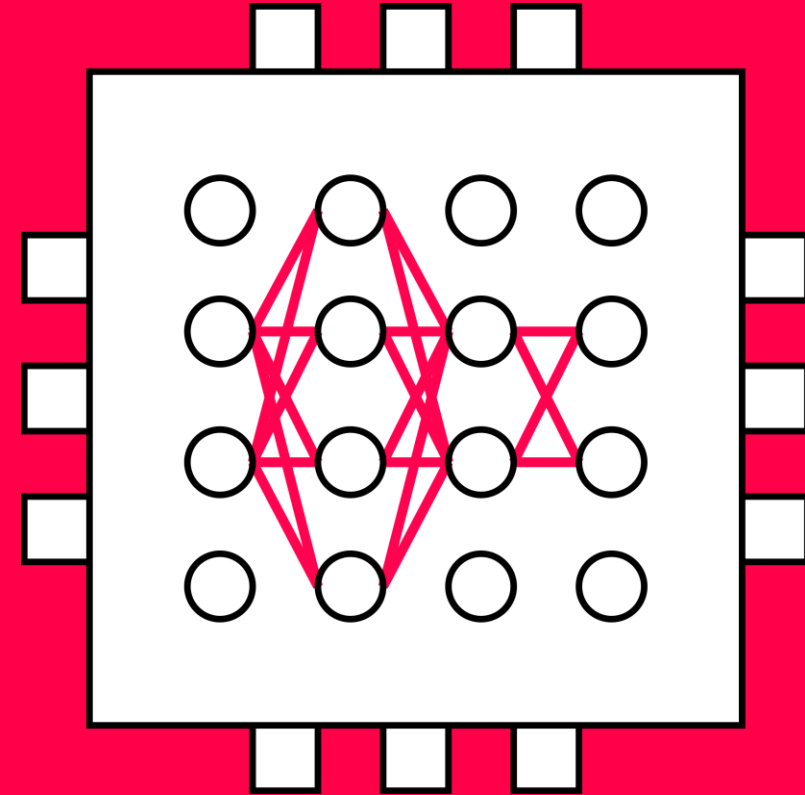


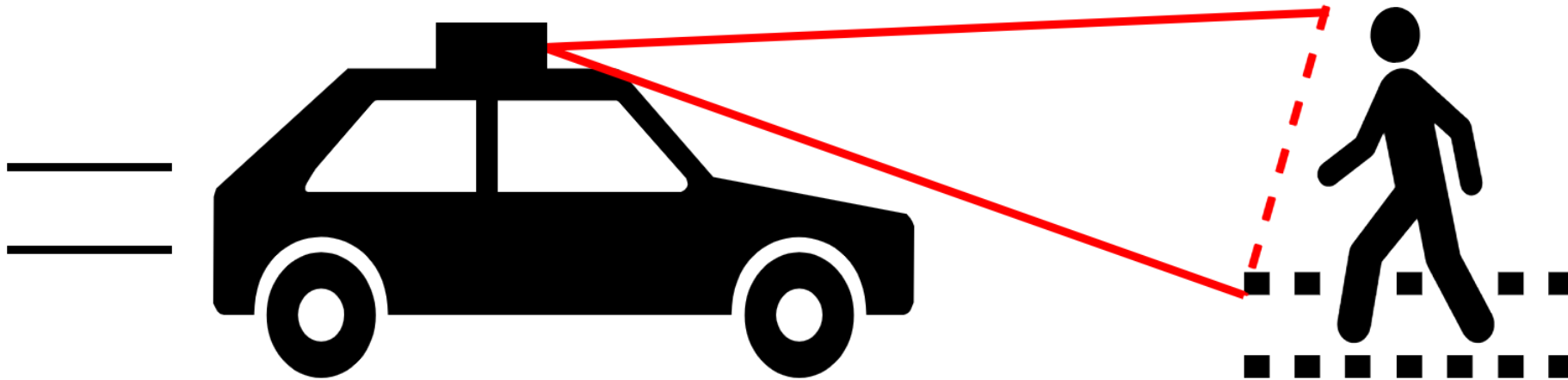
THANNA

Technische Hochschule Augsburg

Neural Network Accelerator



EINLEITUNG



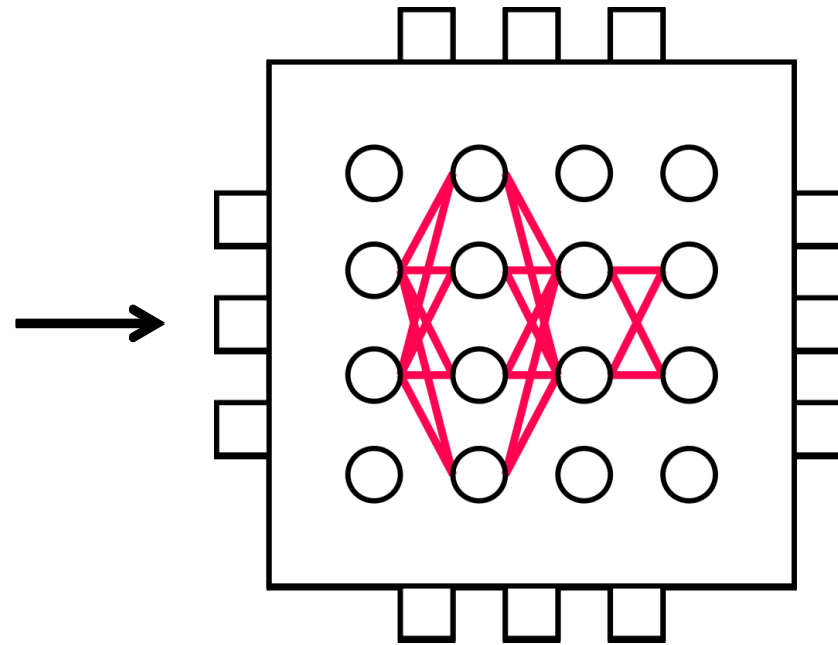
ZIELE

- Optimierte Neuronale Netze
- Effiziente Hardware für KI-Anwendungen
- Optimale Anpassung an gegebene Neuronale Netze
- Einfache Benutzung

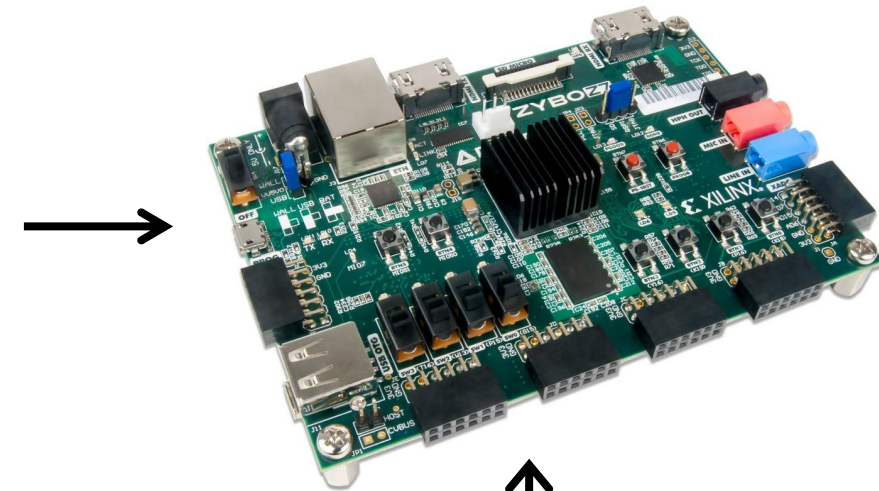
THANNA FRAMEWORK

```
31 def __init__(self, settings):
32     self.file = None
33     self.fingerprints = set()
34     self.logdupes = True
35     self.debug = debug
36     self.logger = logging.getLogger('THANNA')
37     if path:
38         self.file = open(os.path.join(path, 'fingerprint.txt'), 'w')
39         self.file.seek(0)
40         self.fingerprints.update(fingerprints)
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('debug')
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     return request_fingerprint(request)
```

