniser solves the design concerns discussed in the previous section. First, it does not have the set-up and hold time violation hazard that arises when using a conventional FIFO for NRZ synchronisation, because the design of the flit detector is asynchronous. The data is saved in the detection circuit, not moved to another separate buffer. Second, a 4-phase reset signal can be generated from the Gray encoded pointers.

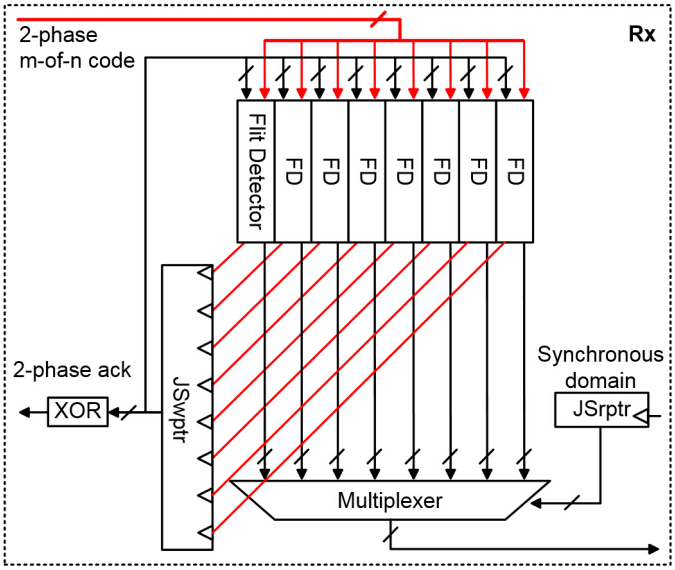

Fig. 7: Asynchronous interface FIFO

A. Encoding for Asynchronous Pointers

The input pointer '*JSwptr*' in Figure 7 is realised asynchronously too. This is in the form of a Johnson counter [17] but with each flip-flop clocked by its own flit detector. The active position in the FIFO is indicated by the circulation of a single, 2-phase edge; the edge can be detected by exclusive-OR gates (shown in Figure 8) and used to enable the subsequent flit detector in the cycle. The timing requirement here is that the movement of the enable level (a 'one hot'

code) needs to be settled before the arrival of the next flit. As the competing timing constraint is to another chip and back, this timing requirement will not be violated in practice. The 2-phase flit acknowledge can be produced from the parity of the pointer output.

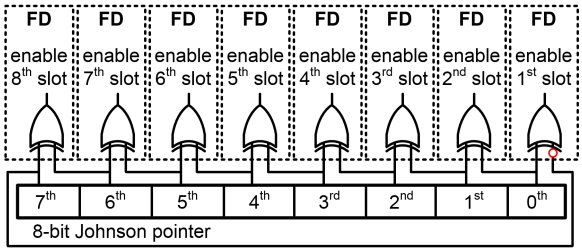

Fig. 8: Enable circuit for edge detector

Figure 9 shows a complete, eight entry flit detector FIFO. There is some additional detail over Figure 5 illustrating the use of the enable signal. (The additional feedback path here ensures the edge detector stays 'set' until explicitly reset.)

