Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Fynn Beuttenmüller

born in Bremervörde (Germany)

Heidelberg, 2014

Interfacing a Neuronal Accelerator to a High Performance Computing System

This Bachelor Thesis has been carried out by Fynn Beuttenmüller at the Institute of Computer Engineering in Mannheim

Supervisor:	Christian Leibig
Refery:	Prof. Ulrich Brüning
Co-Refery:	Prof. Karlheinz Meier

The work at hand describes the connection of configurable neuromorphic computing substrates applying the means of EXTOLL technology. Within the subproject Neuromorphic Computing—as part of the overall Human Brain Project—research is performed in a hardware system that emulates accelerated neuronal networks. The high grade in connectivity and activity of neuronal models, as well as the speed-up factor of 10^3 to 10^4 in comparison to biological real time, are highly demanding with regard to bandwidth and latency for such a system. At first it is checked within this thesis work whether the EXTOLL technology is suitable for integration to the neuronal accelerator. A general survey on the accelerated neuromorphic computing system is performed and the system requirements towards an external data network are determined. Furthermore the EXTOLL technology with ist most current network board Tourmalet and its interface is presented. Latency and bandwidth in relevant network topologies are calculated with regard to neuronal networks and thereby the suitability of EXTOLL asserted. An EXTOLL interface module was designed and implemented, which is able to communicate neuronal pulse event information via an EXTOLL network. This interface connection has potential to be expanded exchanging configuration and system data in addition to the pulse event information via EXTOLL.

Die vorliegende Arbeit beschreibt die Verbindung neuromorpher Recheneinheiten mithilfe der EXTOLL Technologie. Im Rahmen der Subprojekts Neuromorphic Computing des Human Brain Projekts wird an einem Hardwaresystem zur beschleunigten Emulation spikender neuronaler Netzwerke geforscht. Der hohe Grad an Konnektivität und Aktivität neuronaler Modelle und der Beschleunigungsfaktor von 10^3 bis 10^4 ,verglichen zu biologischer Echtzeit, stellen hohe Anforderungen in Bezug auf Bandbreite und Latenzzeit an ein solches System. Diese Arbeit prüft zunächst ob sich die EXTOLL Technologie zur Integrierung in den neuronalen Beschleuniger eignet. Dafür wird ein Überblick über das beschleunigte neuromorphe Rechensystem gegeben und dessen Anforderungen an ein externes Datennetzwerk bestimmt. Weiterhin wird die EXTOLL Technologie in Form der aktuellen Netzwerkkarte Tourmalet vorgestellt und deren Schnittstelle spezifiziert. Mit Bezug auf neuronale Netzwerkmodelle werden Latenz und Bandbreite in entsprechenden Netzwerktopologien berechnet und so die Eignung von EXTOLL festgestellt. Daraufhin wird ein EXTOLL-Verbindungs-Modul entworfen und implementiert, welches neuronale Pulsinformation über ein EXTOLL Netzwerk kommunizieren kann. Diese Verbindung hat das Potential weiterentwickelt zu werden, um neben Pulsinformationen auch Konfigurationsdaten und Systeminformationen über EXTOLL auszutauschen.

Contents

1. Introduction

The task at hand is to interface an accelerated neuromorphic system—the core of the Human Brain Project's (HBP) Neuromorphic Physical Model (NM–PM) platform—to a high performance computing system using the EXTOLL network technology. First the suitability of EXTOLL for this specific application has to be evaluated. In addition to the functional requirements of the neuronal accelerator two key issues are determined: bandwidth and latency. Demands on bandwidth are determined by assessing the amount of neuronal traffic that has to be emulated in such a neuromorphic computing system. The acceptable limit of latency is deduced from neuronal network models and the performance of the neuronal accelerator's core logic.
On EXTOLL side the capabilities of Tourmalet, EXTOLL's most current ASIC, are discussed and the most current EXTOLL communication protocol and the network port interface are specified. With these results the potential of an EXTOLL Tourmalet network is gauged.
Using the EXTOLL technology within the Neuromorphic Physical Model (NM–PM) platform requires compatibility to the EXTOLL network protocol within the control logic of the neural accelerator. This neuromorphic system is wafer-based. One wafer contains 384 High-Input Count Analog Neuronal Network chips (HICANNs) that each implement up to 512 neurons and 11 k synapses. In the current state of development the runtime control of these HICANNs is managed by FPGAs in groups of eight HICANNs. This setup of on-wafer HICANNs, FPGAs and some additional units is referred to as the Neuromorphic Computing System (NCS) and will be elucidated with more detail.
To prepare integration of EXTOLL technology into the NCS the implementation in the mentioned FPGAs is analyzed and a suitable interface for an external network is found and specified. In a next step an EXTOLL interface module is planned to fit the FPGA design. This interface module is implemented enabling the transmission of pulse events via an EXTOLL network with the potential of extending the interface to support the distribution of configuration data.

2. The Neuromorphic Computing System

Within the Sub-Project 9 – Neuromorphic Computing (SP9) of the Human Brain Project (HBP) two platforms are under construction: The Neuromorphic Multi-Core (NM–MC) and the Neuromorphic Physical Model (NM–PM). Both approaches aim at simulating brain activity with custom hardware, which is often referred to as brain emulation. The NM–MC uses a massively parallel system of ARM architecture processors in the Spiking Neural Network Architecture (SpiNNaker) [10]. This system operates in real-time, which also allows for research applications in robotic platforms [10]. The Neuromorphic Computing System (NCS) referred to in this thesis is the core of the NM–PM and will initially comprise four million accelerated analog neurons and almost one billion synapses on 20 eight-inch silicon wafers manufactured in 180 nm Complementary Metal-Oxide-Semiconductor (CMOS) technology [24]. Single wafer setups are currently being tested (Figure 2.1). It is envisioned to apply wafers which are pro-

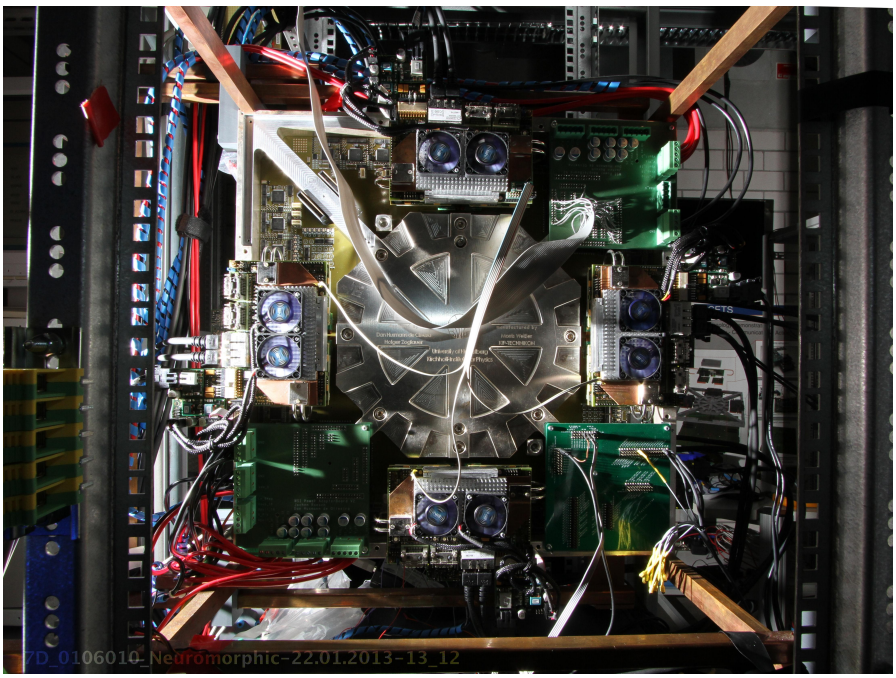


Figure 2.1.: One operating NM–PM wafer. Photograph from *Brain-inspired multiscale computation in neuromorphic hybrid systems (BrainScaleS) website* [2].

duced with a 65 nm process in the long run, allowing for a wider synapse address, an integrated Plasticity Processing Unit (PPU) for a more advanced synapse plasticity model [9] and on-wafer ADCs for synchronous analog read-out [6]. The refined manufacturing process will also allow

to operate the wafers at lower voltage and thus further lowering energy consumption. The NM–PM platform provides access to the NCS throughout the HBP, enabling researchers in other subject areas to conduct their neuronal network experiments. In order to support as many experimental setups as possible the network topology of the NCS, as well as biological and electronic cell parameters are configurable [24, 14]. The operating NCS has a speedup factor of 10^3 to 10^4 compared to biological real-time [24] and is able to emulate point-like neurons, and in later versions it will be able to emulate simple multi-compartment neurons [18].

2.1. The Wafer Module

Each of the above mentioned wafers is placed on a 19-inch Printed Circuit Board (PCB) containing additional electronics for power supply, configuration, communication and analog read-out. Every wafer is used as a whole, i.e. the complete wafer as manufactured, undivided, and it partitioned into 48 reticles. The power control of these reticles is realized individually by a Raspberry Pi single-board computer which coordinates the main power supply PCBs ("PowerIt" board) and special monitoring and control PCBs ("Cure" boards) [24]. This allows to shut down individual reticles for an experiment (e.g., lower amount of required reticles within an experiment) or more importantly due to inoperability (e.g., broken parts, yield).
The communication of reticles is managed by a Kintex-7 XC7K160T FPGA from Xilinx which enables communication of reticles on any wafers, including the same wafer. Post-processing of the wafers makes direct on-wafer communication possible, too.
Figure 2.2 illustrates the relation of HICANNs, reticles and FPGA Communication PCBs (FCPs). Every reticle contains eight HICANNs that each implement the adaptive exponential integrate-