THANNA Quantizer



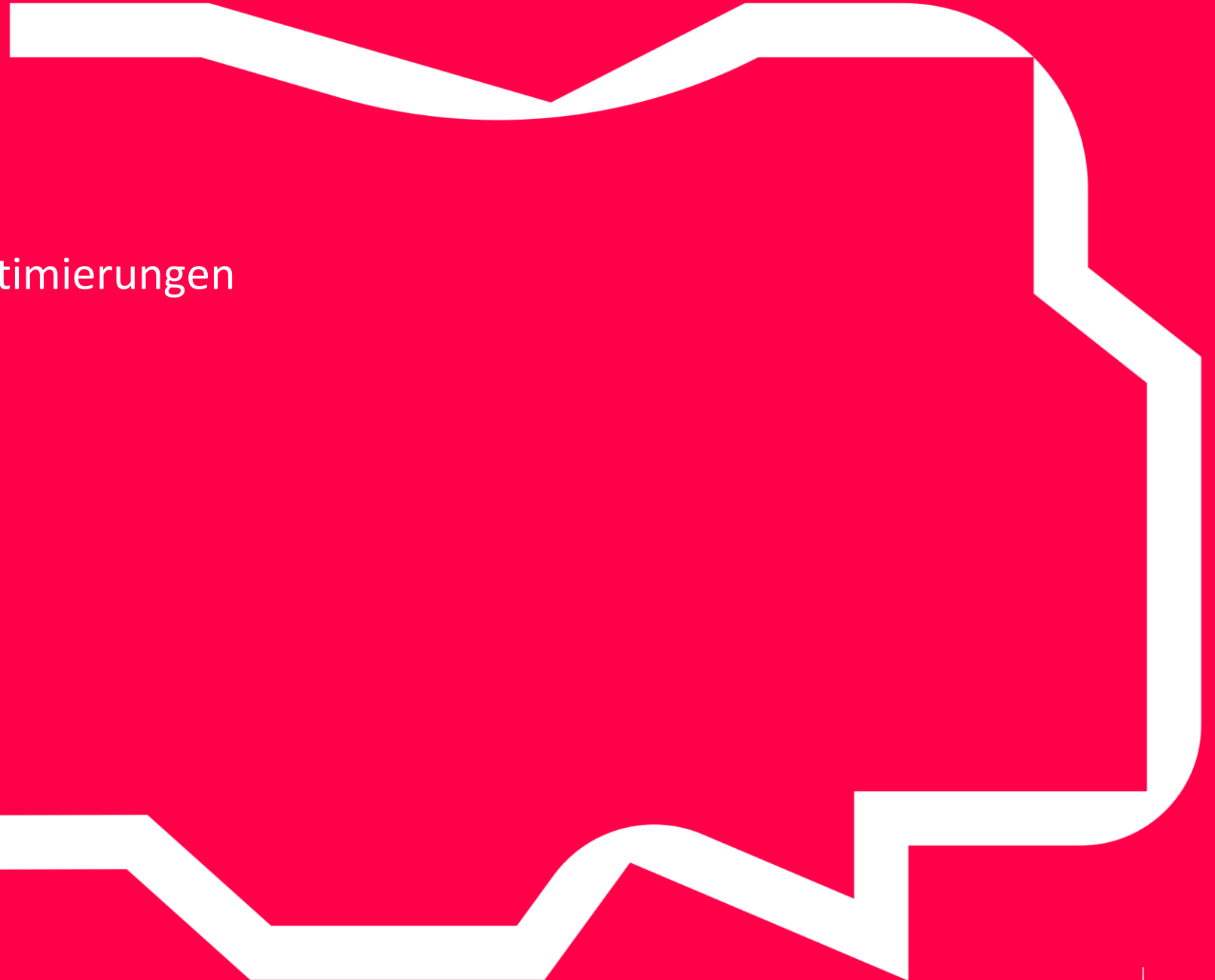
THANNA Prozessor



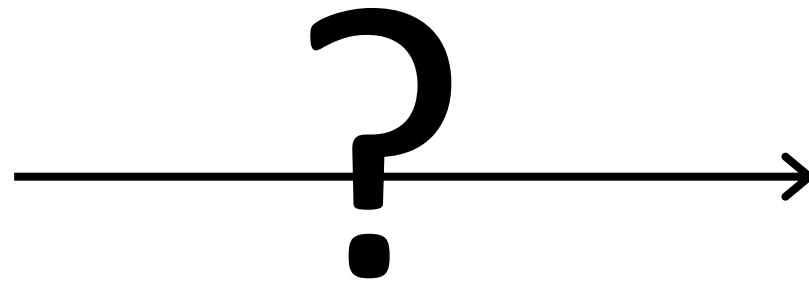
PetaLinux
THANNA Treiber
THANNA Library



1. Neuronale Netze und Optimierungen
2. THANNA Quantizer
3. THANNA Prozessor
4. THANNA Treiber

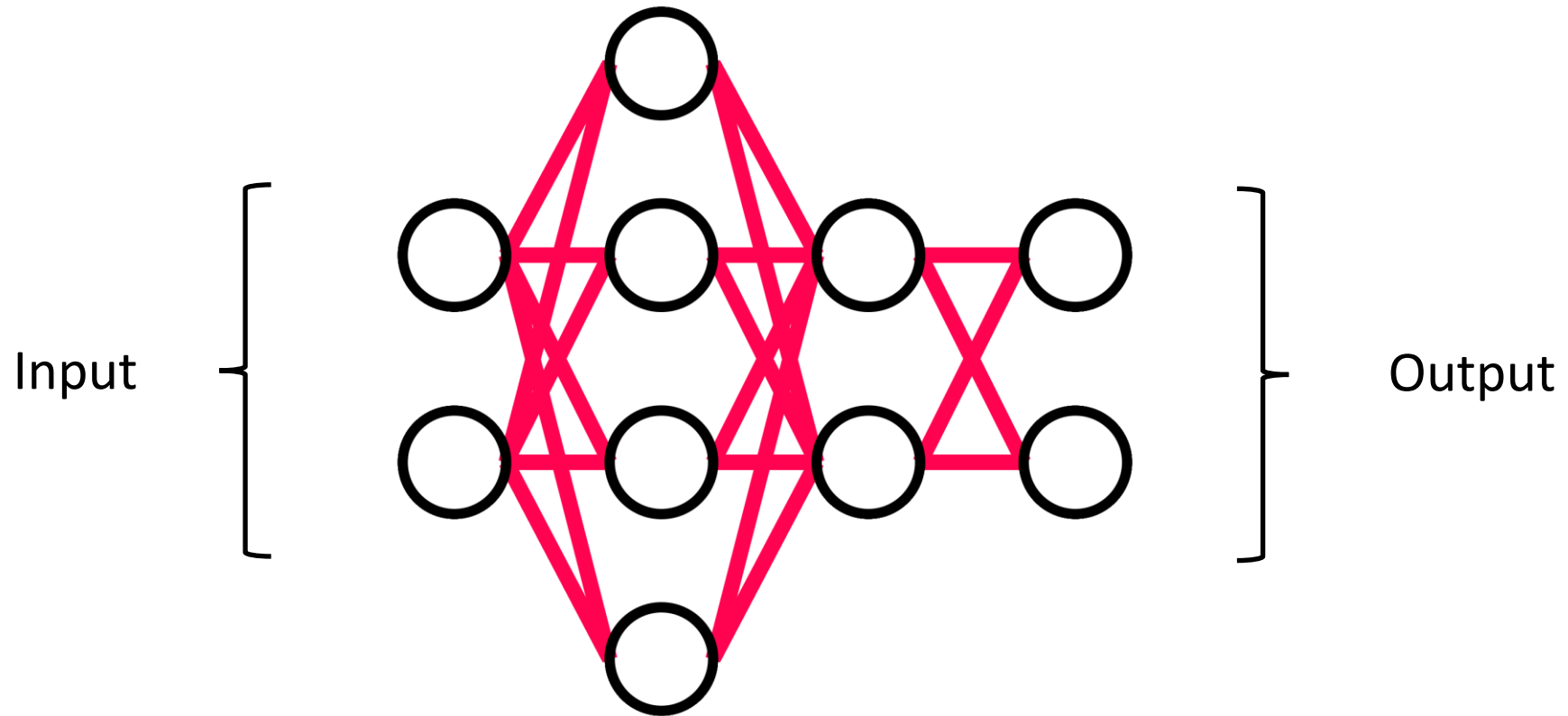


WIE FUNKTIONIEREN NEURONALE NETZE?



Hund

FUNKTIONSWEISE NEURONALER NETZE



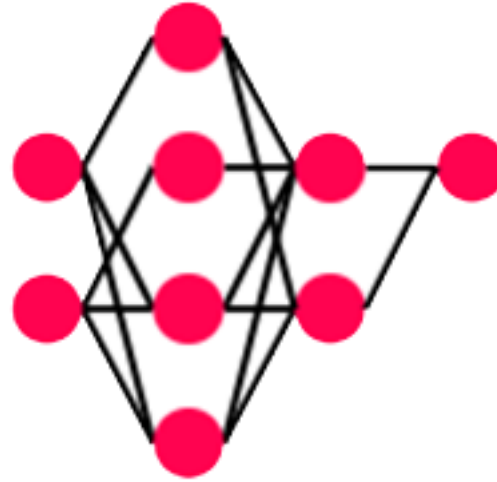
FUNKTIONSWEISE NEURONALER NETZE



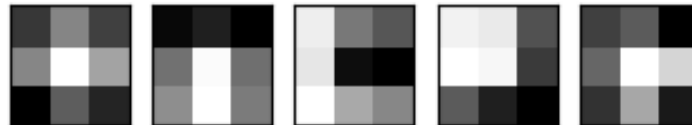
Hund

HAUPTKOMPONENTE

- Fully Connected Layer



- Convolutional Layer



HAUPTKOMPONENTE

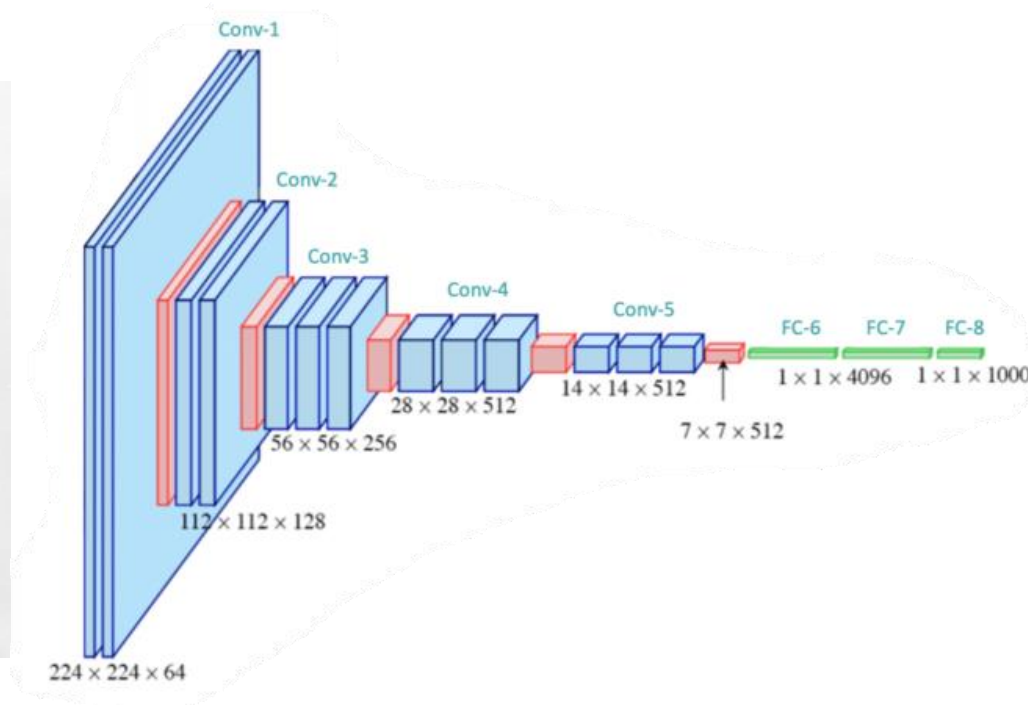
