SpinLink: An interconnection system for the SpiNNaker biologically inspired multi-computer

Kier Dugan, Jeff Reeve, and Andrew Brown
School of Electronics and Computer Science, University of Southampton, UK
{kjd1v07, jsr, adb}@ecs.soton.ac.uk

***Abstract*—SpiNNaker is a large-scale biologically-inspired multi-computer designed to model very heavily distributed problems, with the flagship application being the simulation of large neural networks. The project goal is to have one million processors included in a single machine, which consequently span many thousands of circuit boards. A computer of this scale imposes large communication requirements between these boards, and requires an extensible method of connecting to external equipment such as sensors, actuators and visualisation systems. This paper describes two systems that can address each of these problems.**

Firstly, SpinLink is a proposed method of connecting the SpiNNaker boards by using time-division multiplexing (TDM) to allow eight SpiNNaker links to run at maximum bandwidth between two boards. SpinLink will be deployed on Spartan-6 FPGAs and uses a locally generated clock that can be paused while the asynchronous links from SpiNNaker are sending data, thus ensuring a fast and glitch-free response. Secondly, SpiNNterceptor is a separate system, currently in the early stages of design, that will build upon SpinLink to address the important external I/O issues faced by SpiNNaker. Specifically, spare resources in the FPGAs will be used to implement the debugging and I/O interfacing features of SpiNNterceptor.

I. Introduction to the SpiNNaker Machine

Falling under the auspices of the overarching Biologically Inspired Massively Parallel Architectures (BIMPA) project, SpiNNaker is an experimental chip design that can achieve computing on very large scales [1]. An application specific asynchronous network fabric allows up to 2^{16} nodes to communicate with small source-addressed multicast packets [2]. A node contains, at present, 16 ARM processor cores that can be used for the application and additional processor reserved for monitoring and control of the node. A completely populated SpiNNaker system may hence contain up to 2^{20} (1,048,576) compute-cores available for a single application.

Communication has been a prime emphasis of the project because it is the most obvious potential bottleneck [3]. The solution is an attempt to mimic the connectivity of a mammalian brain on a much smaller scale. Brains are comprised of very large population of densely interconnected neurons. Each neuron is often considered a *simple* element and that complexity is derived from the interconnection of many neurons. Similar logic was applied in the design of SpiNNaker. Rather than grouping together complex processors with a large power budget, much simpler processors are used in a much vaster quantity. This technique allows SpiNNaker to achieve around 200TIPS (trillion instructions per second) across the whole computer while also minimising the power consumed.

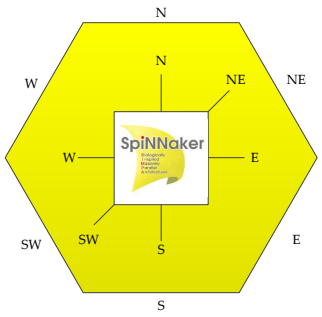

Fig. 1. A single SpiNNaker node showing both the compass direction labels and the hexagonal conceptual representation.

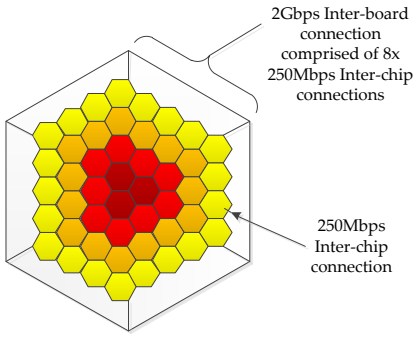

Fig. 2. A multi-node SpiNNaker machine (this case specifically being the thousand processor machine, i.e. the 103) comprised of many nodes tiled together to form a larger tile.

However, connecting this many processors into a single machine requires a sensible circuit board demarcation. A singular node has six 250Mbps ports that are used for communication with its direct neighbours. Traditionally these connections have been given compass references based on position around the chip but it can also be represented as a hexagon (Figure 1).

Following the hexagonal representation further allows the machine to be constructed by simply tiling many nodes together. Figure 2 shows how a SpiNNaker machine containing 816 total processors (768 of which are available for general purpose computation) can be assembled using this technique.