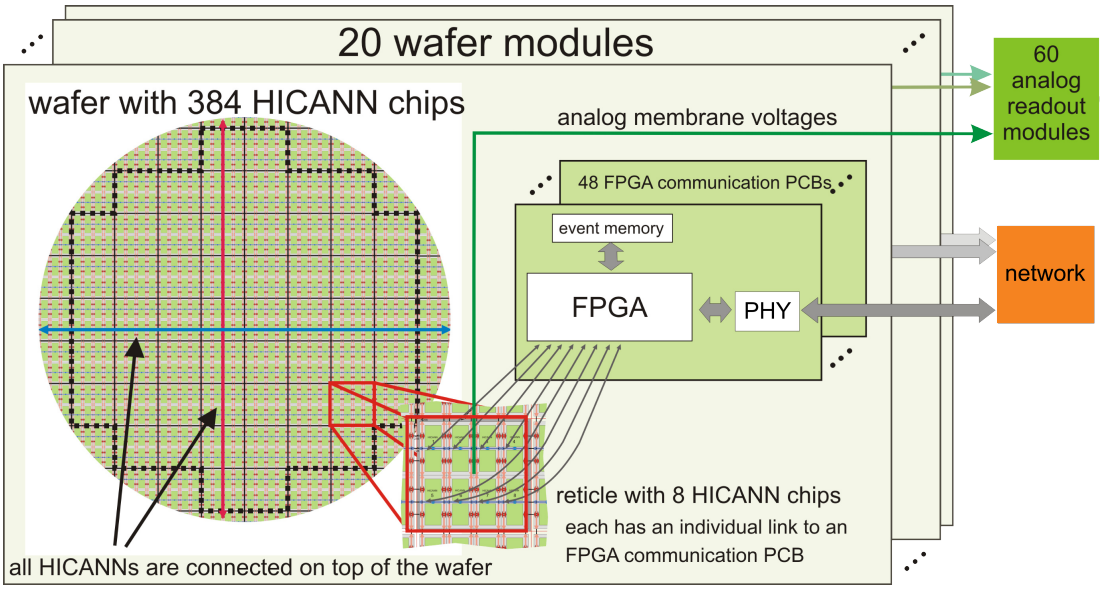


Figure 2.2.: Schematic overview of a NCS with 20 wafers. Graphic adapted from "A wafer-scale neuromorphic hardware system for large-scale neural modeling" [22].

and-fire model for up to 512 spiking neurons with a total of 114 k synapses [24]. The total of 48 reticles per wafer therefore are capable to model 196 k neurons and 44 M synapses. This calculates to 4 M neurons, 880 M synapses for a system of 20 combined wafers. The commissioning phase of 20 wafers is planned for March 2015 (month 18 of the HBP) and these should operate by March 2016 (month 30 of the HBP) [24].

2.2. Neuronal Traffic

In this section, the occurring pulse event traffic within the NCS is estimated. As previously stated the emulated neurons are partitioned into 48 reticles per wafer, each reticle's communication being managed by a FPGA. These 48 FPGAs are the source and target nodes of the interconnecting network for which the suitability of the EXTOLL Toumalet is evaluated.
SP9 aims for supporting a 20 Gbit/s bidirectional bandwidth from and towards each single HICANN, which can be partitioned in steps of 5 Gbit/s for the function of transmitting and receiving channel [13]. In most models the overall amount of required bandwidth from and to all HICANNs should reasonably be the same, because every transmitted event has to be received at some point in time. Differences in experimental in- and output are neglectable in proportion to the total data transmitted. Due to this global bandwidth balance the number of HICANN configurations, that do not allow a partition into locally balanced groups of eight is rapidly declining with an increasing quantity of available HICANNs. However, whether such an allocation is feasible depends on the modeled neuronal network and its mapping on the NCS. In case specific neuronal network topologies cannot be mapped in this manner (e.g., due to on-wafer connections, very high monitoring bandwidth or massive outside stimuli) the number of active HICANNs at unbalanced FPGAs could be reduced under the condition enough HICANNs are available. Therefore, I assume the HICANNs are arranged in a way that each group of eight HICANNs connected to an FPGA has an average receiving and transmitting bandwidth of 10 Gbit/s each. Otherwise for network technologies like EXTOLL, that do not support splitting the total bandwidth of one connection unequally, the provided bandwidth would have to be over-designed for most applied neural network models. With the EXTOLL capabilities and features it would be advisable for further development of the NCS to separate the individual HICANN communication paths and balance transmitting and receiving bandwidth.
In the current implementation, pulse events that are exchanged between a HICANN Interface and its HICANN are 8 bit wide. Combined with 8b/10b encoding the pulse event rate results in a rate of 1 GEv/s (Giga–Events/s). In conclusion the required bandwidth at each FPGA has to support a throughput of 8 GEv/s in each direction. In the FPGA pulse events will be extended to 27 bit, additionally containing source HICANN ID and timestamp. This implies that even without any overhead the raw bandwidth demand at an FPGA results in 216 Gbit/s. The Kintex-7 XC7K160T FPGA only supports eight 12.5 Gbit/s links [1], four of these are used for on-wafer communication in the current NCS implementation [6], thus due to 8b/10b encoding a total

data bandwidth of up to 40 Gbit/s is available for network traffic.
This described Kintex setup is not final. At the end of development a bandwidth of 20 Gbit/s might be achieved between FPGA (or a comparable entity) and a HICANN, but currently this bandwidth attains approximately 1 Gbit/s in each direction. This results in 100 MEv/s per direction and HICANN. Thus 21.6 Gbit/s of raw data bandwidth at the FPGA for 27 bit wide pulse events can be obtained.

2.3. Latency

What are the NCS's requirements with regard to latency? There is no general answer to this question—it strongly depends on the individual experimental setup. In biological neuronal networks a target neuron receives an action potential after a delay of under 1 ms up to over 10 ms [23, Chapter 8]. In some neuronal models delays between pre- and postsynaptic neuron are modeled with 1.5 ms to 8 ms (e.g., 1.5 ms to 3 ms in "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons" [3] or 2 ms to 8 ms in "Functional consequences of correlated excitatory and inhibitory conductances in cortical networks" [15]). Another typical method is using a delay of 0.1 ms (or a multiple of it) with the PyNN API [5] and for example NEST as its simulator back-end [12], as the default time step in a PyNN script is 0.1 ms [21]. With a speedup factor of 10^3 this implies a latency of 0.1 µs for the NCS. For an off-wafer network combined with the current implementation, including a FPGA and not a full custom ASIC, latencies of 0.1 µs are quite a challenge. To put this into perspective of recent research: measurements with a speedup factor of 10^4 quantified a minimal latency of 0.12 µs for on-wafer communication [6]. This corresponds to 1.2 ms on a biological time scale.
In conclusion there is no fixed target for latency requirements, but an off-wafer network's latency should not differ from the on-wafer latency by more than one order of magnitude to be compatible with a variety of neuronal models. The current state of the NCS's development fulfills the preconditions to realistically acquire a latency of maximal 1 µs.

3. The EXTOLL Network Technology

The Computer Architecture Group lead by Prof. Dr. Ulrich Brüning at the University of Heidelberg started the EXTOLL project in 2005 [20]. Six years later, in 2011 the spin-off company EXTOLL GmbH was founded [20]. The EXTOLL network technology is extraordinary in a number of features: ultra low latency message exchange, extremely high hardware message rate, low memory footprint, a switchless design and high scalability [20].
In the previous chapter the NCS's demands on bandwidth and latency were described. Taking into account these requirements the suitability of the EXTOLL network technology is evaluated. The latest EXTOLL Application-Specific Integrated Circuit (ASIC) is the Tourmalet, which will be discussed in the following section.

3.1. The EXTOLL Tourmalet

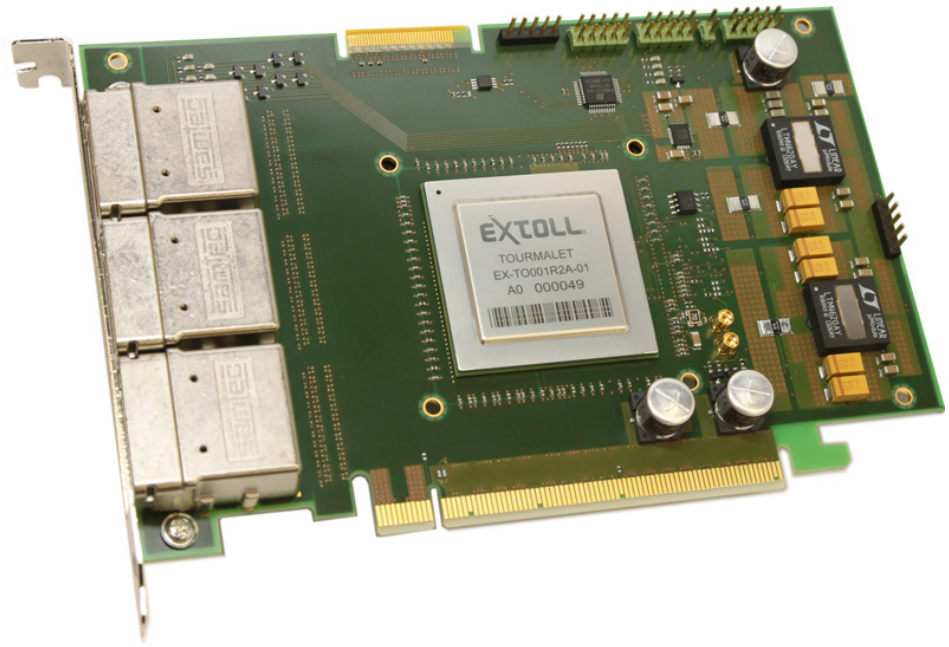


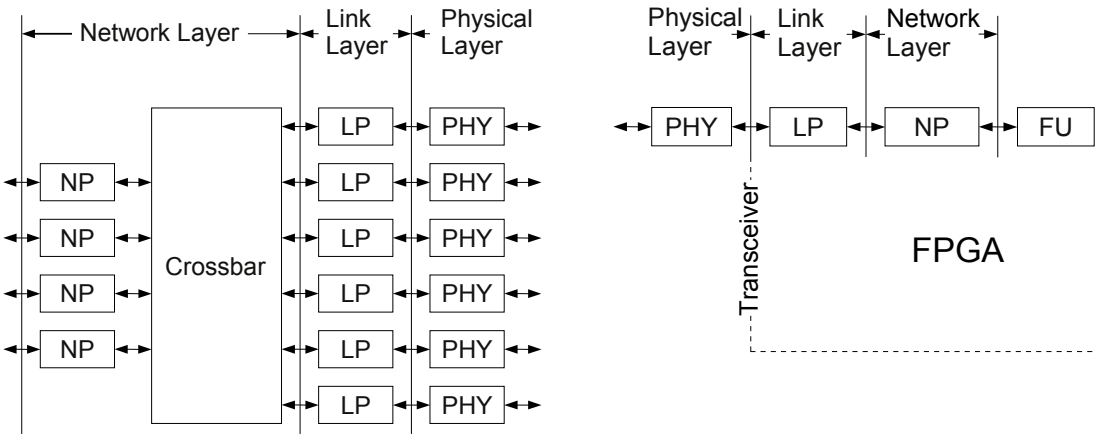
Figure 3.1.: The Tourmalet network card. Graphic adapted from *Introducing EXTOLL Tourmalet, an HPC Network ASIC* [20].