- Fully Connected Layer:

$$y_i = \sum_{j=0}^n x_j * w_{i,j}$$

- Convolutional Layer:

$$y_{i,j} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} x_{i+a,j+b} * w_{a,b}$$

FUNKTIONSWEISE NEURONALER NETZE



Hund

Quelle: Max Ferguson et al. "Automatic localization of casting defects with convolutional neural networks"

KLASSISCHE BESCHLEUNIGER

- Feste Anzahl und Struktur der Recheneinheiten
- Fixe Bit-Breite
- **Geringes Optimierungspotential**



GPU (RTX 2080)



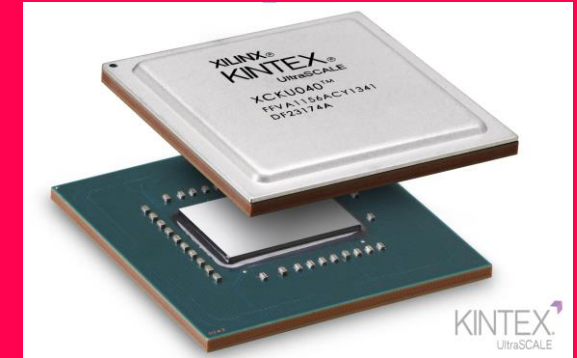
SoC (Edge TPU)

Quelle:wikipedia.com

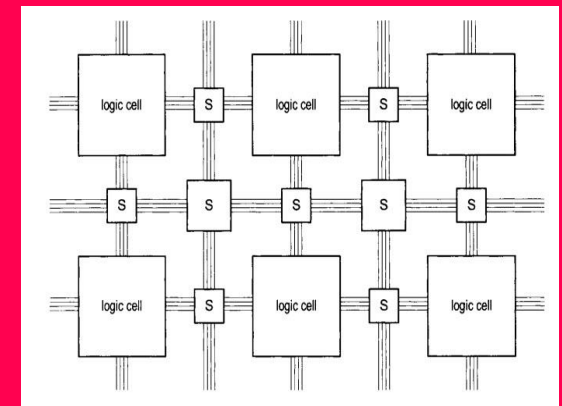
FPGA

(FIELD PROGRAMMABLE GATE ARRAYS)