In the figure, nodes have been grouped into several rings

TABLE I
SPINNAKER MILESTONE MACHINE DESIGNATIONS AND SIZES († DENOTES A SPECULATIVE MILESTONE THAT HAS YET TO BE DETERMINED).

Designation	Number of Processors	Number of Nodes
101	17	1
102	68	4
103	816	48
104†	9762	576
105†	97620	5760
106†	976200	57600

to help demonstrate how the machine must be constructed. A single node can be used in isolation by connecting the diametrically opposed edges together (i.e. N→S, NE→SW, and E→W) but rings cannot be formed around a single tile because the number of edges increases uniformly in each direction and hence cannot be connected correctly. By instead using three tiled nodes as the base element (the darkest nodes in Figure 2) the number of edges still increases uniformly in each direction but the mapping between edges is preserved.

Each group of SpiNNaker nodes arranged in this manner can therefore also be treated as a hexagonal tile and the technique continues to apply. Obviously as the number of nodes in a group increases so does the bandwidth required at the boundary. For this reason 48 nodes will constitute a *board* because there are eight 250Mbps node edges that can be multiplexed through a high-bandwidth connection. This is an engineering decision influenced by available hardware and a convenient address space for the multiplexing.

A. Objectives

The final SpiNNaker machine will include over a million processors but this cannot be achieved in a single step. Instead, several milestone machines will be constructed throughout the project each stage building on the previous by an order of magnitude. Each milestone is designated by rounding the number of processors it contains to the nearest power-of-ten as shown in Table I. The 102 machine is the current testing platform being used at the member universities while the 103 design is being completed.

II. BOARD-LEVEL INTERCONNECTION

Tiling many SpiNNaker boards together to build larger machines is a simple and scalable method of construction but it does present a communication issue. Inter-node links use a self-timed asynchronous bundled protocol that operates at 250Mbps. Each symbol is two-phase 2-of-7 encoded which means that bit transitions represent a logic '1' and that two transitions must occur to represent a symbol. There are 21 valid transitions in 2-of-7 encoding but only 17 are used in SpiNNaker; 16 convey data (i.e. 4 bits per symbol) and 1 is used as the end of packet (EOP) symbol [4].

Figure 3 shows the physical layer connections used by SpiNNaker. Each symbol presented on a data connection

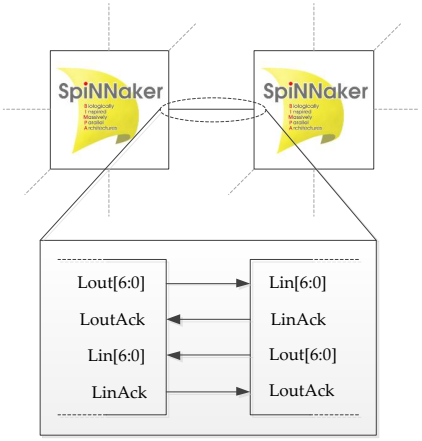

Fig. 3. Electrical connections used by the SpiNNaker inter-node connections.

(`Lin`/`Lout`) must be acknowledged by toggling the acknowledgement line (`LinAck`/`LoutAck`) before another symbol can be produced. This behaviour is not dependent on the clocks in either node which removes the need for clock recovery. However, it requires the two nodes to be located physically close together on the circuit board, otherwise the propagation delay of the tracks introduces enough delay to prevent the maximum bandwidth from ever being achieved.

Each bidirectional link comprises 16 physical wires meaning that each node has 96 pins purely for the inter-node communication ports. Furthermore each edge of 48-node board of Figure 2 contains 8 of these links which leads to a total requirement of 128 pins for board-to-board communication. This is impractical for two reasons: (a) the length of the wires may incur too great a propagation delay (especially if the connection is between two boards located in different cabinets); and (b) the cost required for specialised connectors and large cables would be too great.

A practical solution to this problem is to multiplex each set of inter-node links at each edge within a much faster serial link between the boards. Three Xilinx Spartan-6 LX45T FPGAs will be fitted to each 103 board for this purpose. The LXT series include a number of high-speed serial transceivers (called GTP modules) that provide up to 3.125Gbps of bandwidth. Figure 4 shows the conceptual layout of the current 103 board design.

