# Realizing Biological Spiking Network Models in a Configurable Wafer-Scale Hardware System

Johannes Fieres, Johannes Schemmel, Karlheinz Meier

*Abstract*— **An analog VLSI hardware architecture for the distributed simulation of large-scale spiking neural networks has been developed. Several hundred integrated computing nodes, each hosting up to 512 neurons, will be interconnected and operated on un-cut silicon wafers. The electro-technical aspects and the details of the hardware implementation are covered in a separate contribution to this conference. This paper focuses on the usability of the system by demonstrating that biologically relevant network models can in fact be mapped to this system. Different network configurations are established on the hardware by programmable switch matrices, repeaters, and address decoders. Systematic routing algorithms are presented to map a given network model to the hardware system. Routing is simulated for several network examples, proving the system's practical applicability. Furthermore, the routing simulations are used to fix values for yet open hardware parameters.**

## I. Introduction

ONE of the challenges in simulating large neural networks in a parallel, distributed manner is to ensure sufficient communication bandwidth between the computation nodes. Depending on the neural connection densites and the actual spike rates, communication can in fact constitute the major bottleneck limiting the gain in simulation speed achievable by parallelization [1]. In an accompanying paper [2], a parallel VLSI hardware architecture for the simulation of large-scale pulsed neural networks is presented, being developed within the European Union research project *FACETS* [3]. It features an integrate-and-fire neuron model with current-injecting synapses, and built-in synaptic plasticity, all implemented in mixed analog/digital integrated circuits. The basic elements of the hardware architecture are 10mm$\times$5mm network chips, each implementing 131,072 synapses which can be dynamically partitioned to up to 512 neuron bodies. The neurons operate on a typical time scale which is $10,000$ times faster in comparison to biological real-time, enabling long-term learning experiments or extensive parameter searches. At the same time, space and power consumption are way lower than with conventional computer systems of comparable computing power.

The connectivity problem is approached by wafer-scale integration: The silicon wafers on which approx. 450 chips are produced side-by-side are not cut apart into separate chips but left as a whole. Additional metal layers, deposited onto the wafer in a post-processing step, allow to interface and inter-link the network chips with sufficient connection density and thus to operate large-scale networks consisting of several ten thousand neurons. Larger networks are possible by combining multiple wafer modules.

Two communication protocols are developed especially for this hardware architecture: First, a continuous-time serial bus system using on-chip and post-processing lines for intra-wafer communication ("Layer-1"). Second, a packet-based dynamic routing network implemented by separate custom hardware components which interface the network chips via contacts on the wafer surface ("Layer-2"). Layer-2 is used mainly for inter-wafer communication and for connecting to the control computer.[1]

The communication layers as well as the network chips themselves are configurable as to set up different network topologies, neuron and synapse types, and connection densities. The task faced when employing the hardware system for an actual simulation is to configure its various components to realize the desired network model. Whether or not such a configuration exists, and if yes, how it looks like, is not obvious a priori. The present paper solves this question for network topologies set up using the Layer-1 communication. Model parameters other than the net topology (e.g., electrical neuron and synapse parameters) are not covered here.

The hardware architecture is reviewed in section II. In section III, routing algorithms are outlined for configuring the Layer-1 bus system to realize a given network topology. Section IV presents routing simulations for several example network topologies. The results are also used to fix open parameters of the actual hardware. The paper closes with final remarks and a discussion in section V.

## II. Hardware Architecture

A comprehensive description of the hardware system, including electronic implementation details, is given in the accompanying paper [2]. Here we review the system on a more abstract level from the view points of configuration and operation as described further below.

### A. General Concepts

The network chips, hosting the synapses and neuron bodies, are laid out in a regular grid on the wafer, as shown in Fig. 1. Between the network chips, the buses of the Layer-1 communication system run in horizontal and vertical bundles of 64, respectively 256, single bus lanes. Repeaters at the border between network chips can be configured for each

The authors are with the Kirchhoff Institute for Physics, Ruprecht-Karls University, Heidelberg, Germany. Email: {fieres, schemmel, meierk}@kip.uni-heidelberg.de

This work was funded in parts by the European Union under the grant IST-FET 15879. The authors like to thank Prof. Dr. Schüffny, TU Dresden, for helpful discussions.

[1]The second protocol is developed by the chair of Prof. Schüffny, Technical University, Dresden.



978-1-4244-1821-3/08/$25.00©2008 IEEE

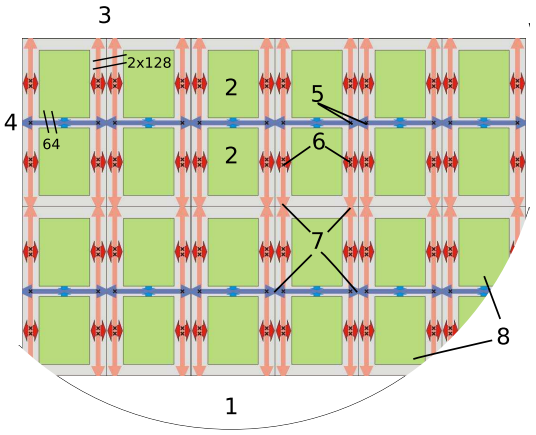


Fig. 1. System overview (not to scale). **1)** Circular silicon wafer; **2)** Network chip (upper and lower half); **3,4)** Vertical and horizontal Layer-1 buses; **5)** Cross bar switches; **6)** Select switches; **7)** Configurable repeaters at chip boundaries; **8)** Incomplete network chips at the wafer border, or defect ones, are not used.

lane separately to relay the signal in either the one or the other direction, or to be off, in which case signal propagation is interrupted at this border. The repeaters and the crossbars which connect the horizontal and vertical buses allow to route signals across the wafer.

