A formal model of FPGA implemented artificial neural networks

Brno University of Technology
Faculty of Information Technology
Božetěchova 2, 612 66 Brno
Martin Krčma, ikrcma@fit.vutbr.cz

18. 12. 2014

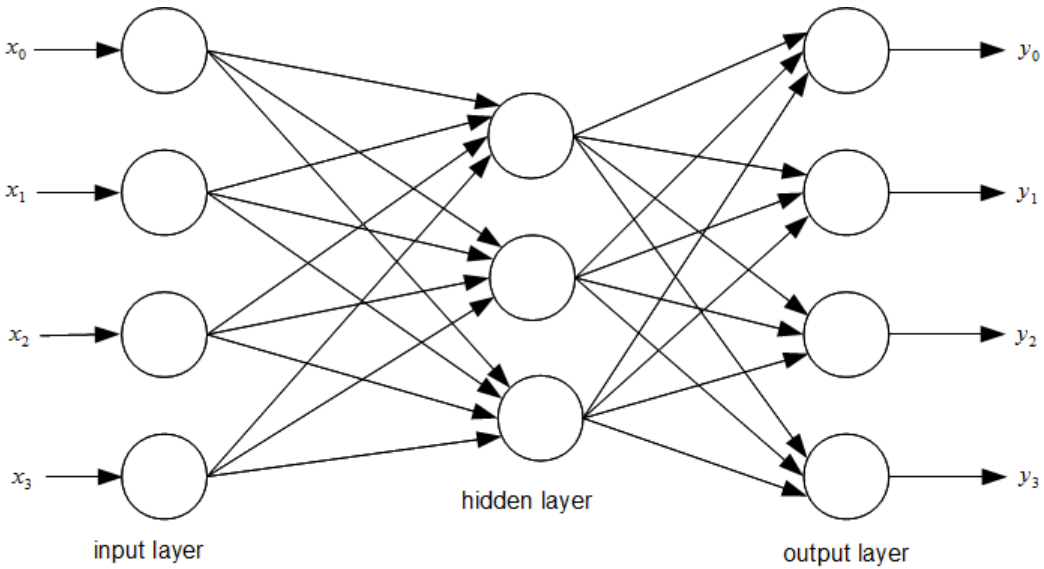
Outline

- Introduction
  - Introduction of artificial neural networks.
  - Introduction of FPGAs.
- Neural networks in FPGAs
  - Definition of FPNA.
  - Definition of FPNN.
  - Definition of grid FPNN.
  - Grid FPNN mapping algorithm for redundancy free fault tolerance.
- Summary

Introduction: artificial neural networks

- Computational model of the human brain
- History
  - introduced in 1943 (Warren McCulloch, Walter Pitts)
  - first conference in 1987

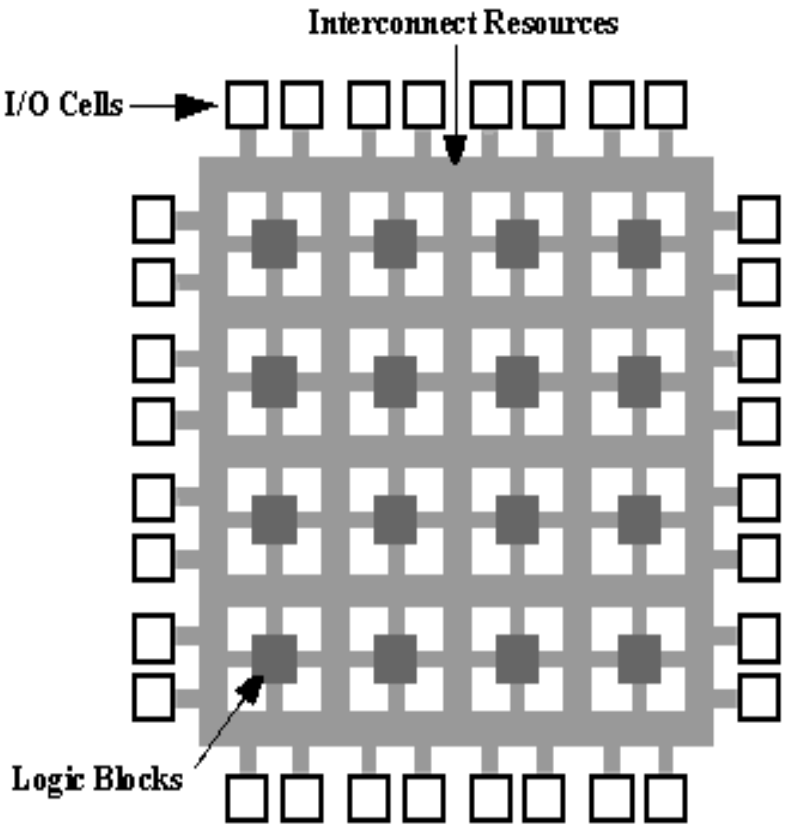
- Usage
  - classification
  - prediction
  - approximation
  - control
  - clustering