The LX45T part was primarily chosen because it has a suitable number of IO pins to service two edges of the 48-node board hexagon and it has GTP transceivers capable of providing the required bandwidth. A serendipitous consequence of this design decision is that the part has four GTP modules where only two are required; hence the *spare* connections can be used for high-speed IO to devices that can provide simulations with real-time and real-world data, thus allowing SpiNNaker to be used for robotics applications.

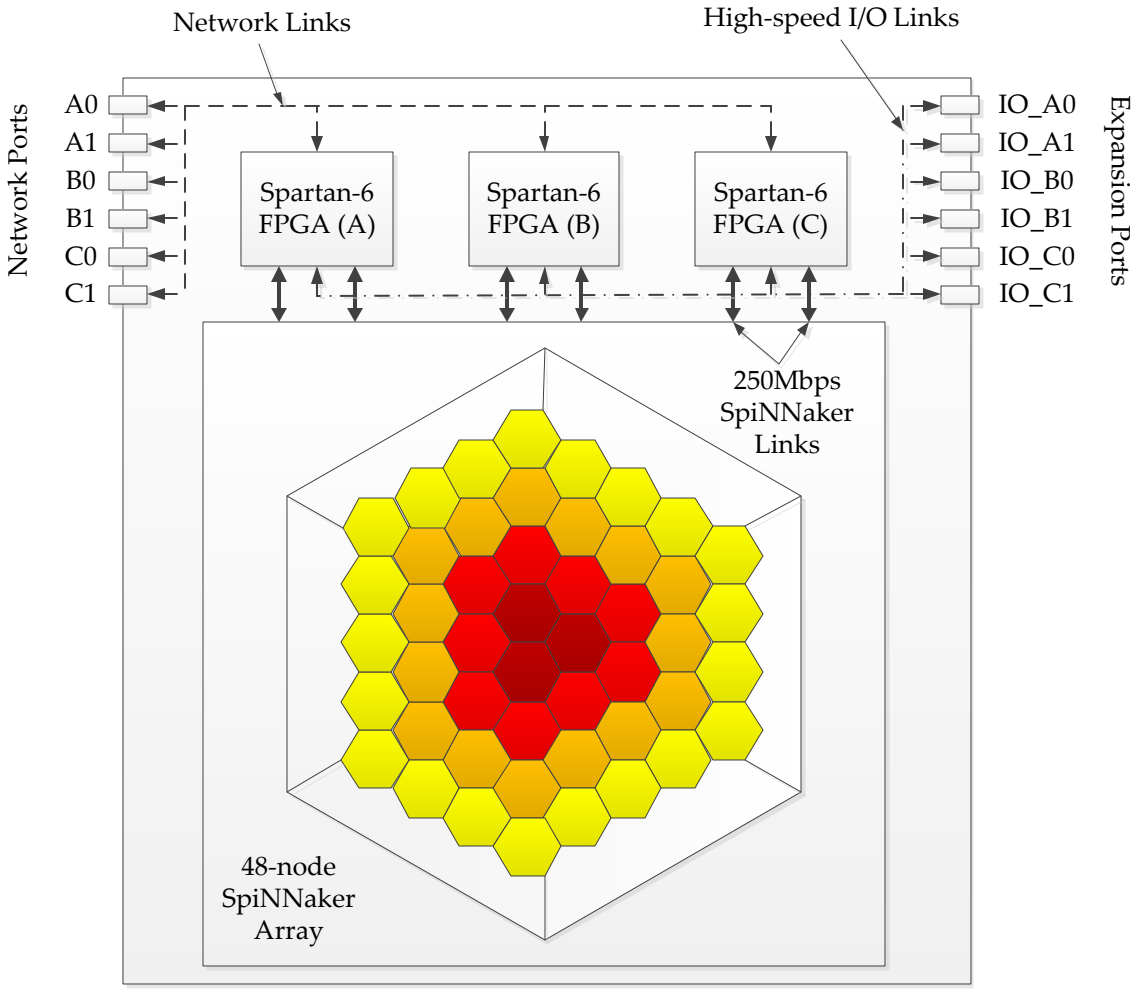

Fig. 4. Proposed layout of the 103 SpiNNaker milestone machine.