For the work related to this thesis project only a subset of the multiple features of the EXTOLL Tourmalet are relevant and presented in more detail. The hardware architecture can be

differentiated into three entities: The host interface, the network interface and the network part itself. A x16 PCI Express 3.0 or a x16 HyperTransport (HT) 3.0 with 2600 MHz can be used to interface a host system [8]. Toumalet's Network Interface Card (NIC) apart from other protocols supports Open MPI 1.6x for low level API access via the Remote Memory Access (RMA) unit [8]. Arbitrary topologies with a bandwidth of 120 Gbit/s per link port and direction can be realized with the six link ports of a Tourmalet NIC [8]. The NIC's network switch has one additional link for board integration [20].

3.2. The EXTOLL Network Protocol

The EXTOLL network protocol defines three protocol layer: the network layer, the link layer and the physical layer (Figure 3.2). The link layer consists of two link ports for every connection within the network. Reliable transmission of packet cells between link ports is ensured by a retransmission protocol [4]. All data traffic between network nodes is also processed by the physical layer. This layer performs tasks such as link initialization, line coding and detects defect lanes of a connection [4]. The network layer handles the packet routing. Part of the network layer is the EXTOLL network port, which can be directly interfaced in a custom hardware design. In combination a link port and a physical link a Functional Unit (FU) is able to send and receive EXTOLL packets.



(a) The EXTOLL network protocol layers as implemented in Tourmalet.

(b) An exemplary custom FPGA design that supports an EXTOLL network.

Figure 3.2.: Standard and custom implementation of the EXTOLL network protocol layers. Graphics adapted from the EXTOLL Network Protocol Specification [4].

Section 3.3 describes how the network port can be interfaced. The packet format is described in the following.

In an EXTOLL network, packets consist of 64 bit wide cells [4]. Apart from data cells, which carry the payload, there are control cells that frame data cells to form packets [4]. They are also

used by the link layer and the physical layer to take care of a variety of network internal tasks, such as exchanging flow control credits or initializing link connections [4]. Figure 3.3 shows the structure of control cells. After eight control bits for encoding in the physical layer 4 bit determine the cell type. The next 36 bit are the control cell payload; its structure depends on the cell type.

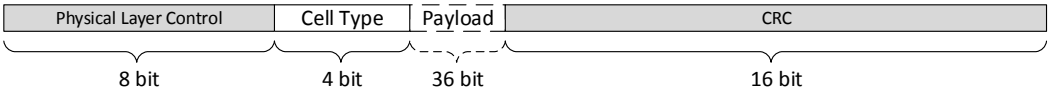


Figure 3.3.: General control cell format

In the current context the Start Of Packet (SOP) control cell is of importance. This type of control cell contains the routing information for following data cells [4]. As depicted in Figure 3.4 this routing information is comprised of 1 bit to distinguish between unicast and multicast packets, 1 bit each for choosing the Adaptive or a Deterministic Virtual Channel (AVC/DVC), 2 bit Traffic Class (TC), 16 bit node ID, 3 bit Target Unit (TU) and 10 bit virtual process ID [4].

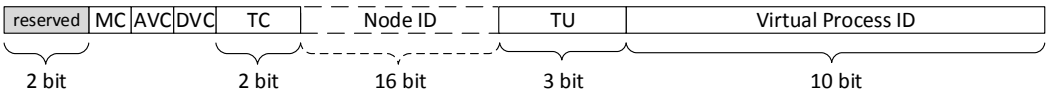


Figure 3.4.: Payload of a SOP control cell (cell type 0x4)

In case of a multicast packet the six most significant bits of the node ID are interpreted as a multicast group ID, thus 64 local multicast groups can be defined. With a sophisticated mapping algorithm these multicast groups may theoretically overlap one another, effectively distinguishing between even more different groups. As these multicast groups do not have to be globally unique, the total number of multicast groups scales up with the size of the network. If the Adaptive Virtual Channel (AVC) bit is set, the packet is routed adaptively. Such packets may not be received in sequence. If the adaptive routing buffer at a node is full the routing falls back on a deterministic channel. There are two deterministic virtual channels, which are distinguished by the Deterministic Virtual Channel (DVC) bit, when the AVC bit is set to zero. AVC and DVC bit must not both be set in a single SOP control cell. Network packets can be assigned to one of four Traffic Classes (TCs), which cannot jam each other. The 16 bit node ID allows to address a node in a network of 64 k nodes. The Target Unit (TU) bits identify the target module (e.g., RMA) within the Network Interface of the target node. The virtual process ID is needed for packets sent to the RMA or another TU called VELO to identify the target process in environments like Message Passing Interface (MPI).

After up to 32 data cells an End Of Packet (EOP) control cell indicates the end of a packet. This cell contains a 32 bit packet CRC along with four reserved bits [4]. EOP control cells must not be sent into the network, because they are generated in the EXTOLL network port.

3.3. The EXTOLL Network Port Interface

The Network Port (NP) is divided into sender and receiver. The NP sender forwards data from the FU to the crossbar, thereby into the network. The NP receiver works inverse. Both data paths are buffered as depicted in Figure 3.5.

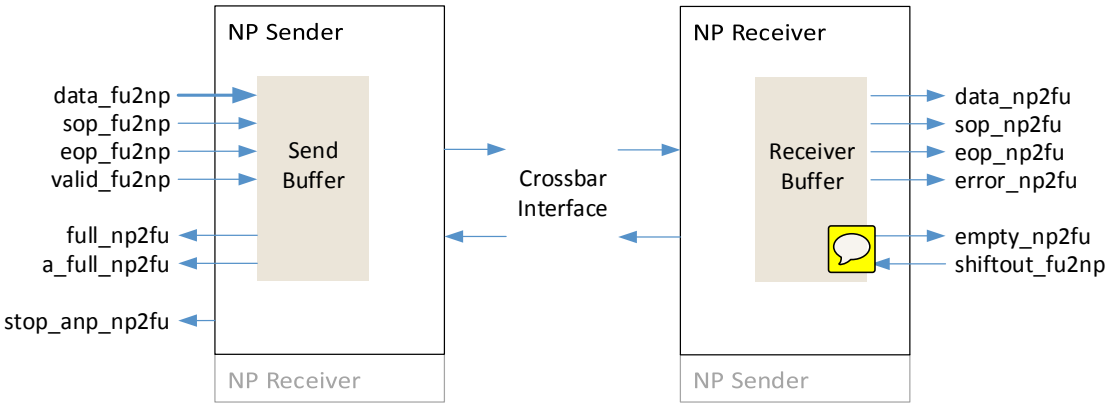


Figure 3.5.: The interface between a Functional Unit (FU) and the EXTOLL Network Port (NP), which is logically divided into a sender and a receiver.

The NP sender's and receiver's mode of operation is controlled with a register file. The NP sender can be adjusted to delay the packet transmission until a threshold of cells is stored in the send buffer. Other than this, if packets of a size below the appointed threshold are to be sent, these will be sent when completely buffered [11]. The NP receiver can either operate in a "cut-through" mode or in a "store-and-forward" mode. In "cut-through" mode single packet cells are stored in the receive buffer directly on recipience [11]. As a result, the FU has to be able to handle corrupt packets. Alternatively the NP receiver can operate in "store-and-forward" mode. In this mode only complete and errorless packets are stored in the receive buffer making error handling dispensable at the price of slightly higher latency.
In order to be able to use the NP in a custom hardware design the signals at the NP sender's and receiver's interface are explained in in more detail.

NP sender

data_fu2np This bus transports all 64 bit wide packet cells from the FU to the NP. Like in all EXTOLL modules the bus width can be a multiple of 64 [11]. However the first cell of a packet has to be a SOP control cell.

sop_fu2np This signal indicates that the current cell is a SOP control cell [11].

eop_fu2np This signal indicates that the current cell is the last cell of the packet [11].

valid_fu2np If this signal is set and *full_np2fu* is not set the *data_fu2np* bus is shifted into the NP Send Buffer [11].

full_np2fu This signal indicates that the NP Send Buffer is full and no data will be accepted [11].

a_full_np2fu This signal indicates that a whole network packet would exceed the capacity of the NP Send Buffer [11]. This means an ongoing transmission has not to be disrupted.

stop_anp_np2fu This signal indicates that the packet transmission should be halted before issuing the next packet [11]. However, within the next three clock cycles new packets are still accepted and ongoing transmissions may be finished [11].

NP receiver

data_np2fu Analog to the *data_fu2np* signal interfacing the NP sender [11].

sop_np2fu Analog to the *sop_fu2np* signal interfacing the NP sender [11].

eop_np2fu Analog to the *eop_fu2np* signal interfacing the NP sender [11].

error_np2fu This signal indicates that the current packet from the last received SOP control cell until the currently transfered cell is corrupt [11].

empty_np2fu This signal indicates if the receive buffer is empty or stores at least one cell [11].

shiftout_fu2np Via this signal a received cell can be shifted out by the FU. It also allows to request the next cell, if the *empty_np2fu* signal is set. In this case a cell will be send to the FU as soon as it is stored in the receive buffer [11].

3.4. Latency

Within the EXTOLL network one hop takes approximately 60 ns [20]. One hop into the network is obligatory, as well as an offset delay of about 484 ns to 640 ns for this specific application of the current NCS. This offset delay consists of a short delay in the HICANN (< 4 ns [6]), the transmission delay between HICANN and `hicann_if` module (224 ns in "rx" plus 184 ns in "tx" direction [7]) and the pulse event processing in the `hbp_extoll_if` module (18 to 42 clock cycles at 250 MHz plus up to 60 ns delay in the accumulation time slot resulting in 72 ns to 228 ns as later discussed in more detail in section 6.2).
The achievable latency depends on network size and topology. There is always a minimum latency of about 550 ns plus 60 ns per network hop.