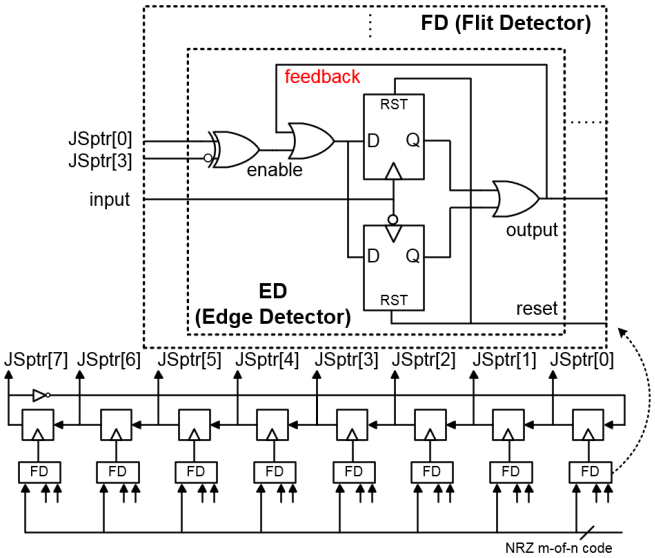

Fig. 9: Eight-bit parallel Johnson pointer

B. Flow control in the receiver channel

The FIFO must not overrun; if it is filled the cycle must be delayed. Instead of comparing two full-length pointers, a special flow control scheme is applied here. The FIFO is divided into two parts. The write pointer can write the first half without stalling. When the write pointer reaches the end of the first half, writing will stall if the read pointer falls behind more than half of the FIFO. When the read pointer reads the same half of the FIFO, the write pointer can proceed and write the other half of the FIFO. Therefore, the full-state assertions are fixed at the end of the two halves of the FIFO as shown in Figure 10.

The read pointer, although synchronous, is also represented as a Johnson counter for this reason. If the desired location is still full during the assertion, the acknowledgement is delayed.

Comparing the full length of two pointers can introduce significant delay to the communication response cycle and, unless implemented with some care, introduce undesirable glitches within the asynchronous circuit. Now the full assertion only requires one bit of the read pointer. The simplicity of

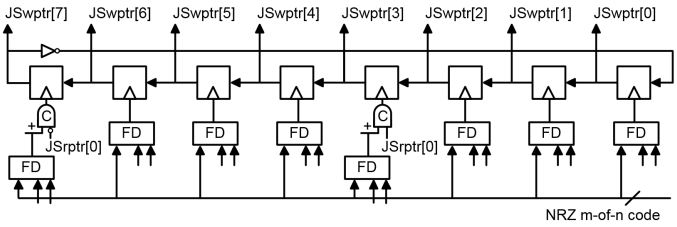

Fig. 10: Flow control units in Johnson write pointer

the comparison diminishes these problems. This flow control is a safe process without arbitration as any potential delay is present before the incoming flit and can only *terminate* before, during or after the flit's arrival. An asymmetric C-element [18] is implemented here to make the flow control unit more time insensitive. The 'full' state assertion is made at the rising edge of the flit detector unit.

C. Four-phase reset

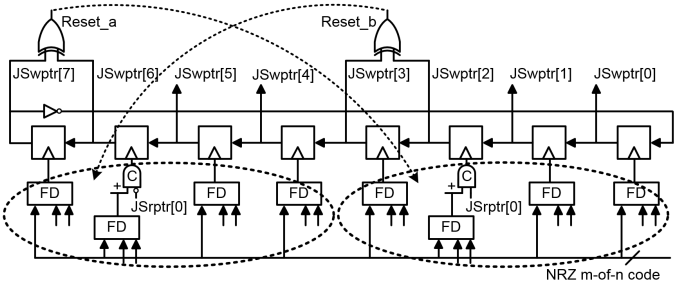

Fig. 11: Four-phase reset signals in receiver channel

Again, a Johnson pointer is used to generate the 4-phase reset signals. The flit detectors are divided into two parts which are reset by two 4-phase signals (Figure 11). The 4-phase reset signals are generated from the parity of two bits of the Johnson pointer. However the circuit can only be reset after the valid data is read. Therefore the 'full' assertion is moved one slot forward to control whether the circuit can be reset or not.

D. Synchronous domain

The final issue is the synchronisation of the input pointer to the clocked domain. This is done conventionally, with the latency 'concealed' by the FIFO action. The associated delays from a given flit detector to the synchronous circuit merely have to be less than the individual synchronisation time – a simple constraint to meet.

V. Complementary Interface Design

Section IV described the asynchronous-to-synchronous interface. Figure 12 shows the interface design for a transmitter based on similar principles. A set of four edge detectors is used to buffer the acknowledge signal and build a four-bit Johnson pointer to index eight memory locations; this is because it moves through eight discrete states. Again, the design aims to minimise the logic delays in the asynchronous cycle on the FPGA. The series of sequential actions is listed below:

1) The acknowledge signal arrives on the input wire; the active input signal triggers the edge detector and is translated into a level.
2) A similar flow control mechanism is implemented in the sender channel (Figure 13). The read pointer is incremented asynchronously and when an NRZ encoded flit is available at the head of the FIFO, it is output.