Each bus lane carries the spike signals of up to 64 neurons. A spike is indicated by an asynchronous serial 6-bit packet encoding the address of the source neuron. The spike time is inferred from the actual moment the packet is received. The network chips insert generated spikes into the horizontal bus crossing at their center. If a chip implements 64 neurons or less, it fills one bus lane, else, for each new group of 64 another lane is filled. Each group of neurons firing to the same lane is arbitrated by a priority encoder: if two neurons fire at the same time, the one with the lower priority is delayed, or it is discarded if the buffer is full. Care must be taken by the routing logic that no more than one network chip is sending on the same lane.[2]

At the borders between network chips, the bus lanes are shifted by one as shown in Fig. 2. The twisting allows for network chips to insert signals to the same local horizontal lane and to use the same cross bar configurations while largely avoiding global signal collisions. As a consequence, the point of signal insertion does not need to be configurable, but it is done for all network chips according to the same scheme: Chips implementing up to 64 neurons insert their output signals into horizontal lane no. 0. This way, only after a horizontal routing distance of 64 network chips, two chips will write to the logically same lane (which is in fact way longer than the maximum distance within one wafer). If a chip has 64 up to 128 neurons, lanes 0 and 32 are used, which allows for a horizontal collision-free routing distance

[2]A technical solution for the same lane being filled by multiple network chips has been provisioned, but it will not be implemented in the first prototype system.

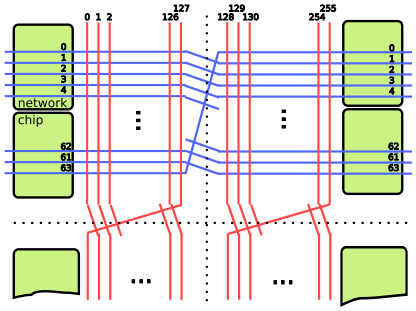


Fig. 2. At chip boundaries, the bus order is shifted by one to facilitate routing. The 256 vertical bus lanes are treated in two groups of 128 in order to avoid having wires twisting across the chip border.

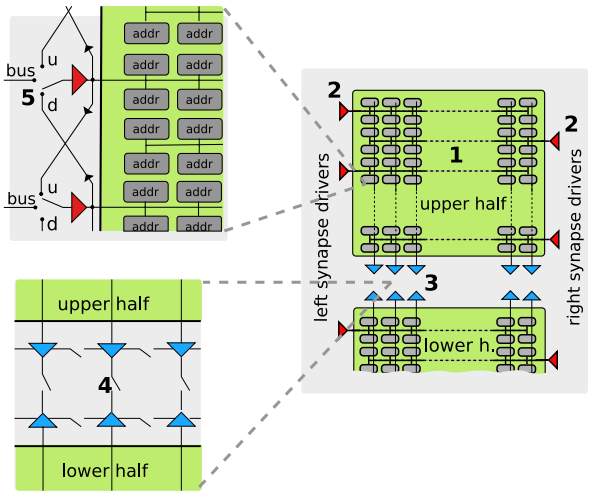


Fig. 3. Schematic of the network chip. **1)** Synapse array; **2)** Synapse drivers; **3)** Neuron bodies; **4)** Neuron builder switches; **5)** Synapse driver mirror switches.

of still 32 chips (after 32 steps, lane no. 0 will run on the 32th lane and vice versa). In general, with more neurons per network chip, the order by which the bus lanes are filled is 0, 32, 16, 48, 8, 24, 40, 56, keeping the routing distance always at the maximum possible value. More than eight lanes are never filled since the maximum number of neurons a chip can implement is $512 (= 8 \cdot 64)$.

### *B. Network chips*

The 131,072 synapses of a network chip are laid out in two regular arrays of 256×256 synapses each, referred to as the upper and the lower half, shown in Fig. 3. 512 neuron body circuits reside in the area between the two halves, beneath the horizontal bus lanes. All 256 synapses of an array column belong to the same neuron body. This constitutes the minimum configuration of 256 synapses per neuron. Using the configurable neuron builder switches (Fig. 3, item 4), the membrane potentials of adjacent neurons can be shortcut, effectively forming neurons with a multiple of 256 inputs. The maximum reasonable number of synapses per neuron is

16,384 since this is the maximum number of different input signals possibly being fed into the network chip.

The synapses receive their inputs from circuits located to the left and to the right of the synapse matrix, called *synapse drivers* (Fig. 3, items 2). A synapse driver is connected to a vertical bus lane via a select switch matrix and relays the signal of this lane to two adjacent rows of synapses.[3] The synapses are equipped with individually programmable address decoders which make them selectively attentive to one of the 64 presynaptic neurons.