- Aufbau aus Logikblöcken
 - Darstellung beliebiger Hardwarestrukturen
- **Hohes Optimierungspotential**



Xilinx FPGA



Interner Aufbau

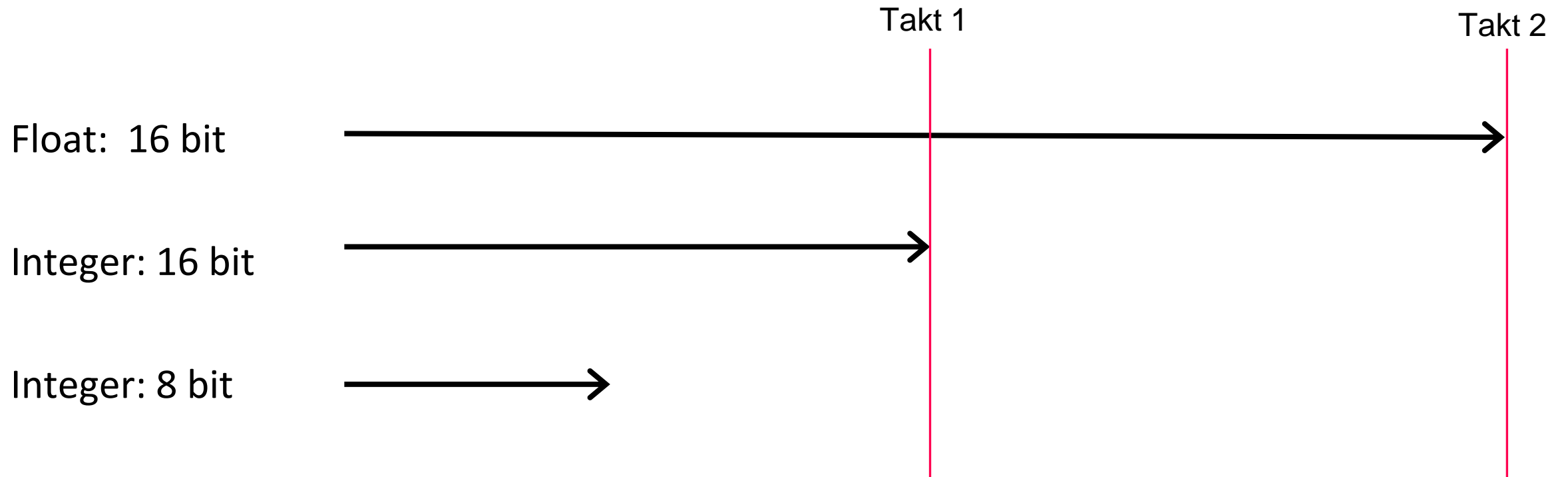
Quelle: xilinx.com

FPGA VS FESTE HARDWARE

	Variable Strukturen	Variable Bitbreiten	Parallelität
FPGA	✓	✓	✓
CPU	✗	✗	✗
GPU/TPU	✗	✗	✓

VORTEILE VON VARIABLEN FIXEDPOINT BITBREITEN

GESCHWINDIGKEIT



VORTEILE VON VARIABLEN FIXEDPOINT BITBREITEN

STROMVERBRAUCH / HARDWARE-AUFWAND

Float: 16 bit



Integer: 16 bit



Integer: 8 bit



1. Neuronale Netze und Optimierungen
2. THANNA Quantizer
3. THANNA Prozessor
4. THANNA Treiber

THANNA FRAMEWORK

```
31 def request_seen(self, request):
32     self.file = None
33     self.fingerprints = set()
34     self.logdupes = True
35     self.debug = debug
36     self.logger = logging.getLogger('THANNA')
37     if path:
38         self.file = open(os.path.join(settings.job_dir, 'THANNA.log'), 'a')
39         self.file.seek(0)
40         self.fingerprints.update(request.fingerprints)
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('debug')
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     return request_fingerprint(request)
```

THANNA Quantizer



Host-PC

QKERAS

Features:

- Open Source
- Quantisierungstool
- Post Training Quantization
- Quantization aware Training

Probleme:

- Fehlerhafte Beispiele
- Mehrere Bugs
- Hoher Genauigkeitsverlust bei vortrainierten Neuronalen Netzen
- 3-4x langsames Training

THANNA QUANTIZER

Features:

- Open Source
- Minimale Funktionalität von QKeras
- Quantisierung kleiner Netze
- Optimierung der Quantisierungsergebnisse

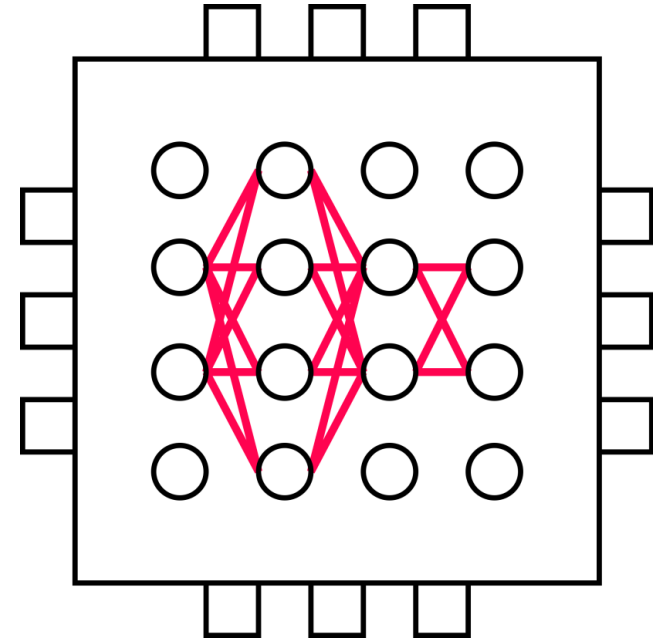
THANNA FRAMEWORK

```
31 self.file = None
32 self.fingerprints = set()
33 self.logdups = True
34 self.debug = debug
35 self.logger = logging.getLogger(__name__)
36 if path:
37     self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
38     self.file.seek(0)
39     self.fingerprints.update(fingerprint)
40
41 @classmethod
42 def from_settings(cls, settings):
43     debug = settings.getbool('debug')
44     return cls(job_dir(settings), debug)
45
46 def request_seen(self, request):
47     fp = self.request_fingerprint(request)
48     if fp in self.fingerprints:
49         return True
50     self.fingerprints.add(fp)
51     if self.file:
52         self.file.write(fp + os.linesep)
53
54 def request_fingerprint(self, request):
55     return request_fingerprint(request)
```