3) The NRZ code is output and the corresponding edge detectors are reset by two 4-phase reset signals generated from the Johnson read pointer (Figure 15).

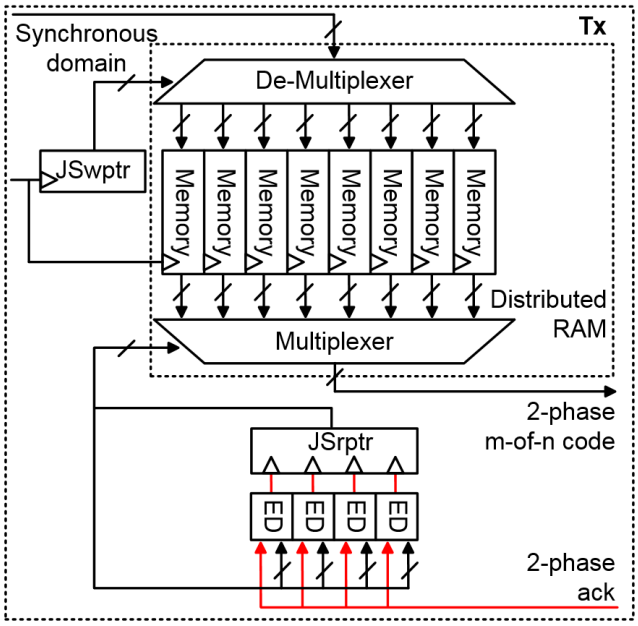

Fig. 12: Acknowledge detector FIFO

A. Flow control in the sender channel

A similar flow control mechanism is applied here, but the asynchronous circuits are in the reading domain. The synchronous write pointer stalls when the difference between the two pointers is equal to half of the FIFO. Here the *empty* indication to the read pointer is local to each location rather than the FIFO as a whole. This is derived by seeing that the corresponding (actually the *next*) bit in the write pointer is in the opposite state (shown in Figure 13). This flow control mechanism along with Johnson pointers can simplify the empty condition comparison in the reading domain.

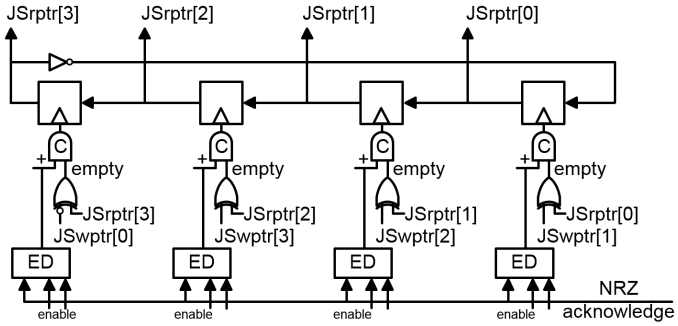

Fig. 13: Flow control units in transmitter channel

The asymmetric C-elements are necessary for the flow control unit here. If a normal AND gate is used, the asserted output will toggle the current bit of the read pointer; the changing bit feeds back to the flow control unit and causes the output of the flow control unit to be de-asserted. However, the related write pointer bit can change before the next assertion, and thus result in a wrong flip in the flow control unit.

Therefore, the asymmetric C-elements are used to avoid multiple flips in the flow control unit. The truth table and the gate level implementation of an Asymmetric C-element are shown in Figure 14. The output of the asymmetric C-element is only de-asserted when port A is de-asserted. The assertion only starts when port A goes to high. Generally, asymmetric C-elements are recommended for use in the asynchronous flow control unit, for safe operation and for time insensitivity.