Since a synapse driver operates two matrix rows, each neuron can chose two of the 64 addresses of a bus lane (assuming no combined neuron bodies). If a neuron is to receive more than two addresses from a given bus lane, one can either combine $N$ array columns to one neuron, which would allow to choose $2N$ neuron signals. This method can be a waste of hardware synapses. As a more efficient alternative, one could connect the same lane to multiple synapse drivers. However, a direct feeding of a bus lane to many synapse drivers via the select switch is electrically difficult because of signal attenuation. As a technical substitution, a synapse driver can mirror the signal of its upper or lower neighbor instead of relaying the direct signal from the select switch. This is indicated by the switch positions $u$ and $d$ in Fig. 3, item 5. With a chain of $M$ mirrored synapse drivers, each synapse column can select $2M$ addresses from the respective bus signal.

## III. Routing Algorithms

In this paper we assume that a particular hardware-independent neural network description (the "model") is given in terms of a list of neurons and a list of individual connections between them. (However, this is not the only possible starting point for setting up hardware simulations; see Discussion.) The mapping of a model to the hardware comprises two steps: placing and routing. The former is concerned with assigning neurons to network chips and deciding how many hardware synapses are reserved for each neuron. The latter, which is the focus of this paper, addresses the realization of the synaptic connections. Surely, a clever placing strategy can facilitate routing. In the experiments reported below, however, placing is done trivially by assigning groups of neurons to network chips one by one in the order they appear in the model description. More elaborate placing approaches and a higher-level view of the entire mapping process are developed by project partners [4].

Routing a network via the Layer-1 bus system comprises the configuration of the repeaters, the cross bars, the select switches, the synapse driver mirror switches, and the synapse address decoders. The task is performed in two stages from coarse (A) to fine (B):

### A. Inter-Chip Routing

The first, coarse, routing stage encompasses the chip-to-chip routing via the bus repeaters and the cross bars. In this stage it is ensured that, for each network chip, every signal required by at least one of its neurons is available on one of the vertical bus bundles left or right adjacent to it. The general routing strategy adopted is to use the horizontal buses as "backbones" from which the target chips are accessed via the vertical buses (see Fig. 4). Note that, since network chips can access bus lanes running on either side, only one lane per two vertical bus bundles needs to carry the same signal.

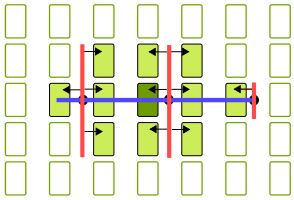

Fig. 4. Routing strategy for connectivity on chip-to-chip level. Dark: source chip; Light: target chips.

This routing strategy is both simple to implement and effective: Each chip-to-chip connection is optimal in terms of crossed chip borders, and a maximum of one crossbar switch must be passed. Moreover, in case of the targets constituting a convex, simply connected region (as in the example in the figure), the available bus lanes are used very efficiently. Also, the choice for having more vertical than horizontal lanes becomes apparent. Each signal occupies only one horizontal lane but many vertical ones. In the examples shown later, the condition of convex connection areas is always met since the networks are either fully connected (every chip is connected to every other), or a local connection scheme can be established by arranging the chips appropriately. However, for other topologies, different routing strategies might be more suited.

The cross bars by which the horizontal and vertical buses are connected are sparse switch matrices of the type shown in Fig. 5. Making all $64\times256$ possible junctions switchable would consume a lot of of chip area, and also, from an electrical point of view, each additional switch (even if it is not closed) adds extra impedance to the wires which complicates signal transmission. A tradeoff between hardware design effort and routing flexibility must be made. Reasonable values of the parameters $S$ and $T$ are determined in the routing simulations.

For the first routing stage, a simple greedy implementation proves to produce good results and still leaves resources available for detouring defect network chips. For each source chip, the convex hull spanned by the target chips is supplied with its signal according to the scheme in Fig. 5. In each step, used bus segments are marked as used. If, at any point, a target cannot be reached because of a lack of unused bus segments or available cross bar switches, the affected synapses are discarded as not implementable.

### B. Intra-Chip Routing

The second routing stage which is a little more complex is concerned with routing the signals from the Layer-1 bus to

[3]Technically, the synapse driver amplifies and decodes the serial packets and provides a combination of parallel address and strobe signals. See [2].

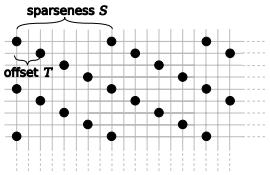


Fig. 5. Sparse switch matrix connecting two bundles of wires. The layout is defined by the parameters $S$ and $T$. Only every $S^{\text{th}}$ junction is switchable. The cross bars and the select switches (Fig. 1, items 5 and 6) are of this type. $T = 1$ is chosen for the select switches. At first glance $T > 1$ seems to restrict connectivity, since some vertical wires cannot make a connection at all, but in fact $T = 4$ seem to be optimal for the crossbars (section IV-A). The $S$ for both switch matrices are left as parameters fixed in the simulations (section IV).

the synapses via the the select switches, the synapse driver mirrors, and address decoders in the synapses.