THANNA Quantizer



quantisiertes
neuronales Netz

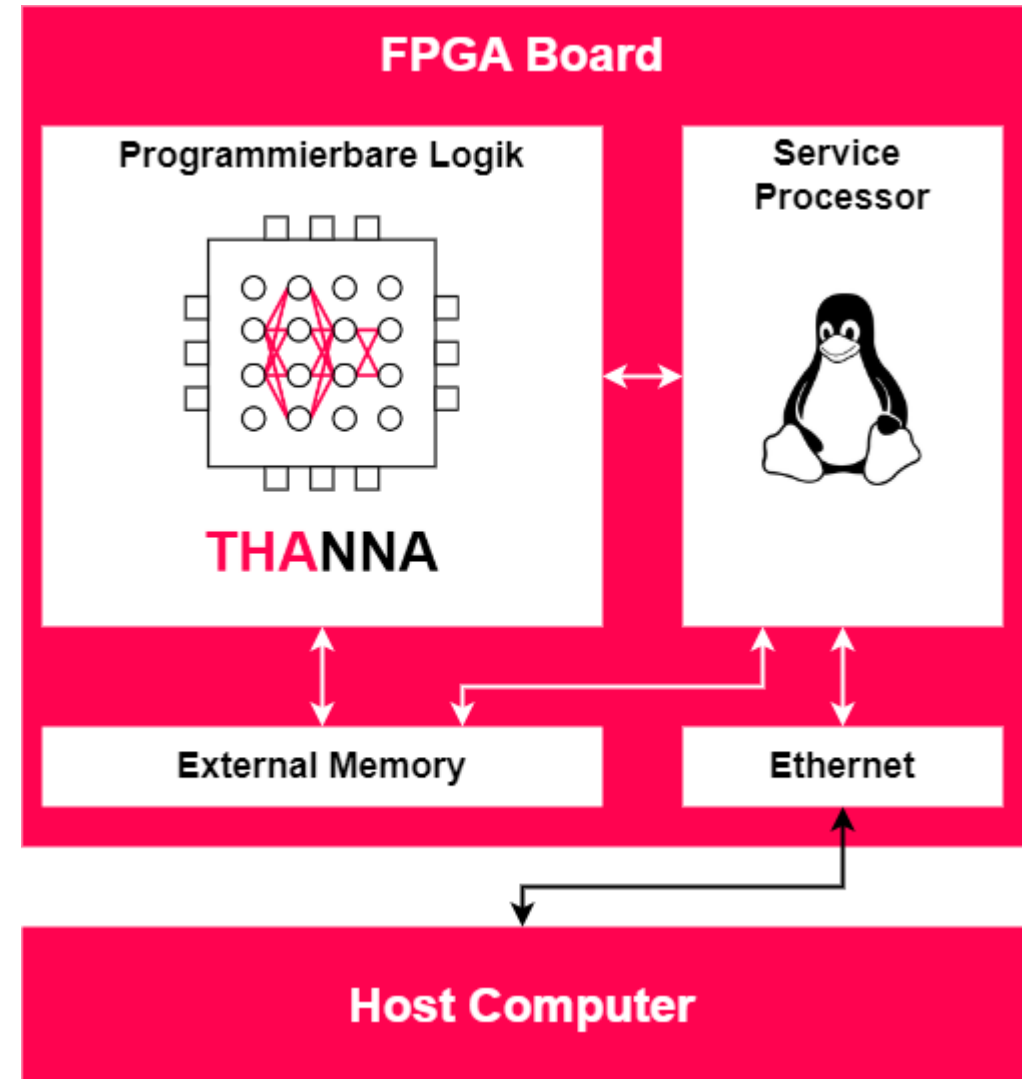


THANNA Prozessor

1. Neuronale Netze und Optiemierungen
2. THANNA Quantizer
3. THANNA Prozessor
4. THANNA Treiber

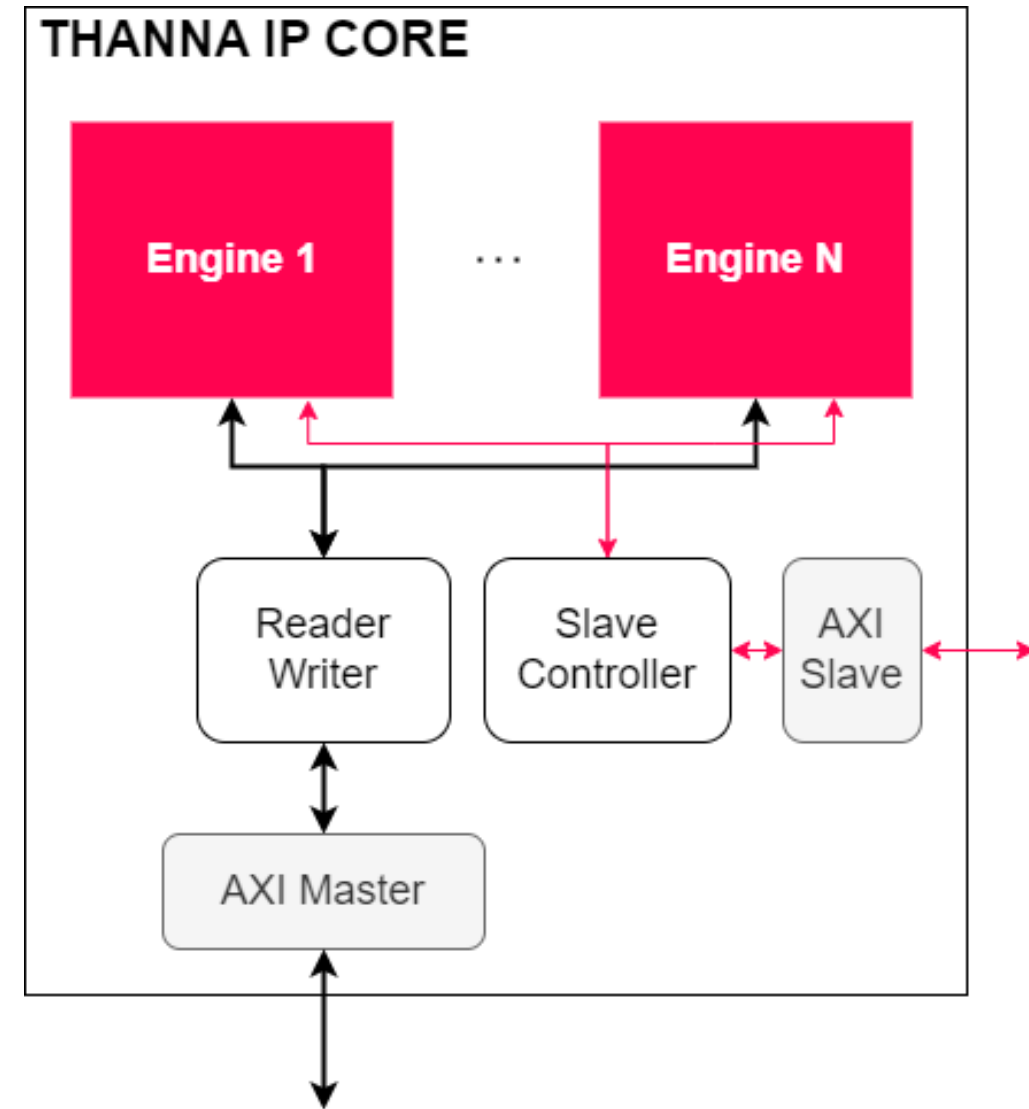
SYSTEM ÜBERSICHT

- Datenübertragung vom **Computer** bis zum **IP Core**



DER THANNA IP CORE

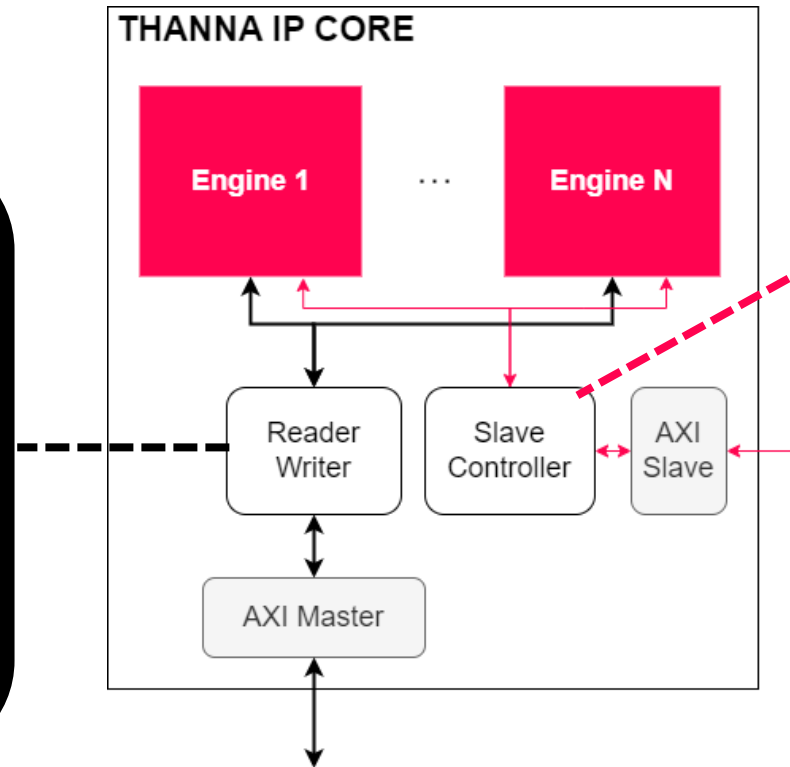
- **Flexible** Architektur
- **Erweiterbar** durch Anknüpfung beliebig vieler Engines