A	B	Z
0	0	0
0	1	0
1	1	1
1	0	Z

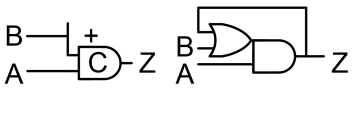

Fig. 14: Truth table and circuit of an asymmetric C-element

B. Four-phase reset

Resetting the circuit is again performed by using two four-phase reset signals generated from the Johnson pointer. The detection circuit is divided into two halves. The fist half is reset by the signal generated from the first two bits of the pointer. The second half is reset by the signal generated from the other two bits of the pointer. Comparing with the 4-phase reset signals in the receiver channel, the transmitter reset signals are generated without coordinating with the write pointer, because no data needs to be extracted before resetting the circuit.

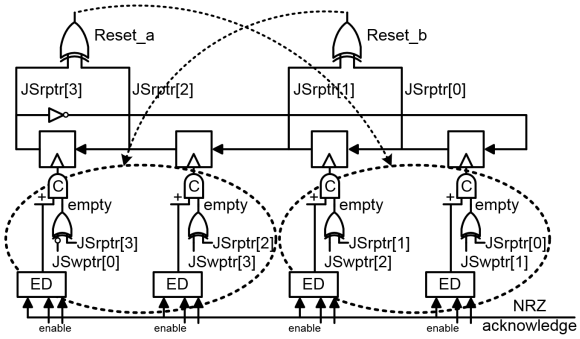

Fig. 15: Four-phase reset signals in transmitter channel

VI. Mapping and Placement on FPGA

The implementation target is a 45 nm Xilinx Spartan-6 FPGA which is used in the existing many-core system. The asynchronous circuits are mapped and placed as macros to minimise the delay on the FPGA side. The asynchronous components are designed by using and instantiating the FPGA logic elements [19], which prevents the synthesis tool from translating the behaviour model in an unexpected way and thus breaking the sequential sequences.

Mapping and placement can be done using relationally placed macros (RPMs), which provides a flexible way to design dedicated IP blocks on a Xilinx FPGA. RPMs allow users to do complete or partial mapping and placement in the macros. Precise mapping can be done by instantiating the FPGA logic elements in a hardware description language (HDL) and the placement can be specified in the user constraint file (UCF). Logic devices within the macro are planned based on relative coordinates and the whole macro can be moved around on the FPGA die. Therefore, RPMs are easier to manage and repeat than fixed hard macros.

For mapping asynchronous circuits, knowledge of the FPGA architecture is required. The following description is applied to the Xilinx Spartan-6 FPGA. Each configurable logic block (CLB) has multiple slices that contain smaller logic units.

Each slice has 4 six-input look-up tables (LUTs) and 8 D-type flip-flops. Some slices have more logic units such as carry logic and wide multiplexers. Figure 16 shows the connectivity between LUTs and flip-flops in the FPGA slice. Note every slice shares a common reset signal. The clock signal or its inverse is common to the whole slice.

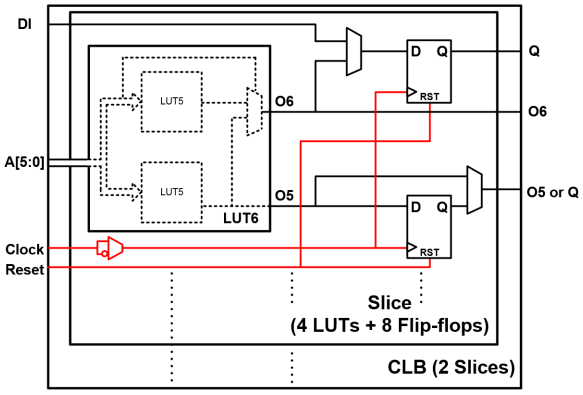

Fig. 16: The elements' connectivity in an FPGA slice

According to the manufacturer's datasheet [20] [21], the propagation delay of the LUT is 0.21 ns which is independent of the implemented combinational function. Therefore, logic delay can be minimised by mapping more combinational logic in a single LUT, which also leads to a more compact layout.

A. *Receiver floorplanning*

An eight-entry asynchronous interface FIFO is built in the receiver channel. Each bit of the link is clocking 8 edge detectors. Each edge detector is mapped in a pair of D-type flip-flops with two associated LUTs. The 8 D-type flip-flops triggered by the rising edge of the signal are mapped in two slices, because every slice shares a common clock signal (uninverted or inverted). The other 8 flip-flops used to detect the falling edge of the signal are mapped in another two slices. The enable logic which only requires 3 inputs can be mapped in the associated LUTs.