*1) Preconditions:* The 512 vertical bus lanes which a network chip can access run in two bundles of 256 to the right and to the left. Thus, half of the synapse drivers can access lanes 0...255 only, the other half lanes 256...511.[4] For the sake of simplicity, it is assumed in the routing simulations reported below that the bus lanes a network chip needs to access are distributed equally across the two bundles, such that 50% of the input signals are on the left and 50% on the right. This distribution can be quite well approximated by the first routing stage by using a random scheme in occupying the bus lanes. The 50-50 ratio is ideal in terms of utilization of the synapse drivers; in contrast, in the worst case which is a 100-0 ratio, effectively only the synapse drivers of one side can be used.

It is further assumed that the number of synapses per neuron is constant within a network chip, and that this number can be written as $256 \cdot 2^K$, with the combine factor $K = 0, 1, \ldots$. The combination scheme is as follows: For $K = 0$, no neuron bodies are combined. For $K = 1$, each two opposite neuron bodies from the upper and lower synapse array are combined. For $K > 1$, horizontally adjacent neuron bodies are combined in addition to the opposite. Combining opposite neurons first maximizes the number of synapse drivers innervating a neuron. This increases the system's routing flexibility.

Having all neurons in a network chip of equal size is not a general restriction of the routing algorithms but it makes their implementation simpler. Moreover, this condition is easily met when placing a network on the hardware resources by choosing $K$ appropriate for the largest neuron of the chip. For the smaller neurons, the weights of superfluous hardware synapses are simply programmed to zero. Of course, ease of implementation must be paid for by a sub-optimal hardware utilization. In principle, the routing algorithms can be expanded for irregular neuron builder configurations.

*2) Layer-1 to Synapse Drivers:* The Layer-1 buses are connected to the synapse drivers via the select switches. The synapse drivers play the role of a limited resource here, so one can picture the bus lanes to "compete" for connections to the synapse drivers. Since the synaptic address decoders are programmable, all synapse drivers are largely equivalent. Therefore, it is initially most important to specify *how many* synapse drivers each Layer-1 lane should connect to, but not yet *which of them*. So, for each lane $i$, the following algorithm will determine the desired number of synapse drivers $A_d(i)$.[5] The goal is to minimize the synapse loss which measures the proportion of model synapses not realized on the hardware. The procedure is done separately for each of the two vertical lane bundles 0...255 and 256...511.

1. Count how many synapses in the chip require a connection from each of the 256 lanes (this can be done without having yet assigned the synapses to array positions). The result is a histogram $S(i)$ with $i = 0...255$.
2. Assign the 128 (resp 64, if $K = 0$) synapse drivers to the lanes, such that the number of synapse drivers for each lane is proportional to the respective histogram entry, thus $A_d(i) := cS(i)$ for all $i$ with a constant $c$. As a special rule, each lane whose histogram entry is larger than 0 receives at least one synapse driver: $A_d(i) \geq^! 1$ for $S(i) > 0$. As another special rule, a lane gets at most as many synapse drivers as are needed to feed all 64 signals of that lane to the neurons.
3. In step 2, due to rounding errors and the first special rule, more synapse drivers can be assigned than actually exist. Therefore, the $A_d(i)$ are decremented until the total number amounts to $\sum_i A_d(i) = 128$ (resp. $= 64$ for $K = 1$). Specifically, the $A_d(i)$ with the greatest ratio $A_d(i)/S(i)$ is iteratively chosen, and decreased by 1.
4. In step 2, due to rounding errors and the second special rule, less synapse drivers can be assigned than actually exist. Therefore, the $A_d(i)$ are increased until either $\sum_i A_d(i) = 128$ (64) or each bus lane has enough synapse drivers to feed all its 64 signals to the neurons.

Due to the limited number of synapse drivers and for reasons explained in paragraph 3) below, some connections might be not routable. The corresponding model synapses are lost. In this case, the described algorithm tends to discard connections from lanes of which only a small number of signals are needed, for this results in the minimum absolute synapse loss. This corresponds to setting sparse regions of the network's connection matrix to zero. Sometimes, this might be not desired, for example when two sub-populations of neurons with a dense connectivity within themselves are interconnected by only a few but functionally essential synapses. Therefore the routing algorithm can optionally ensure that $A_d(i) \geq 1$ for each $i$ with $S(i) > 0$. For this, in step 3, if the bus lane with the greatest ratio $A_d(i)/S(i)$ already has only one synapse driver, instead the lane with the largest absolute value of $A_d(i)$ is chosen for decrement. This approach preserves sparse regions of the connection matrix

[4]Lanes 0-127 run on the left-neighboring chip, the lanes 384-511 on the right-neighboring chip. However, since the select switch matrices extend across the borders, this detail is transparent for the routing process

[5]Due to hardware restrictions, possibly not all of the desired connection might be realized later.