SCHNITTSTELLEN

Datenschnittstelle

- Burstdaten
- Lese- / Schreib-adressen
- Statusinformationen

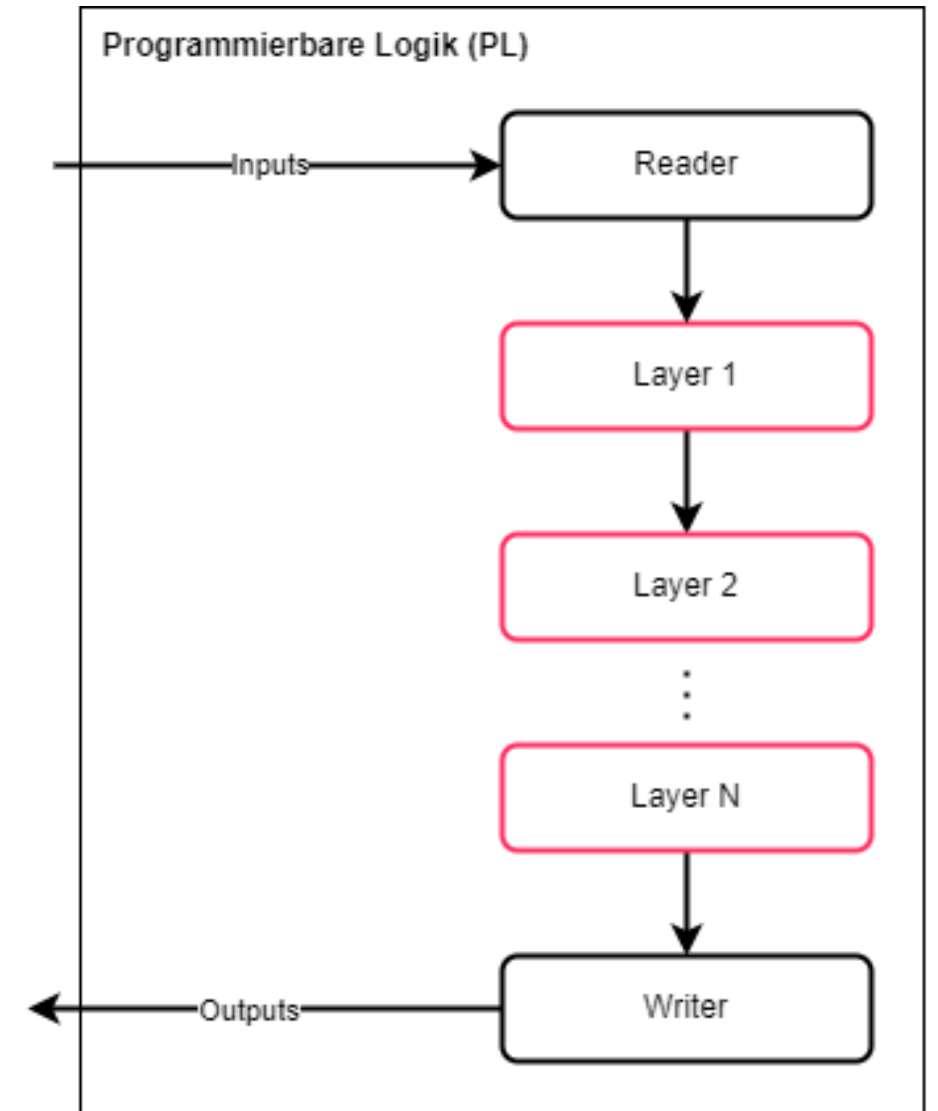


Kommunikations-schnittstelle

- Kommando
- Kommando Kontext
- Status

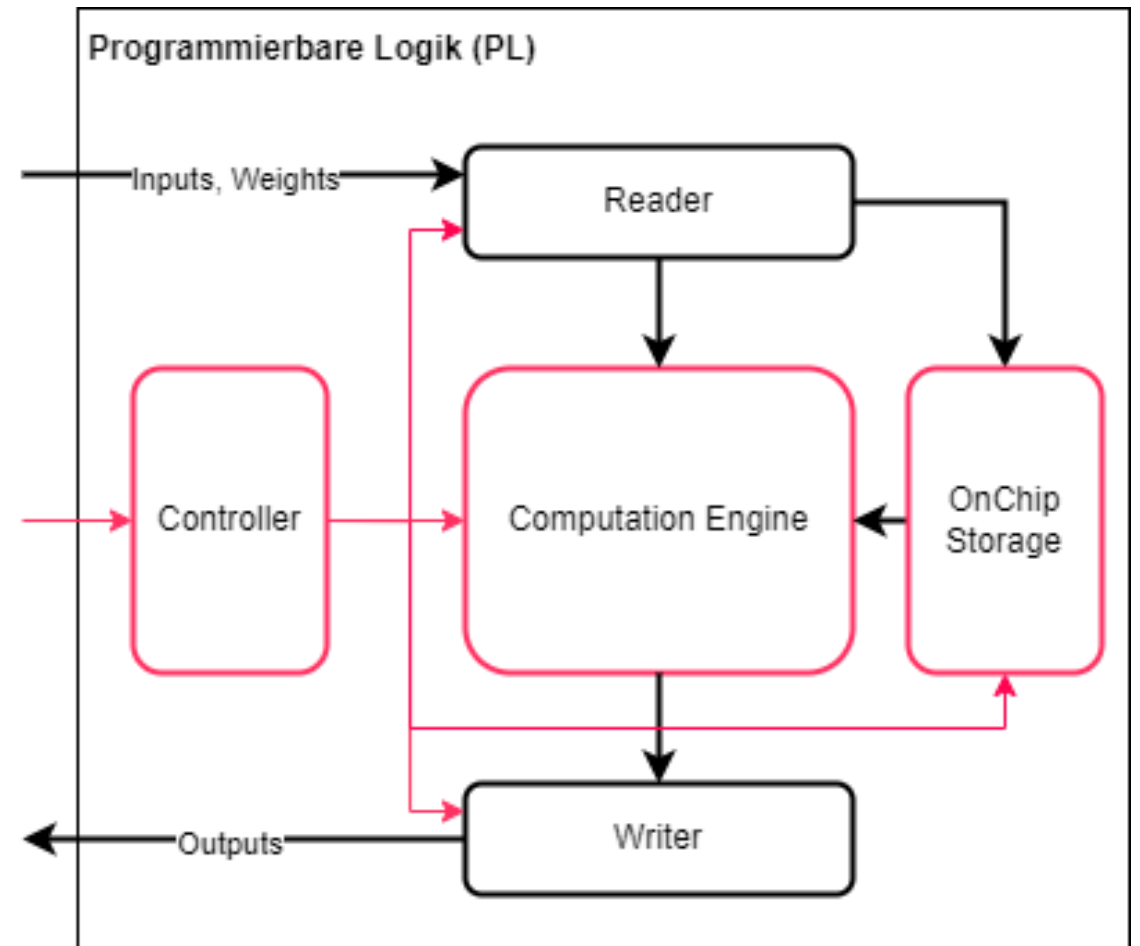
STREAMING ARCHITEKTUREN

- Berechnung von mehreren Layern durch mehrere Recheneinheiten
- **Schnell**
- **Hoher Ressourcenverbrauch**
- **Unflexibel**