3.5. Suitable Network Topologies

With regard to the network topology, the neuronal model topology of a random network is the one with the highest demands. Regarding this worst case, the best scalable Tourmalet network topology is a three dimensional torus. To be able to compare different tori topologies two variables are defined: d_T, the topology diameter—the maximum number of hops through the torus network—and B_T, the supported utilization per I/O link, assuming homogeneous traffic distribution within the network. B_T serves for comparison only, as homogeneous network traffic is the best case scenario for bandwidth utilization within the network. To estimate B the average number of required hops $d_\mathrm{a} = \frac{d_\mathrm{T}}{2}$ (for random network traffic), the number of I/O links l_IO and the number of intra-network connections l_intra is used. In complete symmetric tori with N_T nodes and $3N_\mathrm{T}$ connections d_T and B_T can be calculated as follows:

$$d_\mathrm{T} = \left\lceil \frac{3}{2} \cdot \left(\sqrt[3]{N_\mathrm{T}} - 1 \right) \right\rceil \tag{3.1}$$

$$B_\mathrm{T} = \frac{l_\mathrm{intra}}{l_\mathrm{IO} \cdot d_\mathrm{a}} = \frac{3N_\mathrm{T} \cdot 2}{N_\mathrm{T} \cdot d_\mathrm{T}} = \frac{6}{d_\mathrm{T}} \tag{3.2}$$

As the FPGA bandwidth is limited to 40 Gbit/s (section 2.2) and a single EXTOLL Tourmalet link is capable of forwarding 120 Gbit/s, concentrator nodes have already been proposed in advance of this thesis [16]. With its total of seven links a Tourmalet concentrator node can combine up to six FPGAs, as shown in Figure 3.6.
Considering a 20 wafer NCS $20 \cdot 48 = 960$ FPGAs or at least $960/6 = 160$ concentrator nodes have to be connected. This large number excludes any trivial topology solutions like all-to-all, therefore a 3-torus with or without concentrator nodes is the selected network topology. In torus networks with one layer of concentrator nodes the maximum number of hops through the whole network is $d = d_{\mathrm{T}+2}$. The merging of several concentrator nodes with a second layer of concentrator nodes is inefficient, as supported bandwidth would be reduced extremely.

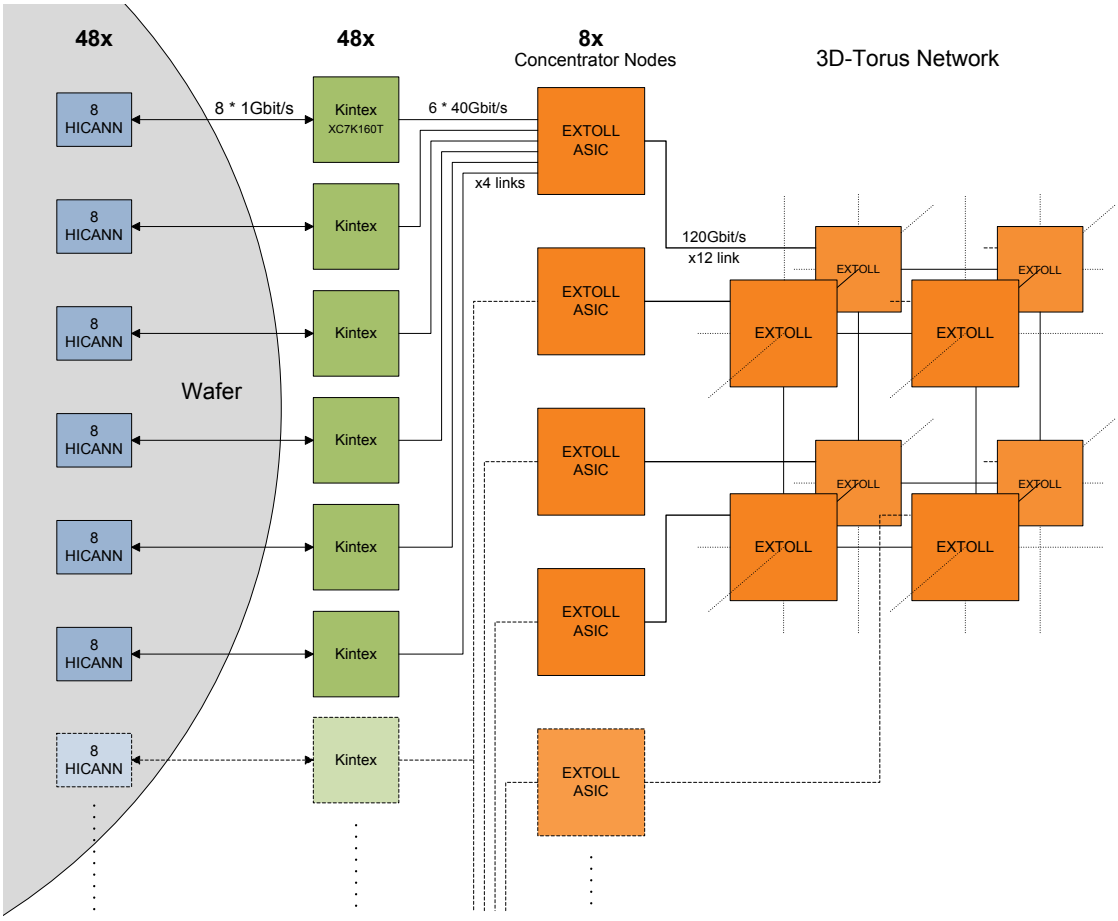


Figure 3.6.: Design of an EXTOLL network with a 3-torus topology and concentrator nodes that combine six FPGAs (6:1). Graphic adapted from hbp_extoll_networking_v2 [16].

Independent of the emulated neuronal network's activity it is reasonable to combine a minimum of three FPGAs with one concentrator node (3:1)—three FPGAs' maximum bandwidths sum up to the maximum bandwidth of one Tourmalet link. A special scenario is the combination of five FPGAs, as this allows for a double connection between the concentrator node and the network (5:2), supporting full bandwidth for five FPGAs at each concentrator node (two or more network connections for lower ratios would be overdesigned). There are two possible network configurations to fit the 5:2 concentration: Two parallel 3-tori each of the size to fit a 5:1 concentration and an "open" 3-torus. The later is a 3-torus like topology with only $l_{\text{intra}} = 2N_{\text{T}}$ intra-network connections, leaving additional N_{T} connections free to connect additional concentrator nodes ($l_{\text{IO}} = 2N_{\text{T}}$). Different concentrator node setups are compared in Figure 3.7

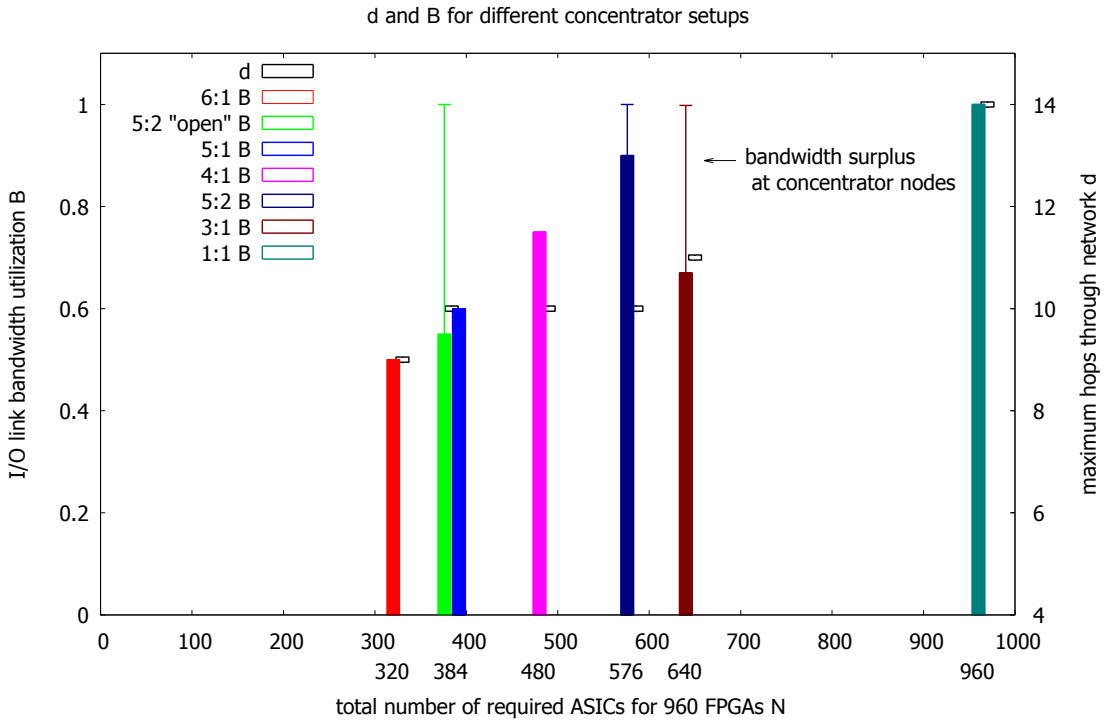


Figure 3.7.: Possible average utilization of I/O node links and maximal number of necessary network hops in a 3-torus topology, assuming every node has one I/O link and six intra-network connection links.

It is evident that the 1:1 setup is overdesigned. It is even disadvantageous compared to the concentrated networks, due to its large 3-torus diameter and despite the fact that concentrated networks require two additional nodes. A large number of EXTOLL ASICs (1:1 and 3:1) has little advantage over low (6:1, 5:2 “open” and 5:1) to medium numbers of network nodes (4:1 and 5:2). Comparing the 5:2 setup to the 3:1 setup it is more likely that the 3:1 setup jams for high overall network load. This conclusion is incomplete, as three free links at each concentrator node in a 3:1 setup was neglected in this analysis. It might be possible that the 3:1 configuration has an advantage for locally coupled networks. However, a symmetric network topology with 3:1 concentrator nodes is not feasible. Both 5:2 configurations have the advantage of full bandwidth support at the concentrator nodes. This allows the network to handle local bursts, if the overall network traffic is not too high. In case of high overall network traffic a problem for the 5:2 “open” setup could arise, as the overall average of I/O traffic may not exceed 60% of the maximum bandwidth per Kintex.

All the discussed network topologies implemented in EXTOLL ASICs with a price of approximately $ 1000 per ASIC are cheaper and possess a higher performance than comparable FPGA switch designs. In order to keep costs as low as possible the 6:1 setup allows for efficient networking, but its concentrator nodes limit bandwidth to half of the full maximum of six

FPGAss. If this is not sufficient emulated neural network activity bursts could be managed by an “open” 5:2 configuration which only needs 20% more resources. If overall network traffic is the limiting factor a 5:1 or 4:1 setup might be required. The largest efficient configuration—the 5:2 setup—requires almost doubled resources compared to a 6:1 setup, but wont cause any additional delays neither due to bandwidth peaks nor due to high overall network traffic rates. To gauge the most suitable configuration more simulations of neural network models have be performed with the discussed topologies.

4. The FPGA's Internal Structure

As mentioned in section 2.1 the current neural accelerator's control logic is implemented in 48 Kintex-7 XC7K160T FPGAs per wafer unit. The structure of such a FPGA is now examined as a preparation for integrating an EXTOLL interface module.
The FPGA's structure is divided into interface and core logic as depicted in Figure 4.1. All data for configuration is forwarded by the UDP interface to which a Joint Test Action Group (JTAG) interface is connected for testing purposes. There is a System monitor module connected to an I²C interface for observing the systems performance, e.g. occurring CRC errors or system temperature. The host Automatic Repeat Request (ARQ) module implements a retransmission protocol. There are three DDR3 interfaces to connect one 256 MB DDR3-1600 SDRAM for Ethernet buffering (connected to the host ARQ) and two—one each for trace and playback data—512 MB DDR3-800 SDRAMs with a 32 bit interface to their controller modules. The trace memory tracks the occurring pulse events during an experiment and can be read out afterwards. The playback memory can be filled with pulse events to set the preconditions for an experiment.