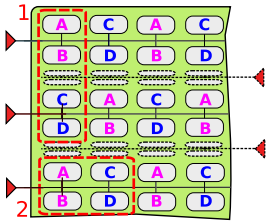


Fig. 6. The two fixed bits of the address decoders can be pre-set in an efficient checkerboard pattern. A=00, B=01, C=10, D=11. Both, the mirroring of two synapse drivers (1), as well as the combination of two neuron bodies (2) covers the entire range of possible addresses. Accordingly for the synapses driven from the right (small, dotted).

at expenses of losing synapses in densely populated areas and usually a larger total number of lost synapses. Note that this routing option requires that the total number of lanes carrying input signals for the network chip does not exceed the number of available synapse drivers. This requirement is generally met for networks of less than 16,384 neurons and for larger networks of certain topology (c.f., section IV-C).

Finally, it is examined whether there is an actual configuration of the select switch and the mirror switches which realize the desired $A_d(i)$. The select switch is a sparse switch matrix according to Fig. 5 with $T = 1$ and $S$ left as a parameter to be determined in the routing simulations. Moreover, as said before, only one synapse driver can be connected directly to a given bus lane. All other synapse drivers which are to relay the same signal must be adjacent to the first one. The faced problem is equivalent to disk defragmentation where the synapse drivers are a one-dimensional array of storage bins, the lane indices $i$ are the file labels, and the $A_d(i)$ are the file sizes. Once more, a greedy approach produces satisfactory results. The algorithm takes the $A_d(i)$ as input and outputs the realized number of synapse drivers $A_r(i)$ with $A_r(i) \leq A_d(i)$ for each $i$:

Each of the 128 synapse drivers of one side is marked as either "free" or "occupied". At the beginning, all drivers are free. The lanes $i = 0...255$ are successively processed ordered by their $A_d(i)$, starting with the highest value. In the disc fragmentation view this corresponds to placing the largest files first. Due to the sparse select switch matrix, each lane can be switched only to a small subset of synapse drivers. For each possible switch point it is examined whether there is a connected block of free synapse drivers of length $A_d(i)$. If yes, the search stops and $A_r(i)$ is set equal to $A_d(i)$. If there is more than one way to place the new block of occupied syndrivers, it is placed to touch an already occupied driver, if possible. This is to leave no fragmented small free areas which can possibly not be used later any more. If none of the possible positions exhibits a sufficient number of free synapse drivers, the search is repeated with $A_d(i) - 1$, $A_d(i) - 2$, ...,0, until enough free drivers are found. $A_r(i)$ is then set to the actual number of connected synapse drivers.

*3) Programming the address decoders:* After the bus signals have been routed into the synapse array via the select switches and the synapse drivers, the task left to do is to program the synapses' address decoders. For minimizing the chip area consumed per synapse it was chosen to make only 4 of the 6 address bits freely programmable and to have the other two pre-determined by a global scheme (details in [2]). So, a given synapse can choose only from a range of 16 of the 64 addresses. The architecture allows to assign the two pre-set bits according to the checkerboard pattern shown in Fig. 6, which has the nice property that any combination of four synapses covers the entire address range. The $2^2 = 4$ possible values of two bits are labeled A–D. The checkerboard scheme makes two vertically or horizontally neighboring synapses cover a different 16-range of addresses. So either by feeding a bus lane into two synapse drivers or by combining two neighboring neuron bodies, a neuron can have access to all 64 signals.

In the routing simulations presented below it is first determined for each lane $i$ how many synapses $s(i)$ of the same neuron it is connected to. This value is $s(i) = m * A_r(i)$, where $m = 2, 4, 6, ...$, depending on how many matrix columns are combined to one neuron body. So, a neuron can select a total of $s(i)$ signals from bus lane $i$, however not arbitrarily, because each synapse can address only a range of 16 of the 64 signals. In the routing simulation it is assumed that different synapses of a neuron can reach different 16-ranges. More specifically: $s(i)$ is split into a part divisible by 4 and into the division remainder. The synapses in the first part are distributed evenly on the four possible 16-ranges, for the remainder, 16-ranges are assigned by random. For the remainder, this approach does not exactly reflect the address pattern given in Fig. 6. However, the routing simulation is only to estimate the expected routing quality (in terms of synapse loss), and since the connections in the biological network models are usually set up with randomness, statistically the correct results are produced.

## IV. Experimental Results

It was evaluated for several network types how well they can be mapped to the hardware system. The simulations were done, firstly, in order to prove the practicability of the system, and secondly, in order to figure out sensible values for open hardware parameters, in particular the layout of the sparse switch matrices.

As a rough measure of the routing quality we take the fraction of connections in the original network model which are actually realized on the hardware system:

$$\text{routing quality} = \frac{\text{\# realized synapses}}{\text{\# model synapses}}. \quad (1)$$

Accordingly, the complement of this value (1 – routing quality) is the relative *synapse loss*. Like above, "model network" refers to a specific network description (e.g., a list of neurons and their interconnections) which is to be mapped onto the hardware. The "realized network", or, "routed network", is the network actually represented in the configured hardware. Generally, the number of synapses (connections) in the realized network is less than or equal