For the mapping of the completion detector, if one logic gate is mapped in one LUT, the longest path will traverse 4 LUTs and more delay will be introduced in the routing. Logic and routing delay can be reduced by mapping more combinational logic on a single LUT. Therefore, the optimised mapping is shown in Figure 17. The four C-elements and the output logic are mapped in one LUT with a feedback signal. Therefore, the longest path now only consists of two LUTs.

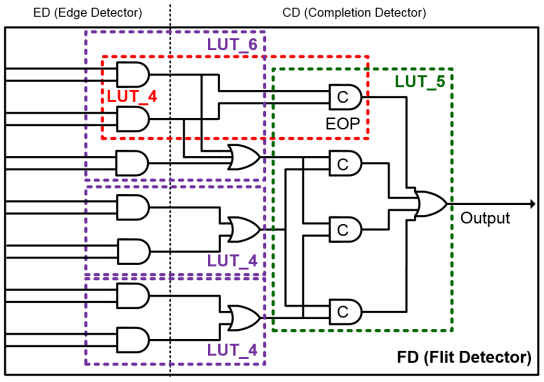

Fig. 17: Mapping of the completion detector

The eight-bit Johnson pointer is mapped and placed into 8 different slices driven by the completion signals from flit detectors, because each slice only has a single clock port. The acknowledge signal is generated from the parity of the 8-bit Johnson pointer. This 8-input exclusive-OR function is mapped in 2 LUTs and placed in the centre of the macro. Finally, two reset signals (two exclusive-OR gates) are mapped in two LUTs and also placed in the centre. The floorplanning is shown in Figure 18.

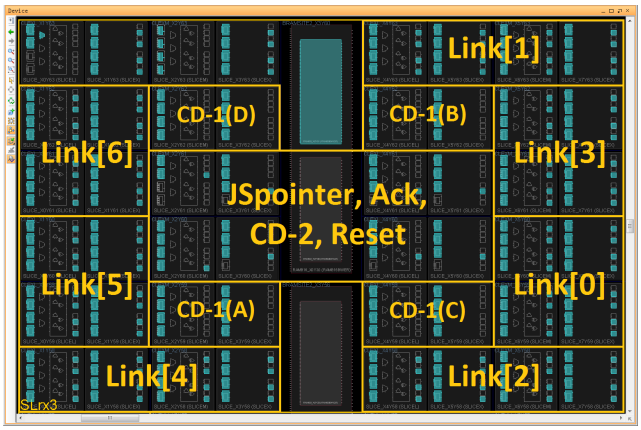

Fig. 18: Receiver layout

B. *Transmitter floorplanning*

A four-bit asynchronous Johnson pointer is implemented to output an eight-entry FIFO. An alternative implementation for the edge detector using LUTs is shown in Figure 19. For this implementation, the edge detector can be mapped in 3 LUTs, and thus 12 LUTs in total for 4 edge detectors. The flow control units are mapped in 4 LUTs. The four-bit Johnson pointer occupies 4 slices to map 4 D-type flip-flops and 1 associated LUT. Finally two reset signals are mapped in 2 spare LUTs in the transmitter macro.

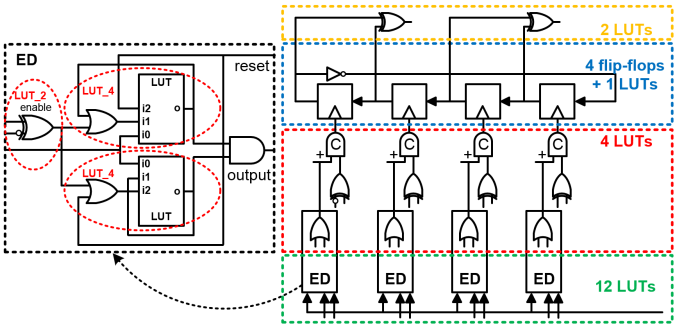

Fig. 19: Mapping of the asynchronous transmitter

The implementation using LUTs to detect edges eliminates the problem of single clock provision in a single FPGA slice. Therefore, the four edge detectors can be placed in the macro without occupying separate slices. The enable circuits can also be placed in the spare LUTs in the macro. Figure 20 shows the layout design of the transmitter channel in the PlanAhead tool.