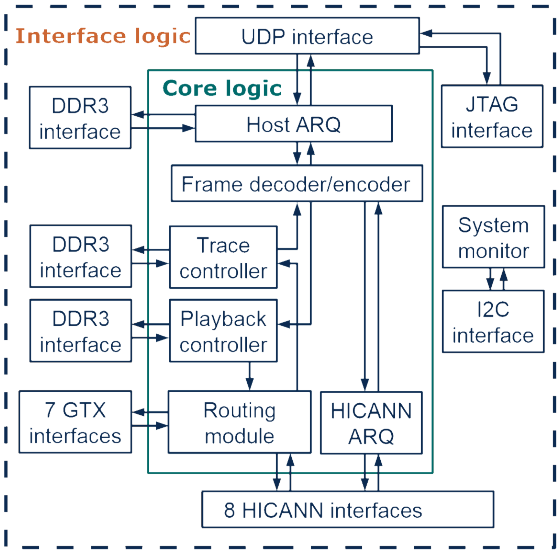


Figure 4.1.: Main modules of the Kintex-7 FPGA firmware. Graphic apdapted from the Neuromorphic Platform Specification [24].

The HICANN ARQ ensures configuration data communication with the HICANNs, which are interfaced through eight HICANN interfaces. From these interfaces pulse events are sent to

a routing module (and vice versa), which is connected to to the playback controller and also interfaces the Kintex' GTX links. A routing module with the described capabilities is still in development. Still the HICANN Interfaces are the best module to connect an external network to and are therefore described more closely in the following section 4.1.
As the structure of the FPGA is liable to change, refer to most current details on this matter in the Neuromorphic Platform Specification [24].

4.1. HICANN Interface Specifications

In order to communicate with the HICANN Interface, in- and outgoing signals at the edge of the module are analyzed. To identify the signals' exact content, also the inner structure is examined.
The HICANN Interface provides a source synchronous serial DDR transmission from and towards one HICANN, and additionally handles en- and decoding of pulse and configuration packets. These functionalities are separated in four submodules: The `ddr_lvds_if_7series`, the `fpga_link_channel`, the `fpga_proto_link_ctrl` and the `packet_ctrl`. Additionally there is the `test_hicann_if_jtag_sync` module for testing purposes. The main signals within the HICANN Interface relevant for transmitting and receiving pulse and configuration data are shown in Figure 4.2. This figure also serves the purpose of displaying the interfaces of the `hicann_if` module towards the HICANN and the `hbp_extoll_if` module, the implementation of which is part of this thesis.
As shown in the bottom of Figure 4.2 connections to and from the HICANN are implemented solely in the `ddr_lvds_if_7series` module. As the HICANN operates on a slower running clock than the FPGA (1:8) and possesses no buffer capabilities, asynchronous FIFOs are used in the `ddr_lvds_if_7series` module to segregate the two clock cycles and to buffer incoming as well as outgoing data. For data transmission Low-Voltage Differential Signaling (LVDS) is used. The eight bit data units from/to the HICANN are combined/segmented in the `fpga_link_channel` module, which also initiates the link training.
The `fpga_proto_link_ctrl` module converts the link channel data into 64 bit packets and vice versa. These packets may contain configuration data or one or two pulse events. Towards the `packet_ctrl` module this is encoded as part of the packet format, in the reverse direction these options are flagged by the *config_enable* signals for configuration data, the *pulse_enable* signal indicates one pulse event and the *pulse_64bit* signal two, respectively. The `fpga_proto_link_ctrl` module is also capable of counting the occurred CRC errors.
At the `hicann_if` module's interface within the FPGA (shown at the top of Figure 4.2), the `packet_ctrl` module buffers incoming packets, which will be sent to the HICANN (*tx_pulse_event* and *tx_config_data*) in separate FIFOs (*tx_pulse_fifo_full* and *tx_config_fifo_full*). Outgoing packets (*rx_pulse_event* and *rx_config_data*) are directly forwarded and are lost if not processed by the receiving module.

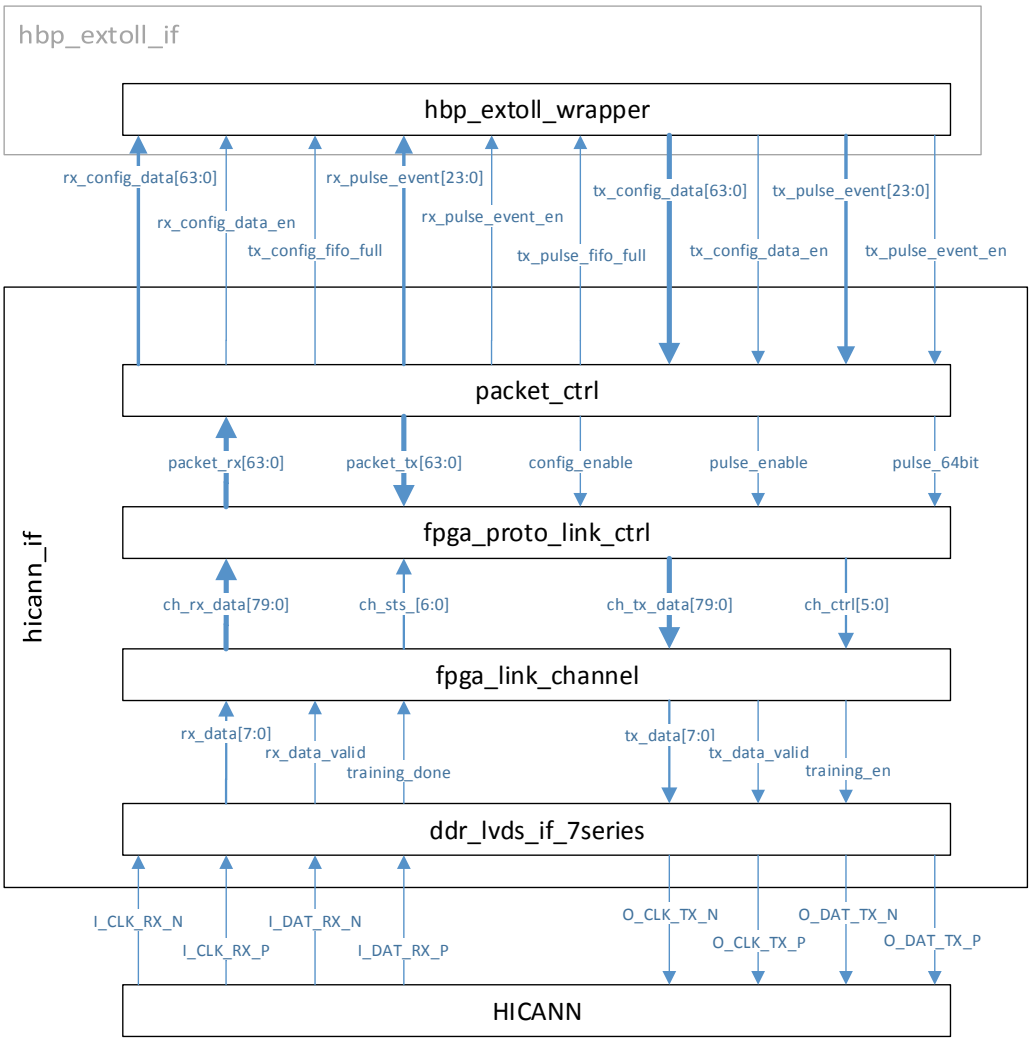


Figure 4.2.: Overview of the HICANN Interface's submodules and main signals. The top of the figure already suggests the EXTOLL interface with one of its wrapper modules. This part is discussed in more detail in chapter 6.

Within this thesis work the names of modules and signals mainly begin with `rx_` or `tx_`, where "rx" stands for receiving channel and "tx" for transmitting channel with regard to a HICANN. This annotation was chosen in analogy to the `hicann_if` module implementation in the subproject s_hmf_fpga of the p_brainscales project.
All signals at the `hicann_if` module's interface, which are relevant for pulse event and configuration data transmission, originate from or are received by the `packet_ctrl` module. This allows focusing solely on pulse events in the first implementation of an EXTOLL module. As explained pulse events and configuration data are completely separated from another at this point of data communication.

The 24 bit wide *rx/tx_pulse_event* signals contain a 15 bit timestamp, which encodes its time of origin. As the `hicann_if` module itself is work in progress, too, in future implementations this timestamp may instead mark the point of time at which the pulse event should take effect. The other part of a pulse event packet contains a 9 bit synapse identifier, labeling the pulse event's origin. This synapse driver address is unique within the FPGA domain, as its three most significant bits identify its source HICANN. It is envisioned to extend the 6 bit synapse address to 9 bit in further development, which is why the calculations within this thesis are based on 27 bit wide pulse events instead of the found 24 bit.

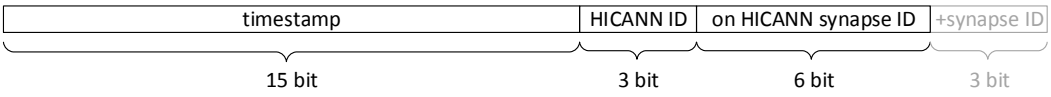


Figure 4.3.: The bit structure of pulse events at the HICANN interface.

Signals Relevant for the EXTOLL Network

clk_sys — The general clock signal within the FPGA is operating at a frequency of 125 MHz.

a_reset_sys_h — Within the FPGA exists an asynchronous, active high reset signal.

`[23:0]` *tx_pulse_event*
`[23:0]` *rx_pulse_event* — A neuronal pulse event received by ("rx") or going to be transmitted to ("tx") a HICANN. The most significant bits of these buses consists of a 15 bit timestamp, which indicates time of the pulse's origin. The remaining bits encode the source synapse driver including its source HICANN ID at the three most significant bits.

tx_pulse_event_en
rx_pulse_event_en — These signals indicate the validity of the above elucidated "tx" and "rx" pulse event buses. If one signal is not set the corresponding bus value is undefined.

tx_pulse_fifo_full — If this signal is set the receive buffer for pulse events in "tx" direction is full. Pulse events wont be shifted in.