Introduction: FPGA

- Field programmable gate array.
- Digital cuircuits repeatedly programmable on low level (hardware level).
- Capable of implementing almost every type of digital cuircuits.
  - CPU, GPU, DSPs, filters, controllers...
- Capable of parallel computing.
- Much faster than general purpose processors in parallel task designs.
- Not as fast as ASIC, but flexible, repetively usable.
- Great development and experimental tool.

Introduction: FPGA



FPNA

- Field programmable neural array.
- Introduced by Bernhard Girau in his master thesis.
- FPGA implementation of neural networks.
- Can be also a resources saving approximation.
- Relaxed definition.
- Strongly optional structure.

FPNA:definition

- Let N be a set of nodes (activators) and E a set of oriented edges (links). We say that graph (N,E) is an FPNA if the following statements hold:
  - Every node has a set of predecessors:

$$\forall n \in N : \exists Pred(n) = \{p \in N | (p,n) \in E\}$$

  - Every node has a set of successors:

$$\forall n \in N : \exists \mathrm{Succ}(n) = \{s \in N | (n,s) \in E\}$$

Notes:

- The $e \in N \cup E$ is called neural resource
- The links are named as (x,y) where $x, y \in N$

FPNA:definition

- There is a set of input nodes:

$$\exists N_i = \{ n \in N \mid Pred(n) = \emptyset \}$$

- Every link has affine operators:

$$\forall (r,n) \in E \wedge p \in Pred(n) : \exists \alpha_{(p,n)} = W_n(p) \cdot x + T_n(p)$$

- Every non-input activator has an iterative operator for potential computing:

$$\forall n \in N : \exists i_n : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

- Every non-input activator has a function operator for activation function computing:

$$\forall n \in N : \exists f_n : \mathbb{R} \rightarrow \mathbb{R}$$

FPNA: it is not enough

- The definition of FPNAs is relaxed and partialy unspecific.
- It defines the existence of the activators and links operators.
- But it **does not** define their concrete values.
- Using *Pred* and *Succ* sets it defines the order of neural resources but it **does not** define the structure of their interconnection.
- It **does not** define the default values.
- Thus for creting a fully specified network (design) there is needed for some more specification than FPNA offers.
- The remaining details specify the Field programmable neural Network – FPNN.