VII. Results

Asynchronous communication throughput is limited by the time to finish the handshake protocol. The proposed interface design achieves higher throughput by reducing the delay on

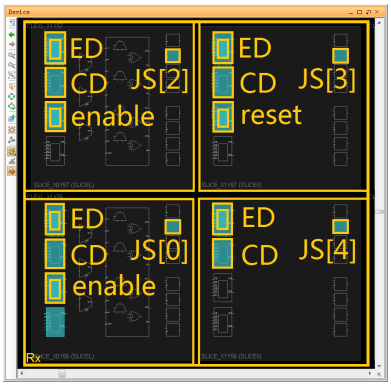

Fig. 20: Transmitter layout

the FPGA side and hiding the synchronisation latency behind the FIFO. The 2-of-7 asynchronous link runs at an average speed of 240 Mbps allowing 16 ns (giving 8 ns for each side) to finish a communication cycle. The throughput of the immediate synchronisation solution is about 150 Mbps costing more than 26 ns.

A. Elements constraints in the interface macros

Table II shows the number of FPGA logic elements mapped and placed in the interface macros. In the receiver channel, the edge detectors contain 112 flip-flops and associated 112 LUTs (7 inputs * 8 entries * a pair of flip-flops and LUTs) in the area of 28 FPGA slices. The completion detector is mapped in 32 LUTs (4 CDs * 8 entries) in the area of 8 FPGA slices. The Johnson pointer is mapped in 8 flip-flops and 1 LUT in the central area. The output logic of the completion detector (1 LUT * 8 entries) are also placed in this area. The centre area also contains 2 LUTs of the acknowledge signal and 2 LUTs of the reset signals.

In the transmitter channel, the Johnson pointer is placed in 4 different FPGA slices consisting of 4 flip-flops and 1 LUT. The other elements are mapped and placed in the same area. The edge detection contains 8 LUTs (a pair of LUTs * 4 entries) and another 4 LUTs for the enable circuits. The completion detection and the flow control units are mapped in 4 LUTs. The reset signals are mapped in 2 LUTs.

TABLE II: Asynchronous macro constraints

	Receiver						Sender			
	ED	CD1	CD2	JS	ACK	RST	ED	CD	JS	RST
FF	112	-	-	8	-	-	-	-	4	-
LUT	112	32	8	1	2	2	12	4	1	2
INST	224	32	8	9	2	2	12	4	5	2
SLICE	28	8	8				4			
Total	277 elements in 44 Slices						23 elements in 4 Slices			

B. Critical path delay analysis

All the logic delay can be calculated from the vendor's datasheet and the internal FPGA routing delay can be inspected in the Xilinx FPGA editor. From the Spartan-6 FPGA datasheet, the clock to output delay of a D-type flip-flop is about 0.45 ns. The input pad delay is about 1.2 ns. The output pad delay varies depending on the settings of slew rate and driving strength. A setting of the output pad with fast slew rate and 12 mA drive strength gives an output pad delay of about 1.71 ns.

For the receiver channel, the propagation delay through the edge detector may be larger than 0.45 ns because two transitions are independent and asynchronous. Transitions may arrive at different times. The total logic delay through the FPGA is listed in the following steps, totalling about 4.65 ns.

1) The 2-of-7 asynchronous inputs go through the FPGA input pads which have a delay about 1.2 ns.
2) When two transitions arrive, two D-type flip-flops are triggered where the clock to output delay for one transition is 0.45 ns.
3) The completion detection of the flit detector and the flow control unit (the full flag - asymmetric C-element) are mapped in two LUTs where the delay is 2×0.21 ns.
4) The Johnson write pointer is incremented where the clock to output delay is 0.45 ns.
5) The parity generator is mapped in two LUTs where the delay is 2×0.21 ns.
6) The acknowledge signal goes out of the FPGA output pads which have a delay of about 1.71 ns for the setting with fast slew rate and 12 mA drive strength.