`[63:0]` *tx_config_data*
`[63:0]` *rx_config_data* — These buses transfer configuration data in and out of the `hicann_if` complying with the ARQ protocol.

tx_config_data_en
rx_config_data_en — These signals indicate the validity of the above elucidated "tx" and "rx" data buses. If one signal is not set the corresponding bus' value is undefined.

tx_config_fifo_full If this signal is set the receive buffer for configuration data in "tx" direction is full. Configuration data wont be shifted in.

`[14:0]` *systime* The system's time counter, used to generate a pulse event's timestamp. The three least significant bits are generated locally. The implementation of *systime* is work in progress.

loopback_en This signal enables an internal loopback, sending back received pulse events without using the analog connection towards the HICANN. This is used for testing purposes only.

use_timestamp If this signal is set the `hicann_if` module uses eight heap buffers to schedule pulse events in "tx" direction according to their timestamp.

`[63:0]` *kill_stat* For each of the above mentioned heap buffers *kill_stat* provides an 8 bit counter for the number of those pulse events that had to be discarded due to full heap buffers or if they were received behind schedule.

`[7:0]` *kill_stat_reset* For each discarded pulse event counter exists a separate reset signal.

Other Signals

`[7:0]` *channel_status* The *channel_status* holds information about the data channel from the `hicann_if` module to the corresponding interface in the HICANN.
channel_status [0]: ready to send a pulse event packet
channel_status [1]: initializing link [7]
channel_status [2]: CRC error occurred [7]
channel_status [3]: reserved (was used in an earlier implementation) [7]
channel_status [4]: received valid configuration packet if no CRC error occurred
channel_status [5]: received valid pulse event packet if no CRC error occurred
channel_status [6]: ready to send a configuration packet
channel_status [7]: pulse event packet contains two pulse events

channel_reset This signal resets the *channel_status* register.

clk_hs — The *clk_hs* signal is a reference clock for the HICANNs with a frequency of 62.5 MHz [24].

reset_hs — The HICANN has this seperate active high reset signal.

`[7:0]` *crc_count* — This register counts the occurring CRC errors for all packets received from the HICANN.

crc_count_rst — This signal resets the above named CRC error counter to zero.

auto_init — This signal enables the use of an automatic eye pattern search algorithm in the HICANN [7]. It is set by default.

init_master — This signal determines the master communication partner for the connection between FPGA and HICANN. As default the FPGA is the master and its HICANN is the slave [7].

start_link — This signal initiates the link training for the HICANN connection [7].

pulse_protocol
config_protocol — These signals indicate a successful CRC for received pulse events and configuration data, respectively. They are not used any anymore [7].

dc_coding — This signal enabled the use of 8b/10b encoding within the FPGA. As this method did not lead to any measurable improvements it was abandoned [7].

`[99:0]` *routing_data*
routing_data_en
`[7:0]` *heap_mode*
`[10:0]` *limit auto_limit* — These four buses and signals can be used to enable special buffer functionalities that are not necessary for standard usage [7].

O_CLK_TX_P
O_CLK_TX_N — These signals transfer the reference clock signal towards the HICANN using LVDS.

I_CLK_RX_P
I_CLK_RX_N — These signals transmit the HICANN clock signal to the `hicann_if`, where it is needed in the asynchronous FIFOs.

O_DAT_RX_P
O_DAT_RX_N — These signals implement the LVDS communication path towards the HICANN.

I_DAT_RX_P
I_DAT_RX_N — These signals implement the LVDS communication from the HICANN into the `hicann_if`.

4.2. Signal Timing at the HICANN Interface

The signal timing is a crucial element of interfacing a module. FUs connected to the `hicann_if` module have to process incoming "rx" data instantaneously, because there are no "rx" output buffers integrated. This applies to pulse events and configuration data alike. The timing in this case is depicted in Figure 4.4 using the example of pulse events.

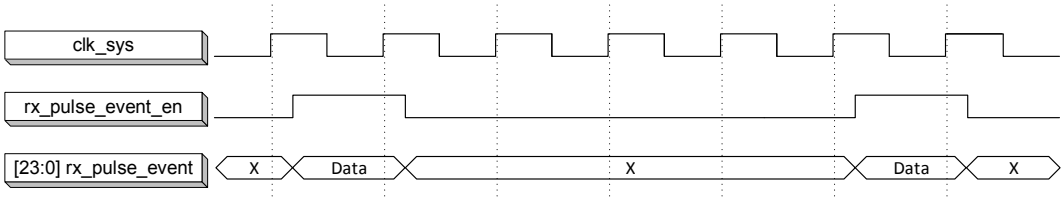


Figure 4.4.: Signal changes at clock edge of "rx" pulse event signals at the `hicann_if` interface.

As there are input buffers for the "tx" communications paths, valid data will be shifted in, unless the corresponding input buffer is full. The connected transmitter unit in the pulse event example represented by Figure 4.5 detects the full-status of the pulse event input buffer and reapplies the data2 signal until the input buffer has free capacity.

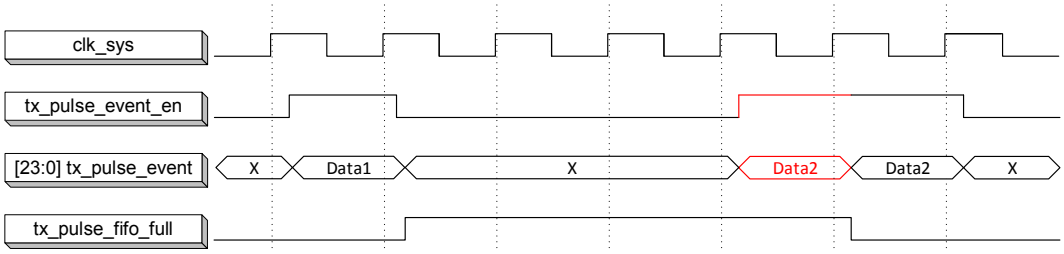


Figure 4.5.: Signal changes at clock edge of "tx" pulse event signals at the `hicann_if` interface.

5. Designing an EXTOLL Interface Module

To establish EXTOLL as part of the implementation in the Kintex, first a simplified module will be included in the existing implementation. The first design should enable the system to use an EXTOLL network for direct pulse communication between different FPGAs, omitting configuration data and read-out functionality for the time being. As latency is not as crucial for read-out and configuration functionality, as it is for pulse event transmission, it is not a problem to focus on pulse events for the first design draft. Moreover, pulse event packets are potentially the smallest packets passing through the network, thus probably causing the highest overhead. Several strategies have to be analyzed to determine a basic design of an EXTOLL Interface module. However, the top level integration into the existing `hmf_fpga` module has to be structured like shown in Figure 5.1 to be consistent with the FPGA structure described in chapter 4.

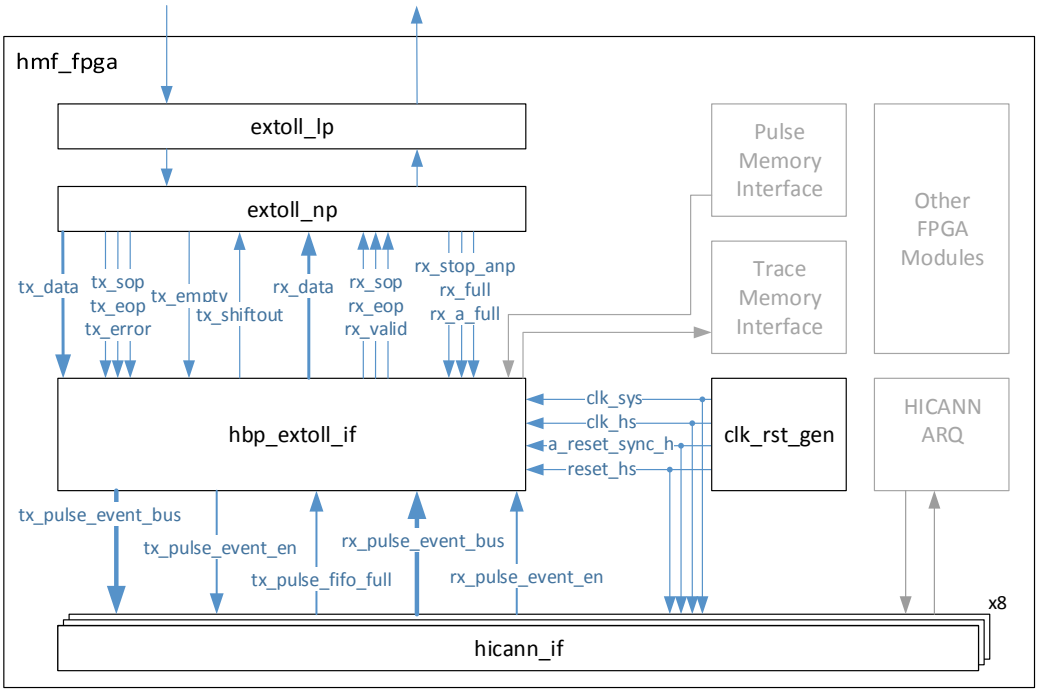


Figure 5.1.: FPGA structure with an integrated `hbp_extoll_if` module.

5.1. Routing Strategy

As pointed out in section 3.2 a 64 bit wide SOP control cell is inevitable for any packet within the EXTOLL network and all data within the network has to be aligned to 64 bit. A pulse events only takes up 27 bit. To avoid massive overhead, pulse events with the same destination node could be sent together. The limited time frame each pulse event is valid is disadvantageous in this, but the network's low latency may allow for further delay.
Another very appealing method is the use of multicast packages, because naturally a single pulse event reaches lots of neurons, as a single neuron has lots of synapses. How common the applicability of multicast packets is, strongly depends on the properties of the neural network which is to be emulated. In uniform random networks—a widely used model for neuronal networks [19, 3]—it is obvious that almost every pulse event has to be forwarded to a number of EXTOLL network nodes as all target neurons are uniformly distributed over the whole network. On the other hand in locally coupled networks—another suggested model [17]—it might often be the case that all postsynaptic neurons are placed within a single reticle, thus the pulse event would only have to be passed through the EXTOLL network once. As the maximum of implemented neurons in a reticle is currently limited to 4096, the target neurons often have to distributed over more than one reticle. In conclusion the EXTOLL Interface module should definitively support multicast packets. Due to the current limit of 64 local multicast groups the implementation should not be restricted to multicast packets. In analogy to unicast packets multicast pulse events could be accumulated in order to reduce the necessary overhead.
If several pulse events are sent in a single packet (unicast or multicast) the duration of accumulation has to be determined. The simplest method is to compare the difference between the pulse event's timestamp and the current system time to a constant threshold. This threshold has to correspond to the difference between the minimally acceptable delay of a pulse event and the maximally occurring latency in an EXTOLL network utilized to its capacity. A more exact approach is the adaptation of this threshold to the specific delay of the pulse events that are on hold or to the worst case latency of the specific node connection (or connections in case of a multicast packet).