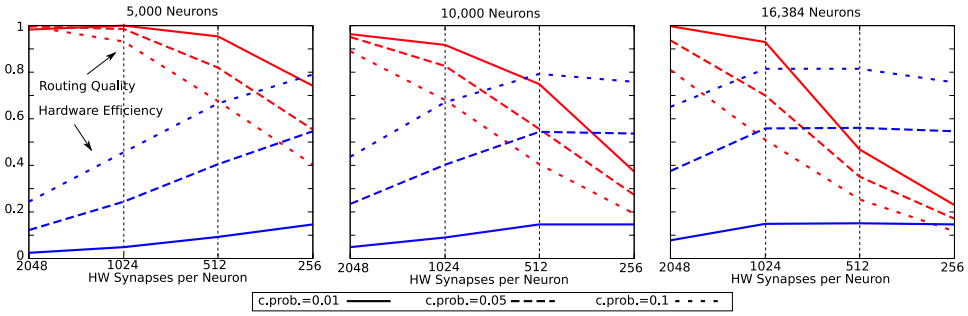


Fig. 7. Homogeneously connected networks with 5,000, 10,000, and 16,384 neurons and connection probabilities (c.prob.) of 1%, 5%, and 10%, routed with 2048, 1024, 512, and 256 hardware synapses per neuron. Upper curves: Routing Quality (percentage of model synapses realized); Lower curves: Hardware Efficiency (percentage of hardware synapses utilized). By reserving more or less hardware synapses per neuron it is possible to trade off routing fidelity for efficient hardware usage. When routing other instances of the same random models, the curves fluctuate by less than 0.002 on the shown y-scale.

to the respective number of the model network due to the hardware restrictions.

Another objective aside from a high routing quality is to use hardware resources efficiently. Usually, parts of the hardware components (synapses, bus lanes, neuron bodies) remain un-used due to architectural constraints. For example, a network with 2,560 neurons and a connection probability of 10% cannot generally routed with a configuration of only 2,560/10=256 hardware synapses per neuron. The fraction of all hardware synapses which actually implement a connection is regarded as a good measure since the synapse circuits contribute most notably to resource consumption.

$$\text{hardware efficiency} = \frac{\text{\# realized synapses}}{\text{\# hardware synapses}}.$$

Usually only a subset of all network chips on the wafer are used for a given setup. Only the synapses of those chips do contribute to the computation of the hardware efficiency. The other network chips can be switched off and thus do not consume any power.

### A. Homogeneous Random Networks

The basic characteristics of the system are explored with a simple class of neural networks where each ordered pair of neurons is connected with a given probability. Assuming a connection probability of 1%–15% in typical biologically realistic setups [6] and having 64–512 neurons per network chip, it is very likely that for two arbitrarily chosen network chips A and B, at least one neuron from A receives input from at least one neuron from B. For large networks, as a result of the statistical law of large numbers, it is further not possible to find a re-mapping of neurons to chips which cancels this fact. Therefore an all-to-all connection between network chips must be established for such networks. For the same reason, the maximum number of neurons in a homogeneous network is bound to 16,384 in the current architecture: a network chip needs to receive every signal in the network, and the maximum number of signals it can be fed with is limited by the 256 synapse drivers each carrying 64 signals (256x64=16,384). For four neuron builder configurations (256, 512, 1024, and 2048 hardware synapses per neuron, corresponding to 512, 256, 128, and 64 neurons per chip), the first routing stage succeeds up to a crossbar sparseness of $S = 32$ (see Fig. 5), which is feasible from an electronic design view. A crossbar offset of $T = 4$, although preventing some of the vertical bus lanes from being reached at all, allows for this maximum sparseness. The lack of the number of connectible vertical lanes seems to be outperformed by the gain of connections per remaining vertical lane. Also, due to the shifting of bus lanes at chip borders (Fig. 2) missed vertical lanes can be used by signals originating at other grid positions on the wafer.

Except for the configuration with 512 neurons per chip, less than 50% of vertical and horizontal bus lanes and less than 10% of all cross bar switches are used, leaving enough resources to detour defect chips on the wafer. With 512 neurons per chip, 100% of the vertical and 50% of the horizontal lanes are used; however, this configuration would probably not be used frequently because it features only 256 input synapses per neuron and only half of a chip's synapse drivers per neuron.

The second routing stage is simulated for a single network chip only since, due to the homogeneous network structure, the task is statistically the same for the other ones. Fig. 7 shows the results for three network sizes and three connection densities. The parameter varied on the x-asis is the number of hardware synapses reserved for each neuron. Assigning more hardware synapses to each neuron obviously increases the number of mapped model synapses by making routing easier, but on the other hand, more hardware synapses remain un-used, decreasing the hardware efficiency. With the two larger networks, the hardware efficiency saturates at some point even if less hardware synapses are assigned per neuron. This is mainly because a certain fraction of synapses has access to the "wrong" 16-ranges of addresses (prescribed by the two fixed address bits) and can thus not be used. This problem vanishes for the small networks, because signals can be fed in by multiple synapse drivers, covering enough synapses with the "correct" 16-ranges. Another reason for the efficiency saturation and also for the bad routing quality with 256 hardware synapses per neuron is the fact that with

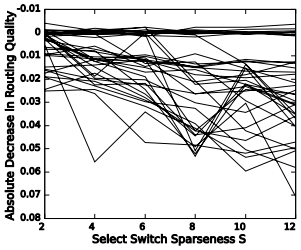


Fig. 8. Decrease of routing quality with sparser select switch for the setups from Fig. 7.