For the Transmitter channel, the element delay is listed in the following sequence, which is 3.99 ns in total:

1) The acknowledge input signal goes through an FPGA input pad which has a delay of about 1.2 ns.
2) When two transitions arrive, the LUT based edge detectors are triggered where the LUT delay is about 0.21 ns.
3) The completion detection and flow control unit (the empty flag - asymmetric C element) are mapped in one LUT where the delay is also 0.21 ns.
4) The Johnson read pointer is incremented where the clock to output delay is 0.45 ns.
5) The read pointer is used to address the Distributed RAM to read the NRZ codes, where the address to output delay is 0.21 ns.
6) The NRZ encoded data goes through the FPGA output pad having a delay about 1.71 for the setting of fast slew rate and 12 mA drive strength.

The routing delay not inspected here can also impact the communication throughput significantly. The routing delay of an FPGA design can be greater than the logic delay. However, the routing delay can be better controlled in a dedicated layout design. The measured throughput result is presented in the next section.

C. Throughput result comparison

Table III and IV show the measured throughput results (Mbps) of the base design and the asynchronous design in different experimental setups. Both designs have been subject to a data integrity test, and then tested under different frequencies and different settings of the FPGA pads to show the delay impact. The default FPGA ouput pad setting is slow slew rate with 12 mA drive strength. The experimental setups have been chosen as follows: Quiet slew rate + 2 mA driving strength per output pad (5.92 ns, slowest); Slow + 6 mA per pad (3 ns, medium); and Fast + 12 mA per pad (1.71 ns, fastest).

TABLE III: Transmitter throughput comparison

	Transmitter					
	100Mhz			150Mhz		
	Base	Async	Impro	Base	Async	Impro
Quiet	100.01	222.95	2.23x	120.03	223.26	1.86x
Slow	100.01	274.74	2.75x	150.04	275.18	1.83x
Fast	100.01	296.82	2.97x	150.04	297.04	1.98x

TABLE IV: Receiver throughput comparison

	Receiver					
	100Mhz			150Mhz		
	Base	Async	Impro	Base	Async	Impro
Quiet	115.46	236.19	2.05x	139.88	236.28	1.69x
Slow	118.93	280.24	2.36x	165.40	280.37	1.70x
Fast	136.66	283.27	2.07x	165.40	299.02	1.81x

VIII. Conclusion

This paper presents a novel asynchronous interface FIFO design for interfacing delay insensitive inter-chip links with synchronous circuits; it is optimised for an FPGA implementation. As such it exploits D-type flip flops for elements such as edge/flit detectors for fast NRZ synchronisation. The throughput is increased by hiding the synchronisation delay behind a FIFO and minimising the delay in the asynchronous communication cycle. The critical asynchronous paths feature causal signal chains so are immune to layout delays and skew; timing sensitive paths are handled by dedicated layout design, chiefly relying on the synchronous part of the circuit as a reference. The interface has been designed as macros to aid automatic place-and-route at macro level, thus simplifying the implementation and improving portability.

The proposed Johnson-encoded asynchronous pointers provide a simple comparison circuit for the flow control signals which can also be applied in synchronous design. The pointer output is also used to generate enable signals and 4-phase reset signals for edge/flit detectors. The simplified enable signal has faster switching to next edge/flit detector. The result shows a significant improvement compared to the immediate synchronisation solution.

The exploitation of the state holding properties, such as the flit detectors, has allowed a considerable performance gain. At the same time the asynchronous pointers allow an implementation in an FPGA relieved of most timing considerations. This paper has dealt with the 2-of-7 NRZ protocol in an existing many-core system, but the proposed asynchronous FIFO can be applied to other m-of-n protocols. Furthermore, an extended ASIC version can be developed to improve the synchronisation throughput in an ASIC GALS system if the asynchronous networks employ a delay-insensitive m-of-n NRZ protocol for the interconnects.