A. *Physical Layer*

The Spartan-6 FPGA includes a silicon IP module that provides a bidirectional high-speed serial physical layer. A single module can provide a channel with a bandwidth of up to 3.125Gbps which is sufficient to allow eight 250Mbps SpiNNaker links to be time-division multiplexed (TDM). There is also enough headroom to support some simple control mechanisms as well as 8b10b encoding to ensure accurate and reliable clock recovery.

SpinLink uses a 20-byte *frame* (shown in Figure 5) which contains two bytes worth of data for each time-multiplexed link and has a timeslot of 64ns. Hence 16 bits of data are transmitted per link every 64ns leading to a bandwidth allocation of 250Mbps per logical link. A ninth logical link exists that can be used for out-of-band communication between the FPGAs but it is only a single byte wide and hence only has a bandwidth allocation of 125Mbps.

Each bit in the flow control byte simply maps onto the equivalent logical link in the remote transmitter (i.e. bit 0 corresponds to logical link 0) and acts as an enable signal. Normally every bit in this byte is set thus enabling every

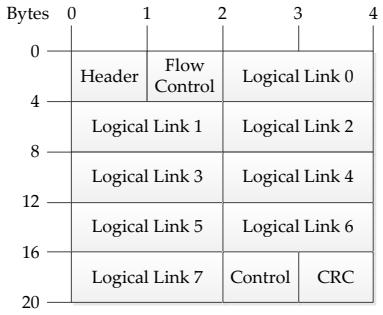

Fig. 5. Structure of the frame used for TDM transmission.

link in the remote transmitter; however, if a local receive buffer approaches its upper limit (set by a register) then it can request that the local transmitter clear the appropriate bit in the flow control byte, which will disable the corresponding remote logical link.

Finally, the header byte is used to differentiate the main data frame as shown in Figure 5 from control frames that are used to re-synchronise the FPGAs at each end of the connection. The Spartan-6 has over a hundred large block RAMs which can be used as ring buffers. Each frame is pushed into this ring buffer before it is transmitted. In the event of a de-synchronisation, each FPGA transmits the last known remote read-pointer and transmission continues from there. This process does incur some delay but it prevents any data from being lost.

B. Packet Checking

Symbols are pushed into an *inspection buffer* as they are received from SpiNNaker which allows the full packet to be built up in the FPGA before it is passed onto the high-speed serial subsystem described in the previous section. The primary use case for this design is that the parity can be checked in the inspection buffer and the packet can be dropped here if the check fails. Discarding packets that fail the parity check is not detrimental to SpiNNaker because this behaviour occurs internally [5].

C. Synchronisation with SpiNNaker

The interface to SpiNNaker presents an interesting set of problems because the protocol is asynchronous and self-timed thus containing no clock or time information. Each symbol can therefore be considered an event that triggers the decoding process. The event is generated by a *completion detection* circuit that asserts a signal when a valid symbol has been received. This interface is further complicated by the GTP modules requiring a consistent, stable and glitch free clock signal.

Initially this problem was addressed with a dual-clock FIFO buffer where the write port clock was driven by the output of the completion detection circuit and the read port clock came from the GTP module clock source. This did not scale to the required number of SpiNNaker links (sixteen) for two reasons: (a) the FIFO buffers require several clock cycles after reset signal has been asserted to successfully initialise, and (b) the clock inputs *must* be driven by global clock buffers, and these are a finite resource inside the FPGA.

A modified version of the circuit proposed by Moore *et al.* [6] has been used to generate clock signals that may be paused by the asynchronous receive logic, as shown in Figure 6. The original circuit was designed to be used with a four-phase system using return-to-zero (RTZ) encoding. It has been extended to function with 2-phase signals [7], however this extension is not suitable here as it would introduce another layer of decoding. By adding a specific acknowledgement generation flip-flop and a non-return-to-zero (NRZ) to RTZ decoder (shown inside the dashed boxes of Figure 6), it is possible to generate the acknowledgement signal within a single clock cycle. This is essential here because the acknowledgement must be generated no more than 8ns after a symbol has been received to ensure that the link can operate at maximum bandwidth. A similar circuit, shown in Figure 7, has been used to read symbols from an output FIFO and transmit them into SpiNNaker.