FPNN: the FPNA instance

- Let N be a set of nodes (activators) and E a set of oriented edges (links). We say that graph (N,E) is an FPNN if the following statements hold:
  - (N,E) is an FPNA
  - default value of activators iteration variable:

$$\forall n \in N : \exists \Theta_n \in \mathbb{R}$$

  - a number of iterations:

$$\forall n \in N : \exists a_n \in \mathbb{N}$$

  - concrete values of: $\forall \alpha_{(p,n)} : W_n(p) \in \mathbb{R} \wedge T_n(p) \in \mathbb{R}$
  - a number of inputs for every input node:

$$\forall n \in N_i : \exists c \in \mathbb{N}$$

FPNN: structure specification

- binary flag determining interconnection between an activator *n* and link *(p,n)*:

$$\forall\, p,n \in N \wedge p \in Pred(n) : \exists\, r_n(p) \in \{0,1\}$$

- binary flag determining interconnection between an activator *n* and link *(n,s)*:

$$\forall\, n,s \in N \wedge s \in \mathrm{Succ}(n) : \exists\, s_n(s) \in \{0,1\}$$

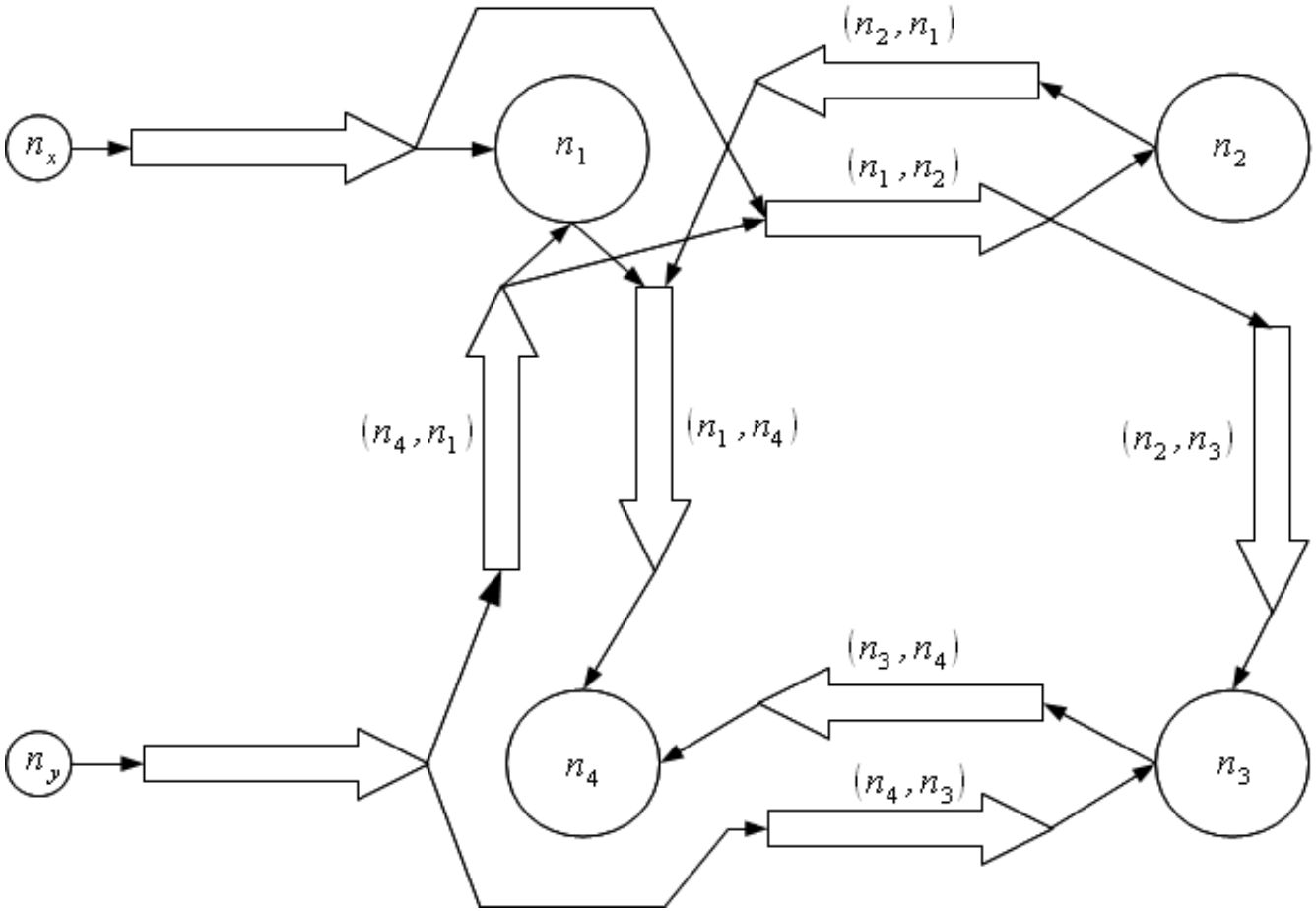
- binary flag determining interconnection between a link *(p,n)* and link *(n,s)*:

$$\forall\, n,p,s \in N \wedge p \in Pred(n) \wedge s \in \mathrm{Succ}(n) : \exists\, R_n(p,s) \in \{0,1\}$$

- binary flag determining interconnection between an input node *n* and link *(n,s)*:

$$\forall\, n \in N_i, \forall\, s \in N \wedge s \in \mathrm{Succ}(n) : \exists\, S_n(s) \in \{0,1\}$$

FPNN: an example



FPNN: the grid FPNN

- For every neural resource there is a set of positive flags:

$$\forall n \in N : \exists \mathrm{setr}_n = \{ r_n(p) | r_n(p) = 1 \wedge p \in Pred(n) \}$$

$$\forall n \in N : \exists \mathrm{sets}_n = \{ s_n(s) | s_n(s) = 1 \wedge s \in \mathrm{Succ}(n) \}$$

$$\forall (n, p) \in E : \exists \mathrm{setR}_n(p) = \{ R_n(p, s) | R_n(p, s) = 1 \wedge s \in Pred(n) \}$$

$$\forall n \in N_i : \exists setS_n(s) = \{ S_n(s) | S_n(s) = 1 \wedge s \in N \}$$

FPNN: the grid FPNN definiton

- Let N be a set of nodes (activators) and E a set of oriented edges (links). We say that graph (N,E) is an (standart/light) grid FPNN if the following restrictions hold:
  - (N,E) is an FPNN
  - a limited number of connected preceeding links:

$$\forall n \in N : |\mathrm{setr}_n| \geq 2$$

  - a limited number of connected successive links:

$$\forall n \in N : |\mathrm{sets}_n| \in (0,1)$$

$$\forall n \in N_i : |\mathrm{setS}_n| \in (0,1)$$

$$\forall (n,p) \in E : |\mathrm{setR}_n(p)| \in (0,1,2)$$

  - a limited number of iterations: $\forall n \in N : a_n \geq 2$
  - a limited number of inputs: $\forall n \in N_i : c_n = 1$