6. The Extoll Interface Module

In this chapter the implemented `hbp_extoll_if` module is documented. In its described state of development it supports the communication of pulse event data.

6.1. Interface Specification

For a better overview of the `hbp_extoll_if` module the communication channels from and towards the EXTOLL NP and HICANN Interface are illustrated in Figure 6.1.

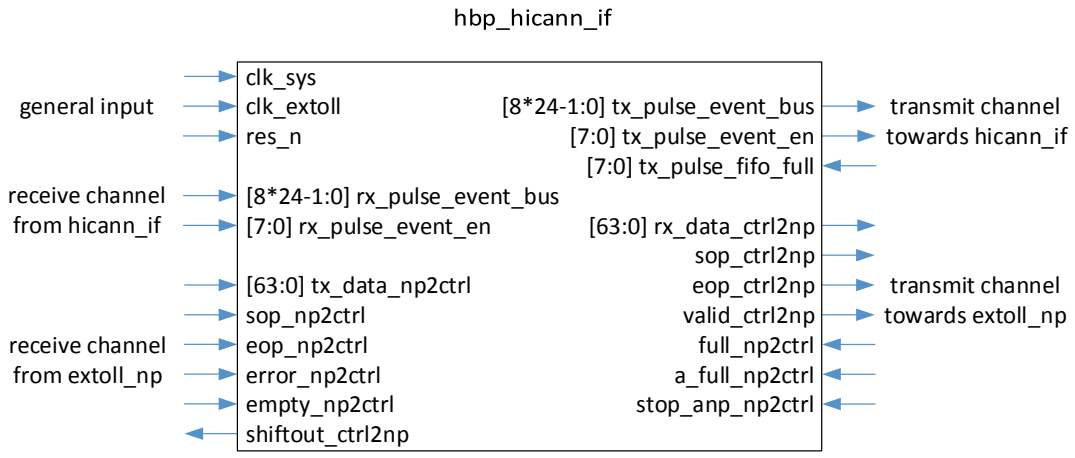


Figure 6.1.: The `hbp_extoll_if` module interface.

6.2. Inner Structure

Figure 6.2 depicts the inner structure of the `hbp_extoll_if` module. “tx” and “rx” communication paths are completely seperated. Only due to the analogy of the “tx” and “rx” FIFOs that buffer data transmission and seperate EXTOLL’s clock domain, the `hbp_extoll_wrapper` module is part of both communication paths.

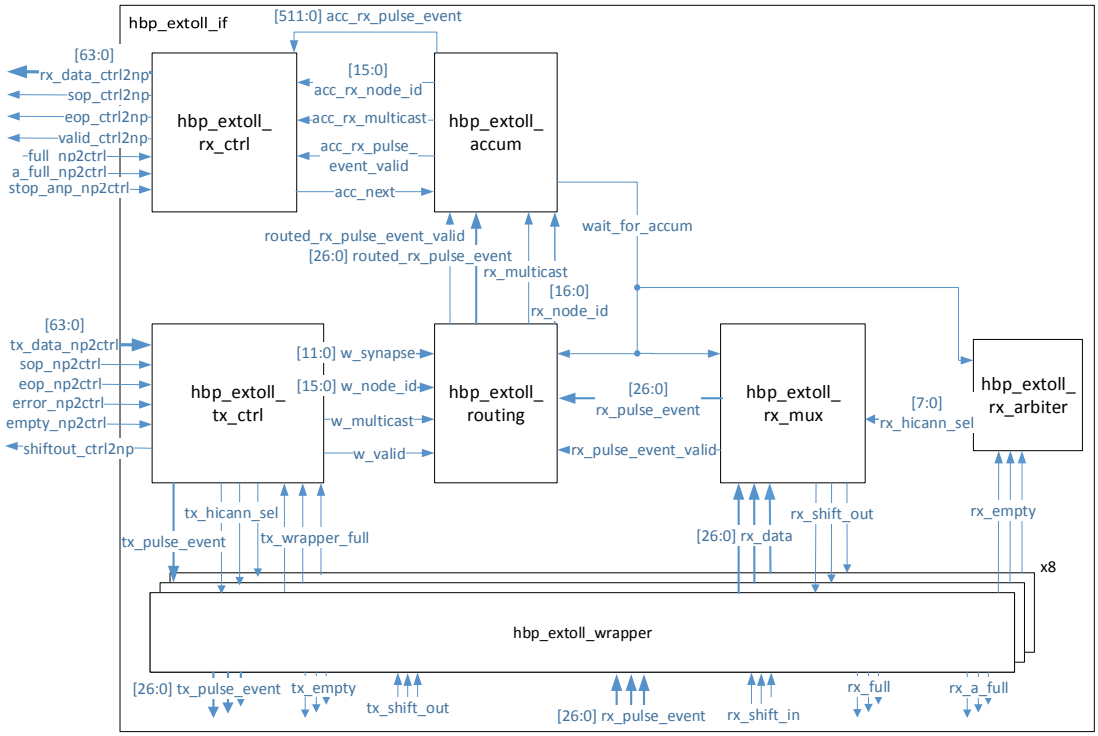


Figure 6.2.: The inner structure of the `hbp_extoll_if` module.

For the whole `hbp_extoll_if` the worst case number of clock cycles add up to 42 plus additional 15 clock cycles that it may take to process pulse events scheduled in the same time slot (refer to subsection 6.2.5 for more detail on this). This corresponds to a delay of 228 ns for a 250 MHz clock signal. If all buffers are empty the minimum number of clock cycles is 18, which corresponds to a delay of 72 ns.

6.2.1. `hbp_extoll_wrapper`

This module consists of two register based First In, First Out buffer structures (FIFOs). One of them buffers *rx_pulse_events*, that are received from the `hicann_if` module via the *rx_pulse_event_bus*, the other one buffers outgoing *tx_pulse_events* towards the `hicann_if` module (via *tx_pulse_event_bus*). To support a different clock signal within the `hbp_extoll_if` module (*clk_sys* and *clk_extoll*) these FIFOs have to be asynchronous.
There are as many `hbp_extoll_wrapper` in a `hbp_extoll_if`, as there are `hicann_if` modules in the `hmf_fpga_top` module (equals the number of HICANNs per FPGA).

6.2.2. `rx_mux`

This multiplexer (mux) forwards one of the *pulse_event* signals of all `hbp_extoll_wrapper` towards the `hbp_extoll_routing` module. From which `hbp_extoll_wrapper` a pulse event is selected is determined by the one-hot *rx_hicann_sel* bus and indicated towards the

`hbp_extoll_wrapper` modules by the *rx_shift_out* bus. If the incoming *wait_for_ctrl* signal is high the mux pauses, not providing any pulse events for the routing module nor shifting out any data with *rx_shift_out*.

6.2.3. `hbp_extoll_rx_arbiter`

Given the *rx_empty* bus, which contains the *rx_empty* signals from all `hbp_extoll_wrapper`, this module generates the above mentioned *rx_hicann_sel* bus in a round-robin manner.

6.2.4. `hbp_extoll_routing`

This module provides the routing information in form of a Random Access Memory (RAM) based LookUp Table (LUT). This block memory saves 1 bit for the multicast command and another 16 bit for an EXTOLL node ID in case of unicast (*rx_multicast* $= 0$), or a 6 bit multicast group ID in case of multicast events. Further more the block memory holds a 4 bit wide "durable delay" that indicates the acceptable latency for each pulse event. For a quick and efficient accumulation of pulse events this durable delay has to be the same for every neuron connection with the same destination node (see also subsection 6.2.5 `hbp_extoll_accum`). During the routing lookup the incoming *rx_pulse_event* is buffered, hence after routing the *rx_multicast* flag and the *rx_node_id* are valid contemporaneously with the forwarded *routed_rx_pulse_event*. As the LUT consumes two clock periods, the whole module finishes processing after three clock cycles.
The LUT can be modified through the *w_multicast* signal and the *w_node_id* bus by setting the *w_valid* signal. This writing access on the block memory is independent of the routing process, because the LUT is implemented as a simple dual-port RAM. If a conflict occurs writing has priority and the pulse event being looked up is discarded.

6.2.5. `hbp_extoll_accum`

The purpose of this module is to accumulate pulse events that have to be send to the same target node. It buffers the pulse events after they have been processed by the `hbp_extoll_routing` module. The routed pulse events are delayed 5 clock cycles plus "durable delay" value times 60 ns at maximum. The pulse event format in the accumulation buffer is compatible with the EXTOLL cell format.

6.2.6. `hbp_extoll_rx_ctrl`

This module sends EXTOLL network packets into the data network.

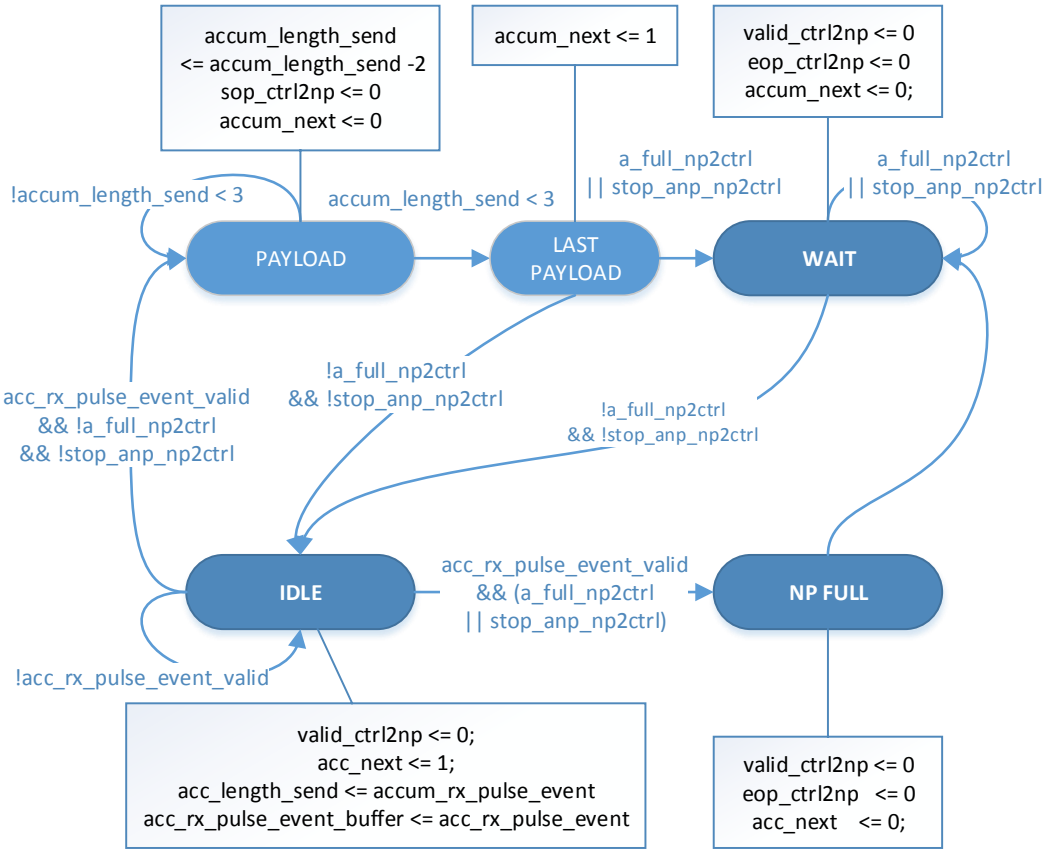


Figure 6.3.: The behavior of the `hbp_extoll_rx_ctrl` module as a state machine.