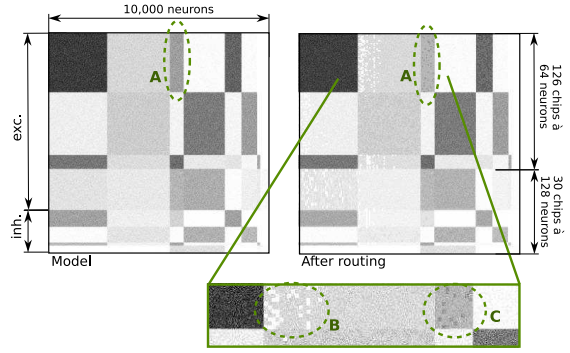


Fig. 9. Routing a model of the Cortical Column with 10,000 neurons as used by project partners [6]. Connection matrix of model (left) and after routing (right). For the specific placing (numbers on the right), 93.6% of all model connections are realized at a hardware efficiency of 38.0%. A, B, C: Visible symptoms of routing difficulties (details in paper text).

this configuration, each neuron spans only one half of the network block and thus has access only to half of the synapse drivers.

*Sparseness of the Select Switch Matrix:* The sparseness parameter $S$ of the select switch matrix was not fixed above. Routing experiments are conducted to determine a sensible value. All experiments from Fig. 7 are repeated with $S \in \{1, 2, 4, 6, 8, 10, 12\}$. For each network it is determined how much, in absolute values, the routing quality differs from the run with $S = 1$ for higher values of $S$. Fig. 8 shows that the decrease of routing quality is in fact moderate. The curves are not generally monotonically decreasing since values of $S$ not dividable by the total height or width of the switch matrix result in an uneven distribution of switches over the bus lanes. The curves staying around 0 over the entire range of $S$ belong to the networks with 16,384 neurons. With 16,384 signals to be fed into each network chip, the synapse driver multiplicities $A_d(i)$ are equal to 1 for all $i$. These simple multiplicities can be realized well almost independent of $S$.

For the final hardware specification, $S = 6$ was chosen as a good compromise. It is reasonable from a hardware designer's point of view while decreasing the routing quality by less than 3 percent points in most cases. The simulations from Fig. 7 and 9 were done with $S = 6$.

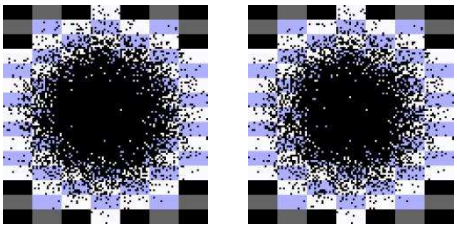


Fig. 10. Locally connected network with Gaussian connection profile routed with 128 neurons/chip. Each pixel corresponds to a neuron. Checkerboard cells mark groups of 16x8 neurons hosted by the same network chip. Neurons sending signals to at least one neuron on the center network chip are marked black. The routing quality is 0.86 at a hardware efficiency of 0.39. Routing with 64 neurons/chip yields 0.99 (routing quality) and 0.22 (hardware efficiency).

### B. Model of the Cortical Column

Routing was simulated for a model of the cortical column [5] as recently used in biologically relevant work by project partners [6]. In the version used here, the model consists of 10,000 neurons (8,014 excitatory, 1,986 inhibitory). Fig. 9 shows the connection matrix of the model (left) and of the routed network (right). More blackness indicates a higher connection density. The gray values are scaled up for better visual contrast. The actual densities are roughly 12 times lower, corresponding to an average connection probability of approx. 7%. The network is placed on 156 network chips, where neurons with more inputs have been reserved more hardware synapses (see labels right of routed matrix). The overall routing quality for the chosen placement is 93.6% with a hardware efficiency of 38.0%. By reserving more hardware synapses per neuron, the balance can be shifted towards higher routing quality at the expense of lower hardware efficiency.

Three symptoms of routing difficulties can be illustrated in this example. The first one (labels A in figure) is an overall decrease in connection density. The indicated section of the routed connection matrix is slightly brighter than in the model. Secondly (B, enlarged view), small patches of completely erased synapses can be seen. This is because the routing algorithm fails to reach some synapse drivers due to the sparse switching scheme in the select switch matrix. Finally (C), we see some small patches being darker than the surrounding. Here, most patches are assigned only one synapse driver which is in fact not sufficient to route all the model connections (this is also the main reason for symptom A). A few extra synapse drivers, however, are still available. They are assigned to subsets of the matrix which thus appear as the darker spots.

### C. Locally Connected Networks

As argued before, network models in which each chip needs input from every other chip are practically limited to 16,384 neurons. One class of networks where this restriction does not apply are locally connected networks. Here, the neurons are arranged in a 2-dimensional sheet and the connection

probability between two neurons is expressed in terms of their distance (many models of the cortex are like this, e.g., [7]). If the probability goes to zero fast enough for large distances (e.g., a Gaussian connection profile), the average number of connections a neuron receives is independent of the total network size. If further the area around a neuron it receives connections from does not include more than 16,384 neurons, such networks can be implemented on the hardware, with the total number of neurons only restricted by the size of a wafer, respectively the number of wafers which can be interlinked with sufficient bandwidth.