FPNN: the grid FPNN definiton

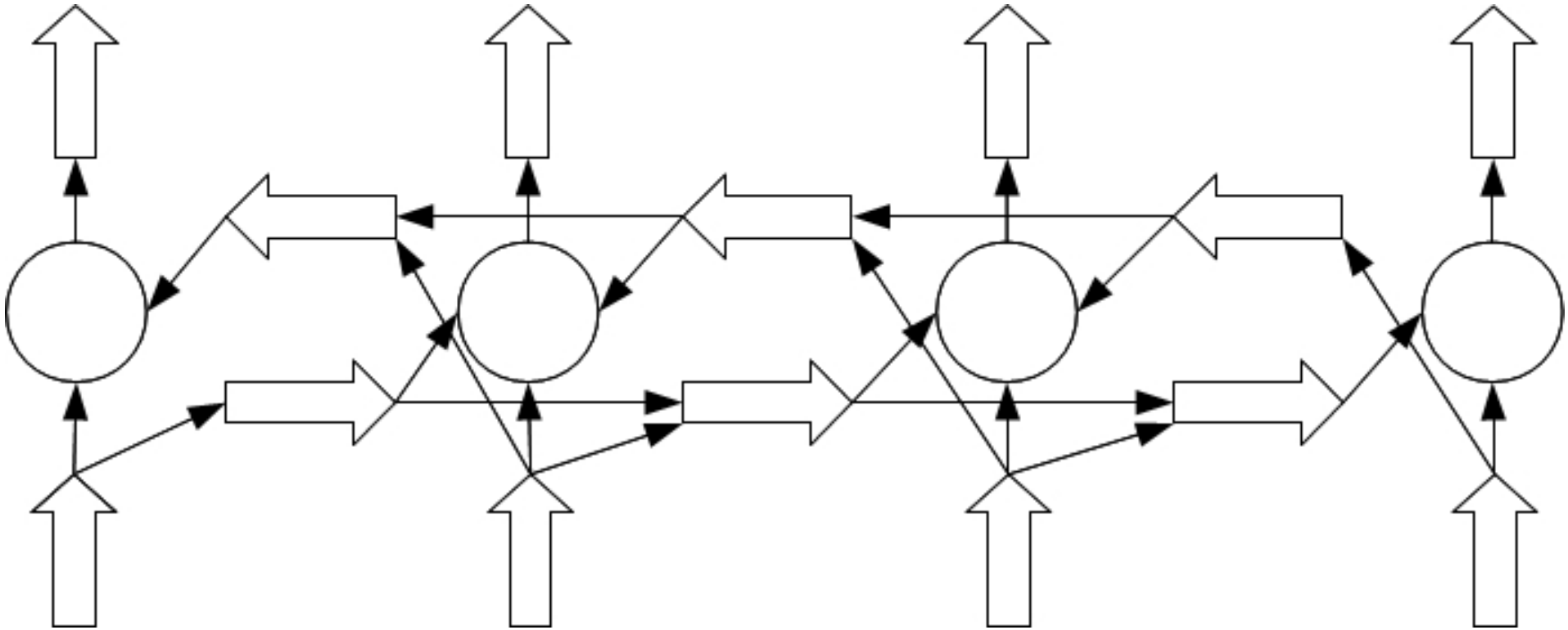
- for the standard grid FPNN:
  - an affine operator for every predecessor in every link:

$$\forall (r,n) \in E \wedge p \in Pred(n) : \exists \alpha_{(p,n)} = W_n(p) \cdot x + T_n(p)$$

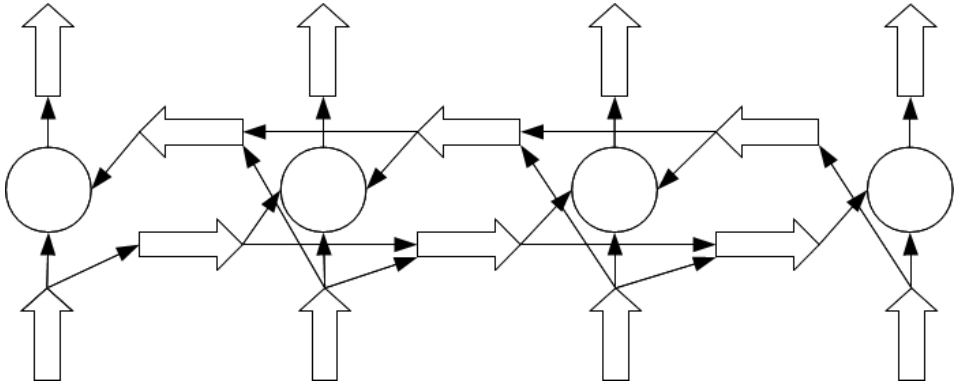
- for the light grid FPNN:
  - only one affine operator for every link:

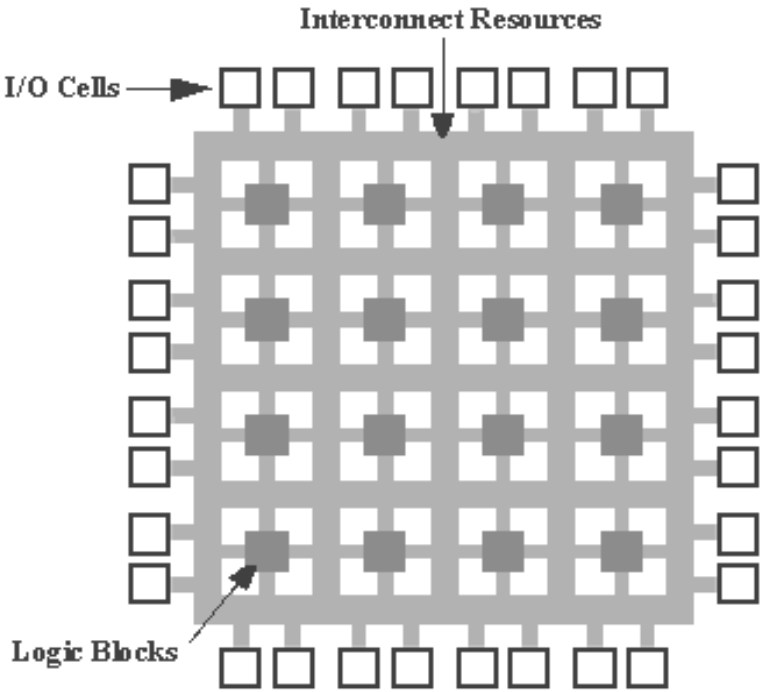
$$\forall p,n \in N \wedge (p,n) \in E : \exists \alpha_{(p,n)} = W \cdot x$$