Acknowledgement

The design and construction of the SpiNNaker machine was supported by EPSRC (the UK Engineering and Physical Sciences Research Council) under grants EP/D07908X/1 and EP/G015740/1, in collaboration with the universities of Southampton, Cambridge and Sheffield and with industry partners ARM Ltd, Silistix Ltd and Thales. Ongoing development of the software is supported by the EU ICT Flagship Human Brain Project (FP7-604102) and HBP SGA1 (H2020-720270, which also supports the first author's PhD studentship), in collaboration with many university and industry partners across the EU and beyond, and our own exploration of the capabilities of the machine is supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement 320689. SpiNNaker has been 15 years in conception and 10 years in construction, and many folk in Manchester and in our various collaborating groups around the world have contributed to get the project to its current state. We gratefully acknowledge all of these contributions.

References

[1] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Proceedings of the 38th Design Automation Conference*, 2001, pp. 684–689.

[2] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[3] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," *PhD Thesis, Stanford University*, 1984.

[4] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber, "Delay-Insensitive, Point-to-Point Interconnect using m-of-n Codes," in *9th IEEE International Symposium on Asynchronous Circuits and Systems*, 2003, pp. 132–140.

[5] J. You, Y. Xu, H. Han, and K. S. Stevens, "Performance Evaluation of Elastic GALS Interfaces and Network Fabric," *Electronic Notes in Theoretical Computer Science*, vol. 200, no. 1, pp. 17–32, 2008.

[6] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[7] Y. Shi, S. B. Furber, J. Garside, and L. A. Plana, "Fault Tolerant Delay Insensitive Inter-Chip Communication," in *15th IEEE International Symposium on Asynchronous Circuits and Systems*, 2009, pp. 77–84.

[8] D. J. Kinniment, A. Bystrov, and A. V. Yakovlev, "Synchronization Circuit Performance," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, pp. 202–209, 2002.

[9] D. J. Kinniment and D. Edwards, "Circuit technology in a large computer system," *Radio and Electronic Engineer*, vol. 43, no. 7, pp. 435–441, 1973.

[10] R. Dobkin, R. Ginosar, and C. P. Sotiriou, "Data Synchronization Issues in GALS SoCs," in *10th International Symposium on Asynchronous Circuits and Systems*, 2004, pp. 170–179.

[11] I. M. Panades and A. Greiner, "Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures," in *1st International Symposium on Networks-on-Chip*, 2007, pp. 83–94.

[12] C. E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," in *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA)*, 2002.

[13] E. Beigne and P. Vivet, "Design of On-chip and Off-chip Interfaces for a GALS NoC Architecture," in *12th IEEE International Symposium on Asynchronous Circuits and Systems*, 2006, pp. 183–192.

[14] Y. Thonnart, E. Beign, and P. Vivet, "Design and Implementation of a GALS Adapter for ANoC Based Architectures," in *15th IEEE Symposium on Asynchronous Circuits and Systems*, 2009, pp. 13–22.

[15] A. Yousefzadeh, L. A. Plana, S. Temple, T. Serrano-Gotarredona, S. B. Furber, and B. Linares-Barranco, "Fast Predictive Handshaking in Synchronous FPGAs for Fully Asynchronous Multisymbol Chip Links: Application to SpiNNaker 2-of-7 Links," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 8, pp. 763–767, 2016.

[16] F. Gray, *Pulse Code Communication*, US Patent Number 2,632,058, March 17, 1953.

[17] M. Zwolinski, *Digital System Design with SystemVerilog*. Prentice Hall Press, 2009.

[18] A. J. Martin, "Programming in VLSI: From communicating processes to delay-insensitive circuits," *Developments in Concurrency and Communication*, pp. 1–64, 1990.

[19] Xilinx, "Spartan-6 Libraries Guide for HDL Designs," UG615 (v14.7), Oct 02, 2013.

[20] Xilinx, "Spartan-6 FPGA Configurable Logic Block User Guide," UG384 (v1.1), Feb 23, 2010.

[21] Xilinx, "Spartan-6 FPGA Data Sheet :DC and Switching Characteristic," DS162 (v3.1.1), Jan 30, 2015.