SINGLE ENGINE ARCHITEKTUREN

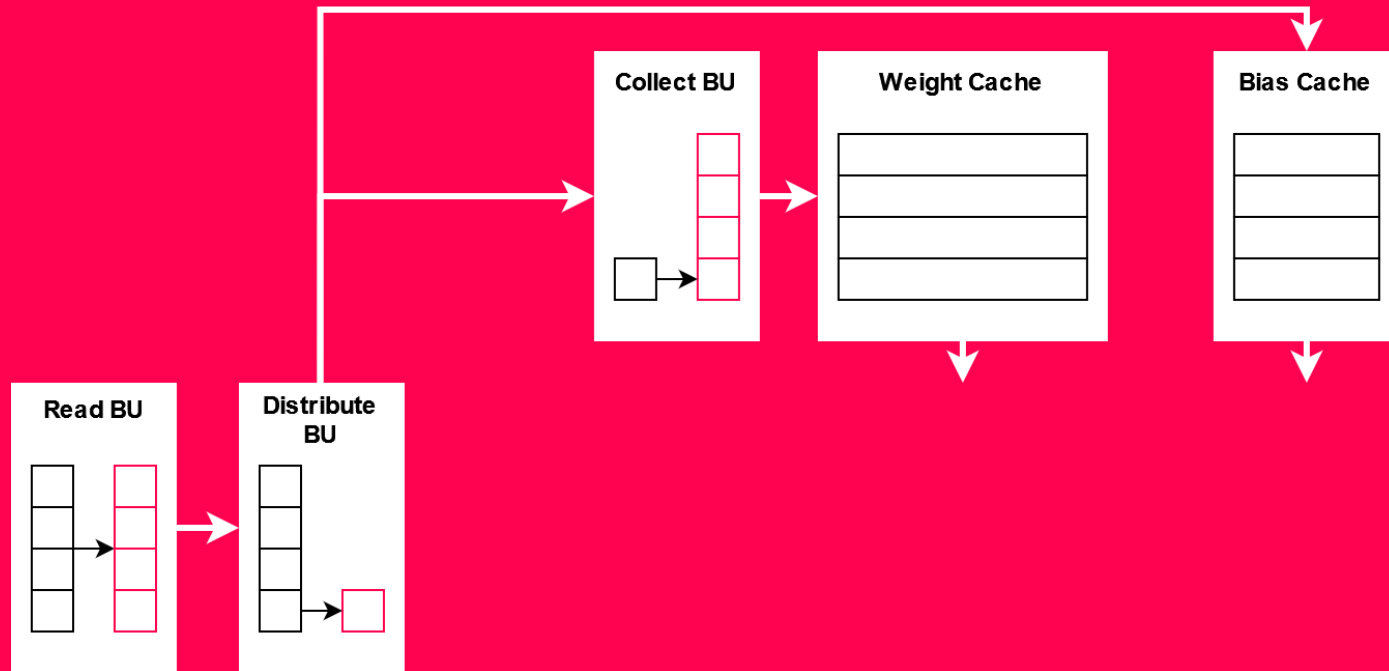
- Berechnung von mehreren Layern durch eine Recheneinheit
- **Langsamer**
- **Geringer Ressourcenverbrauch**
- **Flexibel**



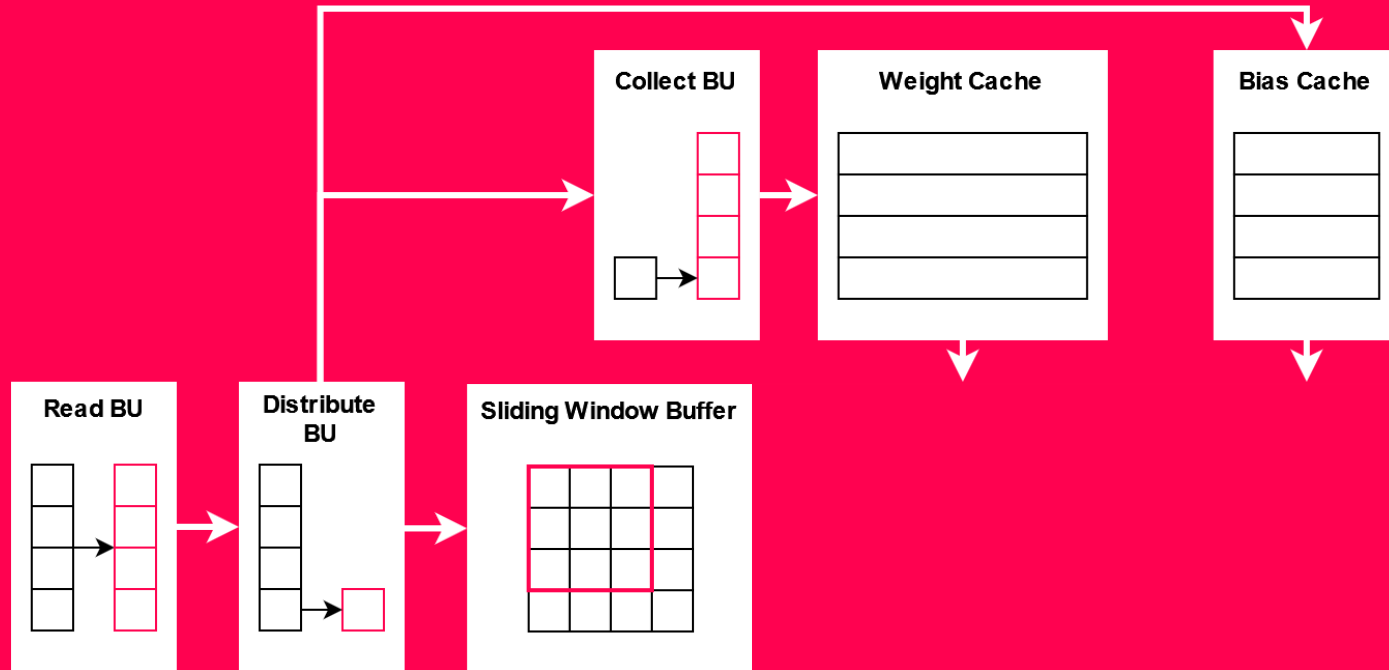
SINGLE ENGINE CORE ZIELE

- Flexibel für unterschiedliche Schichttypen
- Gut kombinierbar
 - Geringer Verbrauch an Logik Gattern
- Anpassbare Funktionalität und Ressourcenverbrauch