6.2.7. `hbp_extoll_tx_ctrl`

This module coordinates the work flow in *tx* direction, that is from the EXTOLL network port towards the `hicann_if` module.

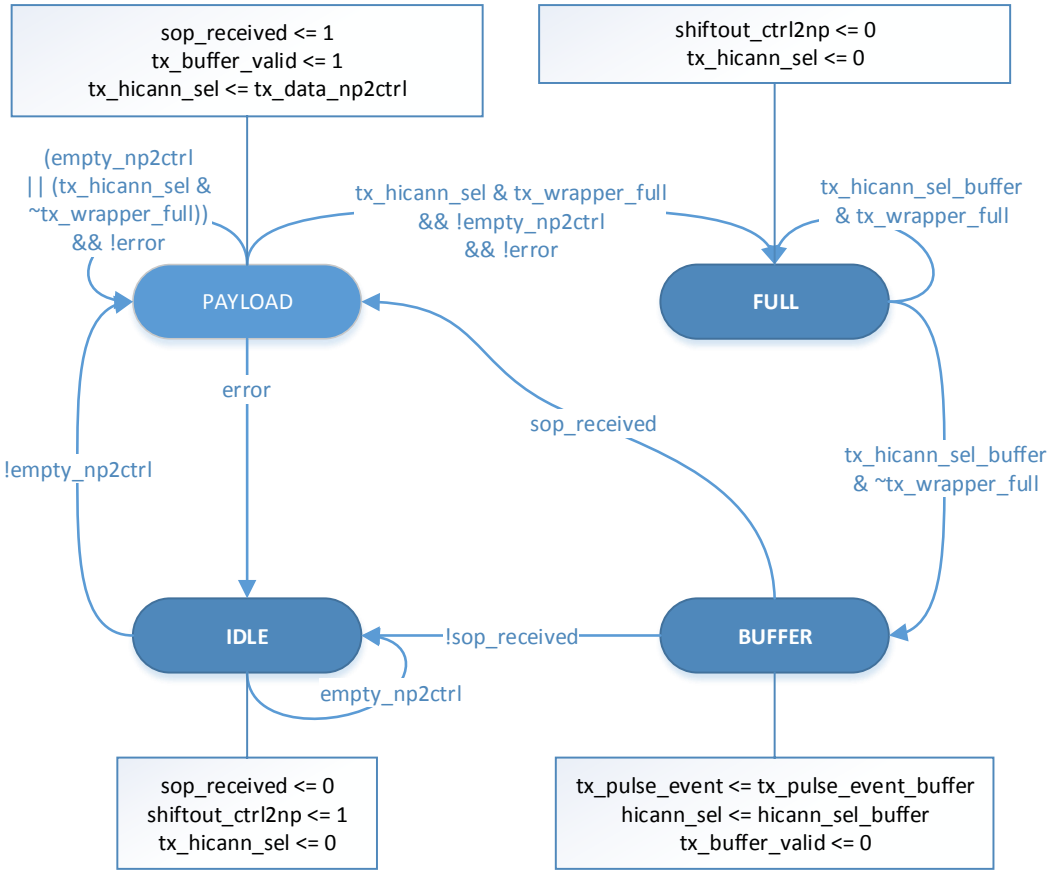


Figure 6.4.: The behavior of the `hbp_extoll_tx_ctrl` module as a state machine.

7. Future Work

The EXTOLL technology was not only found to be applicable for the current state of developement of the neuronal accelerator, but also revealed its potential for future implementations.

7.1. Future Implementations

The current implementation of the HBP EXTOLL Interface could be further developed by

- including a register in the implementation that provides general information, e.g. number of packet errors.
- adding support of configuration packets that could be sent via the RMA unit.
- adding of read-out capabilities for the planned 65 nm wafer version that is envisioned to posses on-wafer ADCs for synchronous analog read-out.

7.2. Envisioned Complications

In the current EXTOLL revision 2 only 16 bit in a SOP control cell are reserved for the node ID. This could be a problem for very large networks (over 1365 wafers without concentrator nodes), as the address space is limited to 65 k nodes. When the NCSs will realize this expansion level, EXTOLL will have progressed as well to revision three or higher, and thus most likely will be able to allow for larger networks by assigning more than 16 bit for the node ID, if an NCS of that size will ever be built based on this technology.

A. List of Abbreviations

ADC	Analog-to-Digital Converter
API	Application Programming Interface
ARQ	Automatic Repeat Request
ASIC	Application-Specific Integrated Circuit
ATOLL	ATOmic Low Latency
AVC	Adaptive Virtual Channel
CMOS	Complementary Metal-Oxide-Semiconductor
CRC	Cyclic Redundancy Check
DDR	Double Data Rate
DVC	Deterministic Virtual Channel
EOP	End Of Packet
EXTOLL	Extended ATOLL
FCP	FPGA Communication PCB
FIFO	First In, First Out buffer structure
FPGA	Field-Programmable Gate Array
FU	Functional Unit
HBP	Human Brain Project
HICANN	High-Input Count Analog Neuronal Network chip
HT	HyperTransport
I²C	Inter-Integrated Circuit
I/O	Input/Output

JTAG	Joint Test Action Group
LUT	LookUp Table
LVDS	Low-Voltage Differential Signaling
mux	multiplexer
MPI	Message Passing Interface
NCS	Neuromorphic Computing System
NIC	Network Interface Card
NM–MC	Neuromorphic Multi-Core
NM–PM	Neuromorphic Physical Model
NP	Network Port
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PPU	Plasticity Processing Unit
RAM	Random Access Memory
RMA	Remote Memory Access
SOP	Start Of Packet
SP9	Sub-Project 9 – Neuromorphic Computing
SpiNNaker	Spiking Neural Network Architecture
TC	Traffic Class
TU	Target Unit
UDP	User Datagram Protocol

B. List of Figures

C. References

[1] *7 Series FPGAs Overview*. DS180 (v1.15). Xilinx. Feb. 2014.

[2] *Brain-inspired multiscale computation in neuromorphic hybrid systems (BrainScaleS) website*. as of 1^{st} August 2014. URL: https://brainscales.kip.uni-heidelberg.de/.

[3] Nicolas Brunel. "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons." In: *Journal of Computational Neuroscience* 8.3 (2000), pp. 183–208.

[4] Niels Burkhardt. *Extoll Network Protocol Specification v2*. internal Document. Computer Architecture Group. University of Heidelberg.

[5] Andrew P. Davison et al. "PyNN: a common interface for neuronal network simulators." In: *Frontiers in Neuroinformatics* 2 (2008).

[6] *direct communication with Dr. Andreas Grübl, Eric Müller and Paul Müller*. involved in SP9 of HBP at University of Heidelberg. July 2014.

[7] *direct communication with Stefan Scholze*. involved in SP9 of HBP at TU-Dresden. July 2014.

[8] *Extoll website*. as of 17^{th} July 2014. URL: http://www.extoll.de/.

[9] Simon Friedmann. "A new approach to learning in neuromorphic hardware." PhD thesis. University of Heidelberg, 2013.

[10] Steve B. Furber et al. "The SpiNNaker Project." In: *Proceedings of the IEEE* 102 (2014), pp. 652–665.

[11] Benjamin Geib. *EXTOLL Network Port Specification*. v2.0. Computer Architecture Group. University of Heidelberg.

[12] Marc-Oliver Gewaltig and Markus Diesmann. "NEST (neural simulation tool)." In: *Scholarpedia* 2.4 (2007), p. 1430.

[13] Stephan Hartmann et al. "Highly integrated packet-based AER communication infrastructure with 3Gevent/s throughput." In: *17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE. 2010, pp. 950–953.

[14] *Human Brain Project website*. as of 27^{th} June 2014. URL: www.humanbrainproject.eu.

[15] Jens Kremkow et al. "Functional consequences of correlated excitatory and inhibitory conductances in cortical networks." In: *Journal of Computational Neuroscience* 28.3 (2010), pp. 579–594.

[16] Christian Leibig. “HBP EXTOLL Networking v2.” Internal technical documentation. Computer Architecture Group at the Institute of Computer Engineering, University of Heidelberg. 2014.

[17] Carsten Mehring et al. “Activity dynamics and propagation of synchronous spiking in locally connected random networks.” In: *Biological Cybernetics* 88.5 (2003), pp. 395–408.

[18] Sebastian Millner et al. “Towards biologically realistic multi-compartment neuron model emulation in analog VLSI.” In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learnin (ESANN)*. 2012.

[19] Mark E. J. Newman. “The structure and function of complex networks.” In: *SIAM Review* 45.2 (2003), pp. 167–256.

[20] Mondrian Nüssle. *Introducing EXTOLL Tourmalet, an HPC Network ASIC*. 13th HLRS/hww Workshop on Scalable Global Parallel File Systems - Self-Managing Data & High Performance Networking Forum (HNF) Europe Spring Meeting. May 2014.

[21] *PyNN API reference: Simulation control*. as of 27^{th} July 2014. URL: http://neuralensemble.org/docs/PyNN/reference/simulationcontrol.html.

[22] Johannes Schemmel et al. “A wafer-scale neuromorphic hardware system for large-scale neural modeling.” In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2010, pp. 1947–1950.

[23] Gordon M. Shepherd. *The synaptic organization of the brain*. Vol. 3. Oxford University Press New York, 2004.

[24] SP9 partners: UHEI, UMAN, CNRS-UNIC, TUD, KTH. “Neuromorphic Platform Specification.” work in progress. 2014.

Declaration

I confirm that I have authored this thesis work independently and that I have not used other than the listed sources and auxiliary means.

Heidelberg, August 6, 2014