FPNN: the grid FPNN example

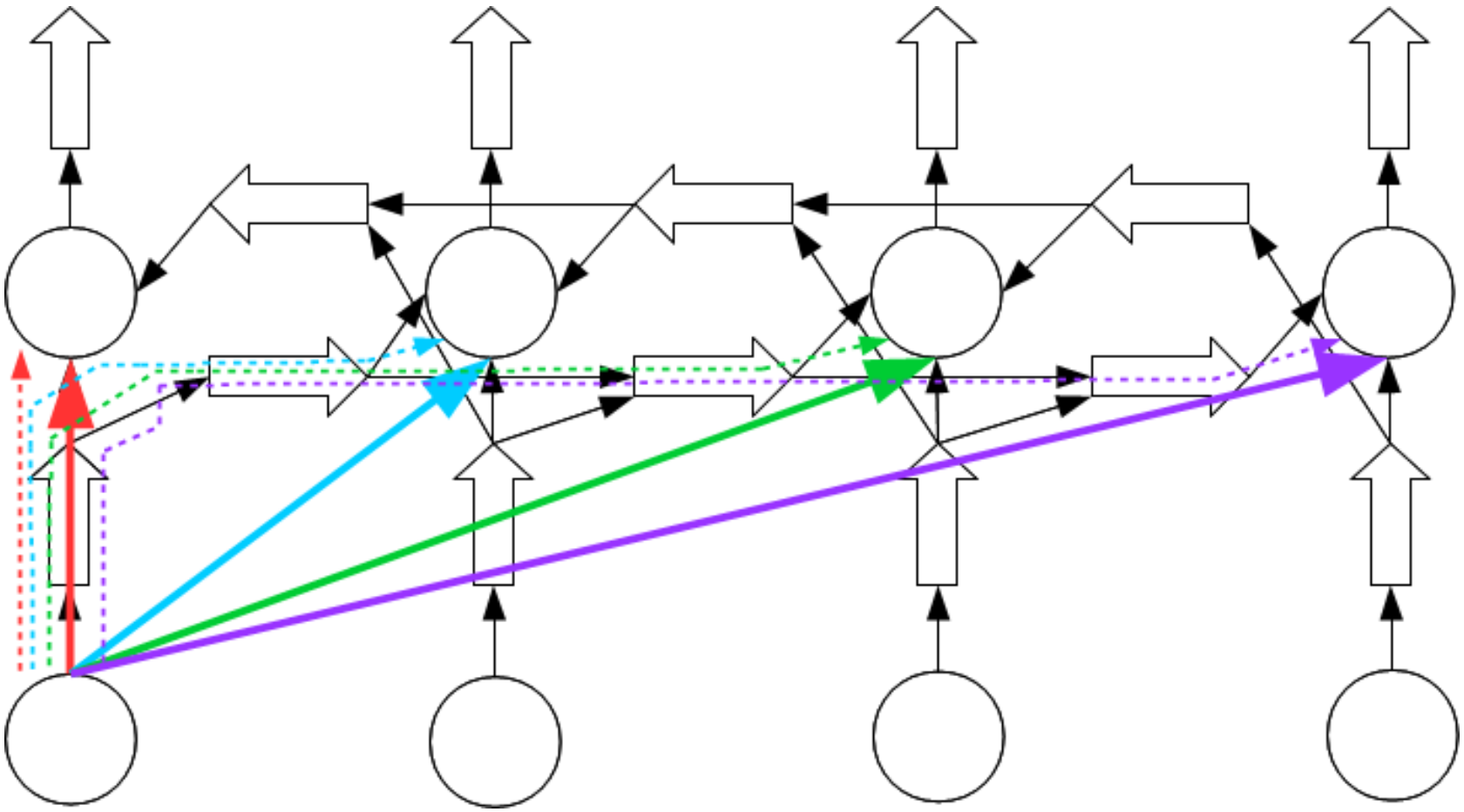


FPNN: the grid FPNN example





FPNN: the grid FPNN example



Terms

1)Chain of links

- A sequence of links bordered by links having no predecessors (the beginning of the chain) or no successors (the end of the link).

2)Path between links

- Is a subset of chain of links.

Fault tolerant mapping algorithm

0) Declarations

a) Neural Network is a pair $(Neurons, Synapses)$

b) (N, E) is a grid FPNN.

c) Functions:

- Mapping neurons to activators: $NeuToAct : Neurons \rightarrow N$
- Inverse mapping: $ActToNeu = NeuToAct^{-1}$
- Source and destination neuron of a synapse:

$$srcNeuron : Synapses \rightarrow Neurons$$

$$dstNeuron : Synapses \rightarrow Neurons$$

Fault tolerant mapping algorithm

- Sorting by path length: $sortByLength : E^n \rightarrow E^n$
- Find path: $findPath : E \times E \rightarrow E^n, n \geq 1$
- Path's first node : $firstNodeOf : E^n \rightarrow E$

d)Sets

- Set of synapses ending in the link:

$$\forall u \in E : \exists aproximatedSynapses_u \subset Synapses$$

- Set of synapses passing trough the link:

$$\forall u \in E : \exists passingSynapses_u \subset Synapses$$

- Set of connected preceding activators:

$$\forall u \in E : \exists connPred_u \subset N$$

Fault tolerant mapping algorithm

e)Variables

- Synapses weight:

$$\forall\, syn \in Synapses : \exists\, syn_{value} \in \mathbb{R}$$

- Partial product approximating the synapse:

$$\forall\, syn \in Synapses : \exists\, syn_{product} \in \mathbb{R}$$

- Last multiplicand of approximation product:

$$\forall\, syn \in Synapses : \exists\, syn_{multiplicand} \in \mathbb{R}$$

f)Others

- $paths \subset E^*$ is an ordered collection of paths

Fault tolerant mapping algorithm

1) Initialization
   a) Construct a graph of the link connection.
   b) Separate components of the graph and store them in CONN.
   c) Select the links intended to have a non-redundant security and store them in SEL.