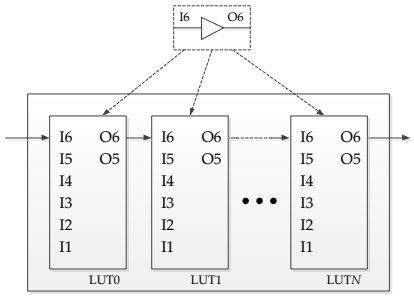

Fig. 8. Delay line implementation at block diagram level.

These circuits do not remove the requirement for a large number of global clock buffers but they do provide a way to circumvent it. Both figures show a single mutual exclusion element (MUTEX) to arbitrate between the clock feedback and the asynchronous request signals. Each MUTEX may be swapped for an N-input concurrent arbiter which allows a single clock generator to be shared by N links. This solution is fully compatible with SpiNNaker because each link is truly asynchronous.

D. Clock generation with FPGA elements

The delay line must be accurately tuned for these circuits to work correctly. A clock with a period of 8ns will provide a window that will guarantee a response within 8ns if the event from SpiNNaker arrives in the positive *level* of the clock. This cannot be guaranteed when the clock is first started or even for the first symbol transmitted, but SpiNNaker is guaranteed to synchronise with the proposed circuits after the first acknowledgement has been generated.

Tuning the period of the clock generators was achieved using a combination of Xilinx FPGA Editor and Xilinx PlanAhead (both contained within the Xilinx ISE Design Suite), and a parameterised Verilog module that instantiated a set of *identity* look-up tables (LUTs) as shown in Figure 8. Xilinx PlanAhead was used to deliberately position the LUTs to introduce a large propagation delay, then the design was resynthesized using the Xilinx XST tool-chain and the delays estimated by Xilinx FPGA Editor. The estimations were valid for a half-cycle of the generated clock and were hence doubled to estimate the period. Three tuned delay-lines were synthesised into the clock generator designs discussed previous and then instantiated multiple times in the target Spartan-6 FPGA. Overall eight type A and ten of each type B and C were included in the test design. Table II lists the delay