In the following simulations, an infinite grid of both model neurons and network chips is assumed. Rectangular regions of neurons are mapped to one chip. e.g., in a configuration with 128 neurons per chip, a patch of 16x8 neurons in the model is mapped to one chip. Fig. 10 (left) shows a section of the neural grid, the partition into network chips indicated by a checkerboard pattern. The first routing stage is conducted for a square grid twice as large as the section shown in the figure. This routing stage ensures that each network chip can receive signals from all chips within a local circular region centered around itself consisting of 85 chips (=10,990 neurons). For the chip shown in the center of the figure this circular region is marked with a brighter background. Each neuron is randomly assigned 500 input connections from a Gaussian distribution of 17 neurons width. The chosen inputs for all 128 neurons of the center chip are marked black. On average, two connections per neuron lie outside the bright region and must be discarded or realized by Layer-2 communication. In the right (routed) picture, some connections from the dense region of the model are missing. The second routing stage has been done with preserving regions of sparse connectivity (see section III-B.2), otherwise the algorithm would have favored the dense region in expense of sacrificing the few long-distance connections. The routing quality in the shown example is 0.86 and the hardware efficiency 0.39. When the routing is repeated with a configuration of 64 neurons/chip the values change to 0.99 (routing quality) and 0.22 (hardware efficiency).

## V. Discussion

The practicability of the hardware architecture [2] is plausibly demonstrated by providing routing algorithms and evaluating them for realistic network models. Moreover, sparse layouts of the select switch matrices and cross bars are determined which are feasible from both the engineering and usability point of view.

The high speed, small form factor, and low power consumption compared to numerical simulations with digital super-computers must be paid off by certain restrictions in flexibility. When mapping given neural models to the hardware, these restrictions translate to losing some of the inter-neuron connections, typically only a few percent in the covered examples. It is not entirely obvious what impact such small "brain-damages" will have on neuro-computational results, but the authors claim that certainly some relevant experiments will be possible. Moreover, to some extent, the routing algorithms allow to trade off efficient hardware usage for a better mapping fidelity. It should be noted here that all examined networks are instances generated from probabilistic models. Instead of chosing a specific instance and mapping it synapse-by-synapse to the hardware, one could also include the hardware specification in the generation process and produce only model instances which are 100% routable. Whether or not such an approach affects the statistical outcomes of the experiments remains to be evaluated. In order for the hardware to get accepted by experimentors in the first place the former approach has been chosen as a start.

The hardware efficiency is around 40% in the reported experiments. Of course, networks with 100% hardware utilization can be set up, but, again, only for model instances which are designed especially for the hardware system. Since inactive synapses do not consume any power, a low hardware efficiency does not constitue a major issue except for more network chips may be necessary to map a given network.

Critical factors limiting the routing capabilities have been identified in the experiments: It's not the number of available bus lanes but rather the limited number of signals which can be fed into one network chip and the design choice of only four programmable bits in the synapse addresses. These findings can be incorporated in the design of further hardware versions, e.g., by providing more synapse drivers per chip, and by making the full address space programmable in the synapses. However, the general idea, to route continuous-time serial address signals over a multi-lane bus system, thus combining temporal and spacial multiplexing, proves being correct.

Due to typical manufacturing characteristics, some dies or parts of them will be defect. When dealing with separate dies, affected chips are normally sorted out and discarded, only lowering the production yield. In a wafer-scale system the defective components remain on the wafer and have to be dealt with. It has been argued in this paper that presumably enough resources are available to detour signals around defect cells. Actual concepts will be developed and practicability remains to be proven in further work.

## References


[1] M. Migliore, C. Cannia, W. W. Lytton, H. Markram, M.L. Hines: Parallel network simulations with NEURON. J. of Computational Neuroscience **21** 119-129 (2006)

[2] J. Schemmel, J. Fieres, K. Meier: Wafer-Scale Integration of Analog Neural Networks. Proceedings IJCNN 2008, IEEE Press (2008) (accepted)

[3] FACETS: Fast Analog Computation with Emergent Transient States. E.U. grant no. IST-FETPI 15879. http://www.facets-project.org

[4] K. Wendt, M. Ehrlich, C. Mayr, R. Schüffny: Abbildung komplexer, pulsierender, neuronaler Netzwerke auf spezielle Neuronale VLSI Hardware. DASS07, Dresden, 127 - 132 (2007)

[5] T. Binzegger, R. J. Douglas, K. A. C. Martin: A Quantitative Map of the Circuit of Cat Primary Visual Cortex. J. of Neuroscience **24**(39) 8441–8453 (2004)

[6] J. Kremkow, A. Kumar, S. Rotter, A. Aertsen: Emergence of population synchrony in a layered network of the cat visual cortex. Neurocomputing **70**(10-12) 2069–2073, (2007)

[7] L. Tao, M. Shelley, D. McLaughlin, R. Shapley: An Egalitarian Network Model for the Emergence of Simple and Complex Cells in Visual Cortex. PNAS **101** 366-371 (2004)