2)Mapping of links

$$\forall (U,V) \in CONN:$$

a)Expansion of predecessors *approximatedSynapses:*

$$\forall (n,s) \in SEL \wedge (n,s) \in U:$$

$$\forall (p,n) \in Pred(n) \wedge R_n(p,s):$$

$$\forall\, syn \in approximatedSynapses_{(n,s)} \wedge \exists\, act \in connPred_{(p,n)} \wedge$$
$$\wedge\, ActToNeu(act) = srcNeuron(syn) \Rightarrow$$
$$\Rightarrow syn \in approximatedSynapses_{(p,n)}$$

Fault tolerant mapping algorithm

b)Paths determining:

$$first = \{u \in U \mid deg^{+}(u) = 0\}$$
$$last = \{u \in U \mid deg^{-}(u) = 0\}$$
$$paths = \emptyset$$
$$\forall u \in first \wedge \forall v \in last:$$
$$p = findPath(u, v)$$
$$paths = paths \cup 2^{p}$$
$$sortByLength(paths)$$

Fault tolerant mapping algorithm

c)mapping path by path

$$\forall r \in paths:$$

i. Multiplicands computation:

$$(p,n) = firstNodeOf(r)$$

$$\forall synapse \in approximatedSynapses_{(p,n)}:$$

$$syn_{multiplicand} = \frac{syn_{value}}{syn_{product}}$$

$$W_n(p) = syn_{multiplicand} \Leftrightarrow$$
$$\Leftrightarrow srcNeuron(syn) = p \wedge$$
$$\wedge dstNeuron(syn) = n$$

(standard)

$$W_{(p,n)} = \frac{\sum_{syn \in approximatedSynapses_{(p,n)}} syn_{multiplicand}}{|approximatedSynapses_{(p,n)}|}$$

(light)

Fault tolerant mapping algorithm

ii. Updating the products using the new parameters:

$$\forall\, synapse \in passingSynapses_{(p,n)}:$$

$$synapse_{product} = synapse_{product} * W_n(p) \Leftrightarrow$$
$$\Leftrightarrow srcNeuron(syn) = p \wedge$$
$$\wedge\, dstNeuron(syn) = n$$

(standard)

$$synapse_{product} = synapse_{product} * W_{(p,n)}$$

(light)

iii. Deleting finished node from the path:

$$r = r \setminus \{(p,n)\}$$
$$paths = \{p \mid p \in paths \wedge p \neq \emptyset\}$$

Fault tolerant importance ranking algorithm

1) Declarations

a) Length of rest of a chain: $restOfChain : E^n \times E \rightarrow \mathbb{N}$

b) Set of chains a link belongs to: $chains : E \rightarrow \{E^n\}^m$

c) Fault tolerant ranking: $\forall link \in E : \exists rank_{link} \in \mathbb{R}$

2) Algorithm

$$\forall link \in E :$$

$$rank1 = \sum^{syn \in approximatedSynapses_{link}} syn_{value}$$

$$rank2 = \sum^{chain \in chains(link)} restOfChain(chain, link)$$

$$\boxed{rank_{link} = rank1 + rank2}$$

Summary

- The FPNN is an approximation of artificial neural networks.
- It is usable for resource saving implementation in the FPGAs.
- Especially grid FPNNs show the resources saving properties.
- Trained artificial neural networks can be directly mapped onto FPNNs.
- The redundancy free fault tolerance can be established during the mapping.
- The links can be ranked with relation to their importance in the circuit.

Bibliography

- Girau, B.: FPNA: Applications and Implementations. In FPGA Implementations of Neural Networks, editation A. R. Omondi; J. C. Rajapakse, Springer US, 2006, ISBN 978-0-387-28487-3, p.81-136, 10.1007/0-387-28487-7-4. URL http://dx.doi.org/10.1007/0-387-28487-7-4
- Martin Krcma: The neural networks acceleration in FPGA, bachelor's thesis, Brno, FIT BUT in Brno, 2012
- Martin Krcma: The neural networks acceleration in FPGA, master's thesis, Brno, FIT BUT in Brno, 2012
- http://en.wikipedia.org/wiki/Field-programmable_gate_array
- http://home.mit.bme.hu/~szedo/FPGA/fpgahw.htm

Thank you for your attention