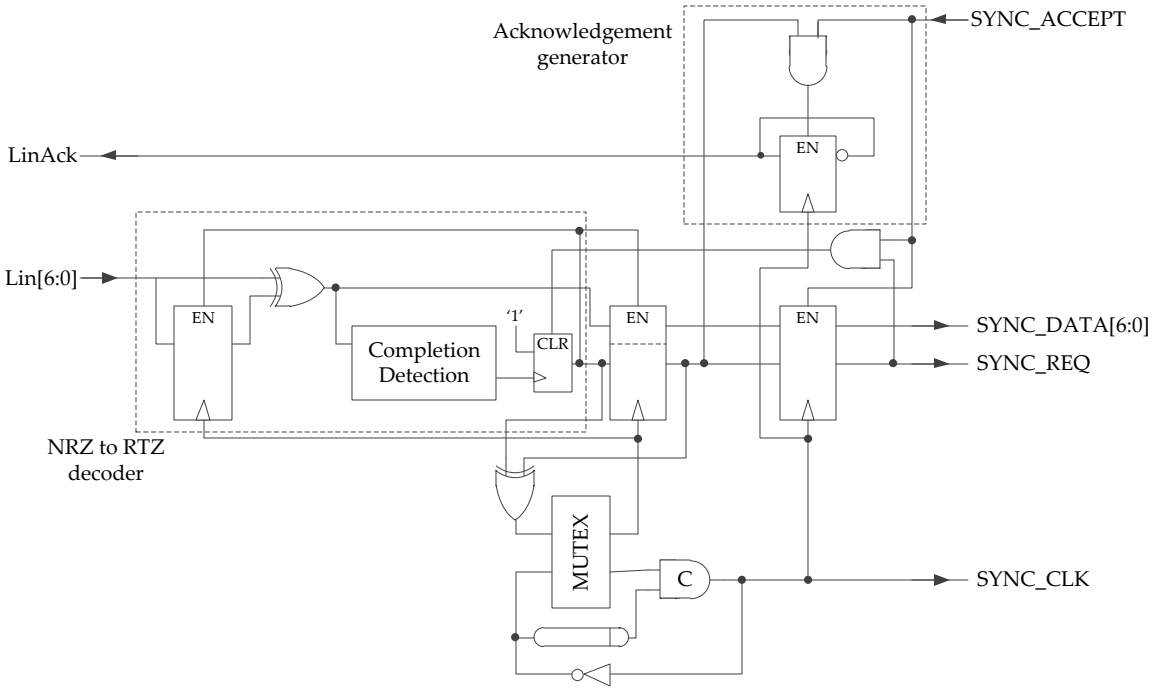

Fig. 6. Input SpiNNaker synchronisation circuit and clock generator.

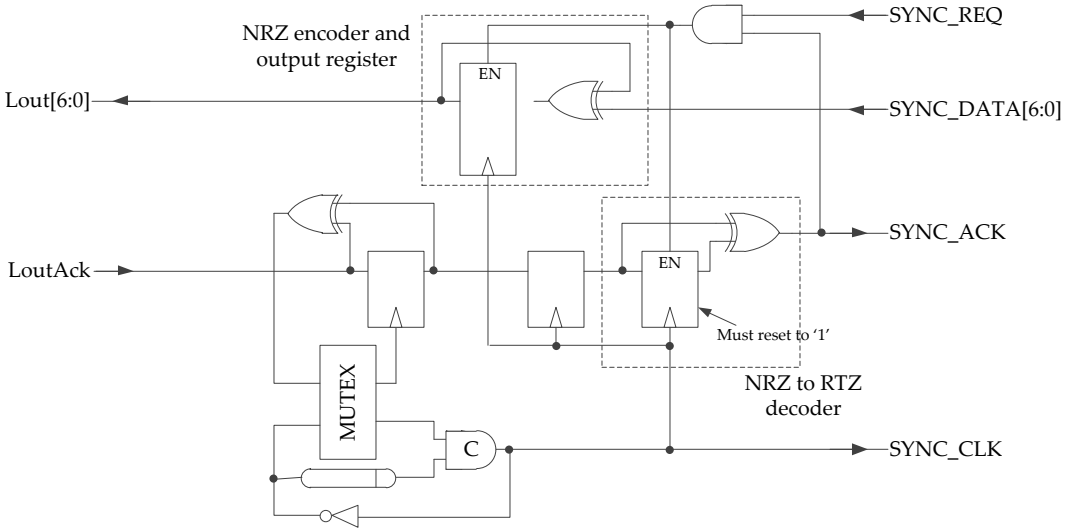

Fig. 7. Output SpiNNaker interface circuit and clock generator.

measurements obtained experimentally and those predicted by Xilinx FPGA Editor for ease of comparison.

The *worst relative error* given in the table was calculated using Equation 1, where $\overline{T}$, T_{min}, and T_{max} are the average, minimum, and maximum periods measured respectively.

$$WRE = \max\left(\frac{\overline{T} - T_{min}}{\overline{T}}, \frac{T_{max} - \overline{T}}{\overline{T}}\right) \quad (1)$$

This technique shows that even the most extreme outliers

TABLE II
AVERAGE DELAY-LINE MEASUREMENTS TAKEN FROM THE INSTANTIATION OF THREE TYPES OF TUNED CLOCK GENERATORS COMPARED TO THE PREDICTED RESULTS.

Type	Estimated Delay (ns)	Average Measured Delay (ns)	Worst Relative Error (%)
A	2.91	2.92	0.96
B	3.91	3.58	1.56
C	4.67	4.07	1.41

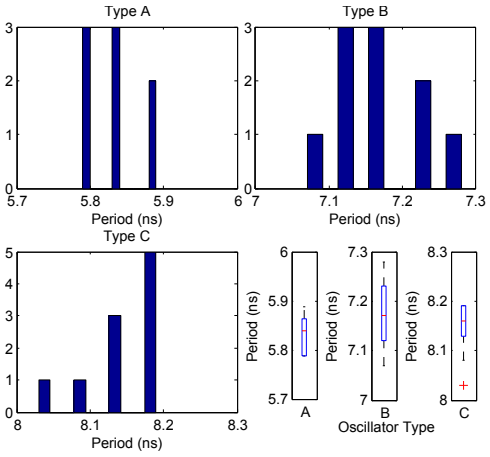

Fig. 9. Histograms of the measured periods of multiple instantiations of the three clock-generator designs. The box plots show that the spread of each period is minimal with regards to the mean.

were within at most 1.6% of the mean for each type of clock generator, which can also been seen in the various plots of Figure 9.

This implies that the oscillators can be reliably instantiated as part of a design for the Spartan-6 FPGA despite the clear differences between the estimated and measured delays. Delay-lines can be manually optimised post-synthesis based on experimental measurements to reduce this difference. Another possible solution to this problem is to replace the fixed delay-line with a dynamically recalibrating design as proposed by Taylor *et al.* [8].

III. SPINNTERCEPTOR

While SpinLink can address the issue of connecting multiple boards together to form a functioning machine, it does not provide a method of allowing SpiNNaker to interact with other equipment. The current testing and development boards, and those being developed, use a single 100Mbps Ethernet connection to allow SpiNNaker to be controlled from a conventional PC. This interface is adequate for issuing commands and can support small real-time simulations with data, but it simply cannot scale as the machine grows.

SpiNNterceptor is currently being developed to address this problem by combining two similar concepts: debugging and high-speed I/O. Firstly, the *debugging* aspect comes from an existing concept design that behaves as a packet-sniffer between two Xilinx Virtex-5 FPGAs containing a much older

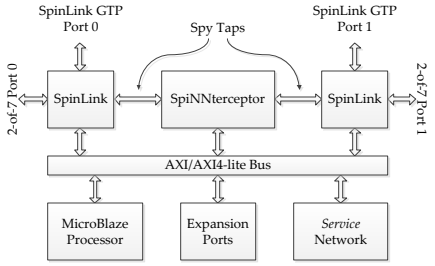

Fig. 10. Proposed high-level plan for the implementation of SpinLink and SpiNNterceptor.

SpinLink implementation. SpinLink packet data is streamed through the *SpinLink-Spy* (installed in a host PC as a single-lane PCI-Express device) and each packet is compared to a trigger. All packets received after a match has been detected are stored in a sample RAM that can be inspected by software running on the PC.

Secondly, a *high-speed I/O* interface is being design to make use of the FPGA resources that are not being used by SpinLink. These include two 3.125Gbps GTP transceivers and a large (but not accurately determined at the present time) number of the logic resources. A protocol will be developed that will allow various real-time and real-world peripherals to be connected to SpiNNaker. Auditory sensors and a Silicon Retina [9] are two types of input device that have been identified as candidates for this interface.

A. Proposed Implementation

SpiNNterceptor can be thought of as an extension of the inspection buffer explained previously. Two registers can be added to this part of the design to provide comparison and mask values that can be used to trigger the packet-sniffing interface. Additionally, extra logic can be included to detect packets that are intended for SpiNNterceptor instead of standard SpinLink traffic. We intend to delegate one bit of the reserved address range of multicast packets for this purpose so the logic should be simple. If a SpiNNterceptor packet is detected then it will be removed from the stream as it will serve no further purpose and forwarded onto the SpiNNterceptor module.

We intend to connect the FPGAs together as a ring so that output packets can be received by any FPGA on a board and then routed to the required expansion port outside of the SpiNNaker network so that unnecessary congestion is not introduced. Devices connected to the expansion ports will communicate using a high-speed serial protocol that is yet to be designed. A standard protocol will be chosen, if possible, to maximise the scope of devices that can be used.

Figure 10 shows the current design plan for the FPGAs of Figure 4. Here the *service network* will be ring topology connecting the FPGAs together and the expansion ports will be the GTP modules unused by SpinLink. The MicroBlaze may provide a flexible and convenient approach to implementing

the protocols required to support the inter-FPGA network, communication with external peripherals, and advanced debugging and packet sniffing behaviour. Further investigation is required to determine the feasibility of this design however.

B. Software Implications

Simulating large-scale neural networks is the flagship application of SpiNNaker but any problem that can be sufficiently discretised can be written as a SpiNNaker application. Applications are currently described by a problem graph that represents the connectivity between interrupt handlers that implement the desired functionality. This implies that the problem graph should contain information about the connectivity of the program to external I/O devices.

The notion of source and monitor nodes will be included into the problem graph and allowing them to be bound to specific expansion ports on the SpiNNaker board (shown in Figure 4). Input data can therefore be routed into a specific problem circuit using the existing SpiNNaker routing topology but it will originate from one of the Spartan-6 FPGAs. Outputs can be realised simply emitting a packet in a single direction because it will be guaranteed to reach a board boundary where it will be read into one of the FPGAs. We intend to delegate one bit of the multi-cast packet address range to indicate that the packet is intended for SpiNNterceptor and not transmitted with SpinLink. The inspection buffer can be used to detect the value of this bit and hence remove the packet from the stream and instead send it to the SpiNNterceptor module.

IV. Concluding Remarks

SpiNNterceptor will be an important enabling technology for SpiNNaker because it addresses two distinct problems. Streaming real-world data into SpiNNaker in real-time allows for sophisticated simulations to be executed on SpiNNaker, but data can also be streamed *out* in real-time into post-processing and novel visualisation systems. We hope to provide a very simple and extensible interface that will allow commodity computing equipment to be easily connected to SpiNNaker to serve this purpose. Further work is required to allow SpiNNterceptor to be conveniently used in the compilation and runtime stages of an application, and to integrate it into SpinLink.

Most of the design work has now been completed on SpinLink and has been verified by simulation and simple tests. Connecting SpinLink to a physical SpiNNaker test board is the next logical step in this process. Following the production of the 103 test board, SpinLink can be verified *in situ* and will allow the 103 board to function as a complete and standalone SpiNNaker machine.

References

[1] SpiNNaker home page. Unpublished. University of Manchester. [Online]. Available: http://apt.cs.man.ac.uk/projects/SpiNNaker/

[2] L. A. Plana, S. B. Furber, S. Temple, M. Khan, Y. Shi, J. Wu, and S. Yang, "A GALS Infrastructure for a Massively Parallel Multiprocessor," *IEEE Design & Test of Computers*, vol. 24, no. 5, pp. 454–463, Sep. 2007.

[3] S. Furber and A. Brown, "Biologically-Inspired Massively-Parallel Architectures - Computing Beyond a Million Processors," in *2009 Ninth International Conference on Application of Concurrency to System Design*. IEEE, Jul. 2009, pp. 3–12.

[4] J. Wu and S. Furber, "Delay Insensitive Chip-to-Chip Interconnect Using Incomplete 2-of-7 NRZ Data Encoding," in *Proceedings of the 18th UK Asynchronous Forum*. Newcastle upon Tyne, UK: University of Newcastle, 2006, pp. 16–19.

[5] J. Wu, S. Furber, and J. Garside, "A Programmable Adaptive Router for a GALS Parallel System," in *2009 15th IEEE Symposium on Asynchronous Circuits and Systems*. IEEE, May 2009, pp. 23–31.

[6] S. Moore, G. Taylor, R. Mullins, and P. Robinson, "Point to point GALS interconnect," in *Proceedings Eighth International Symposium on Asynchronous Circuits and Systems*. IEEE Comput. Soc, 2002, pp. 69–75.

[7] J. Pontes, R. Soares, E. Carvalho, F. Moraes, and N. Calazans, "SCAFFI: An intrachip FPGA asynchronous interface based on hard macros," in *2007 25th International Conference on Computer Design*. IEEE, Oct. 2007, pp. 541–546.

[8] G. Taylor, S. Moore, S. Wilcox, and P. Robinson, "An on-chip dynamically recalibrated delay line for embedded self-timed systems," in *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000) (Cat. No. PR00586)*. IEEE Comput. Soc, 2000, pp. 45–51.

[9] T. Delbruck, "Frame-free dynamic digital vision," in *Proceedings of Intl. Symp. on Secure-Life Electronics, Advanced Electronics for Quality Life and Society*, 2008, pp. 21–26.