SINGLE ENGINE CORE



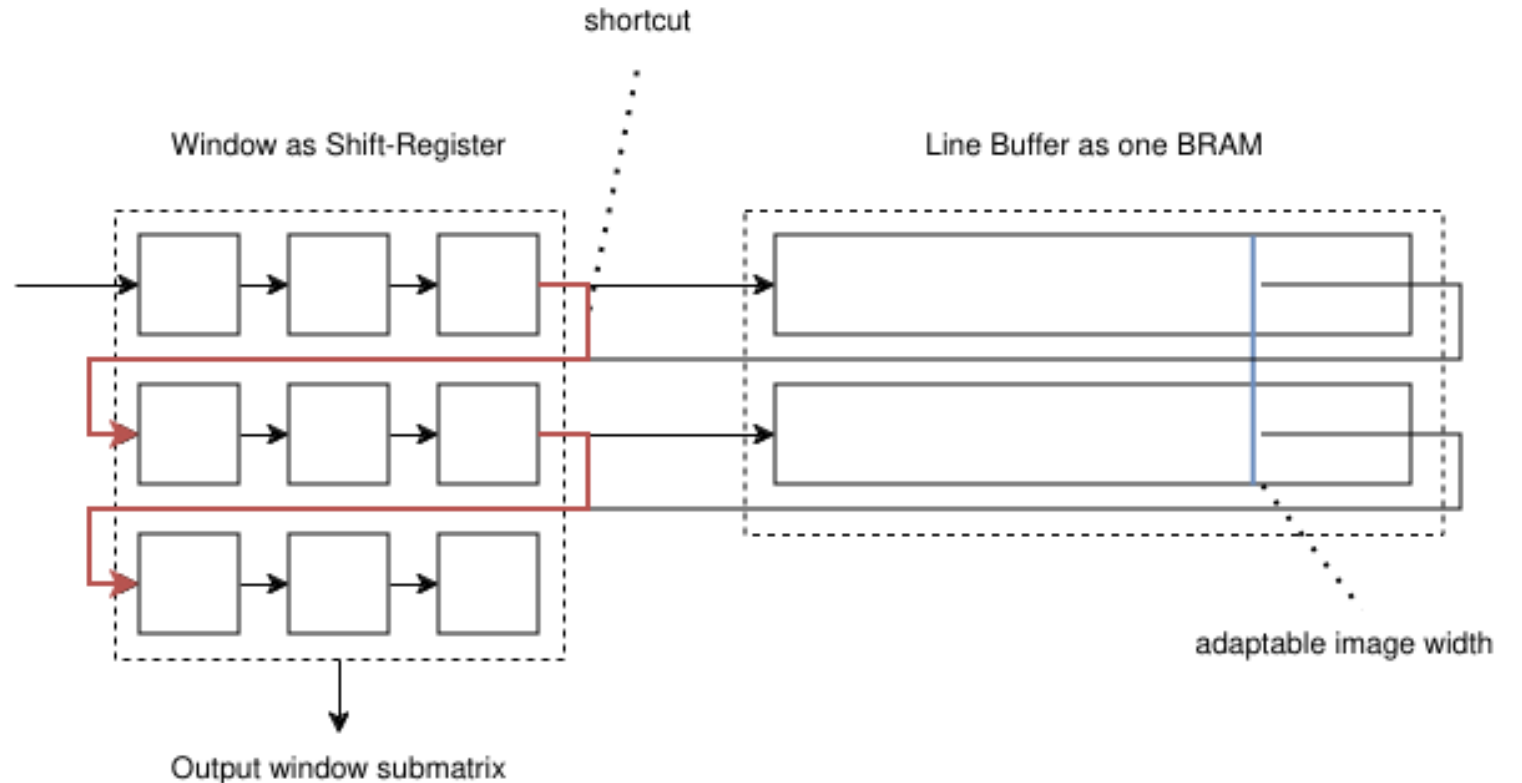
SINGLE ENGINE CORE



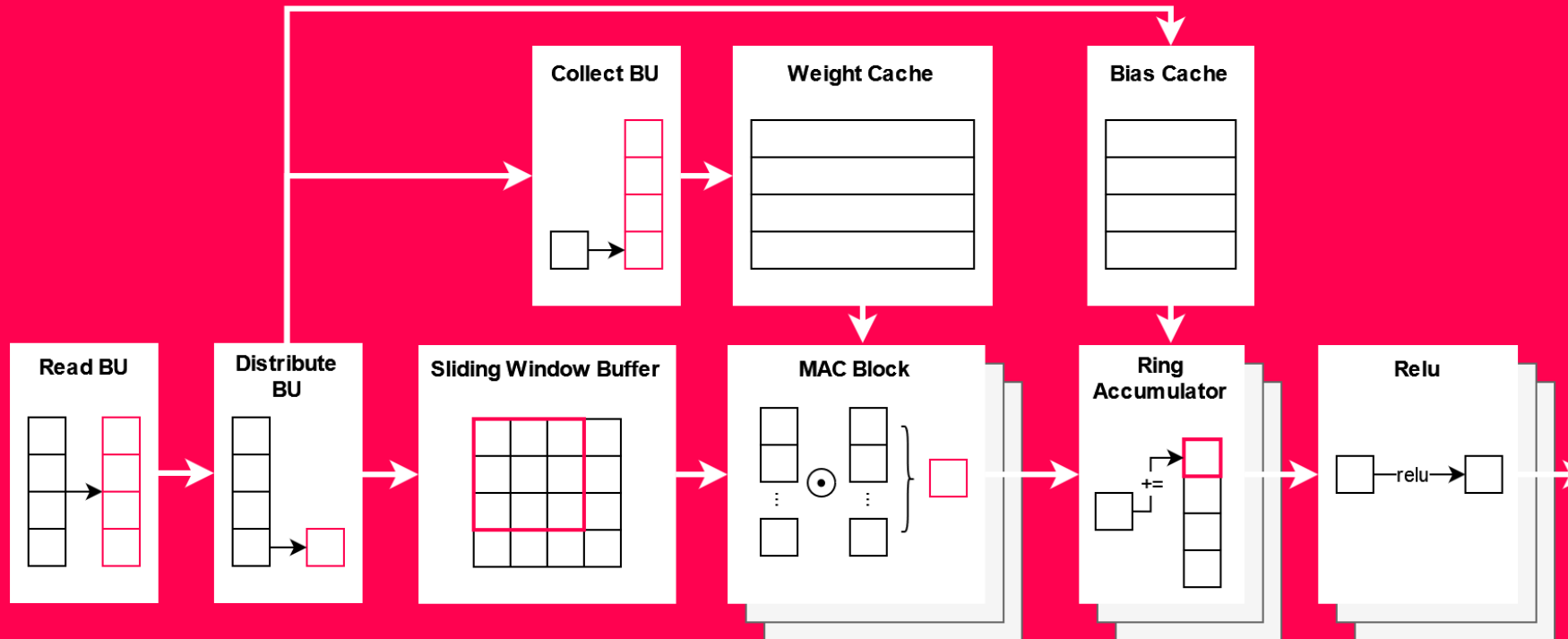
SINGLE ENGINE CORE

SLIDING WINDOW BUFFER

- Für Konvolutionen und Fully Connected Schichten
- Geringer Ressourcenverbrauch



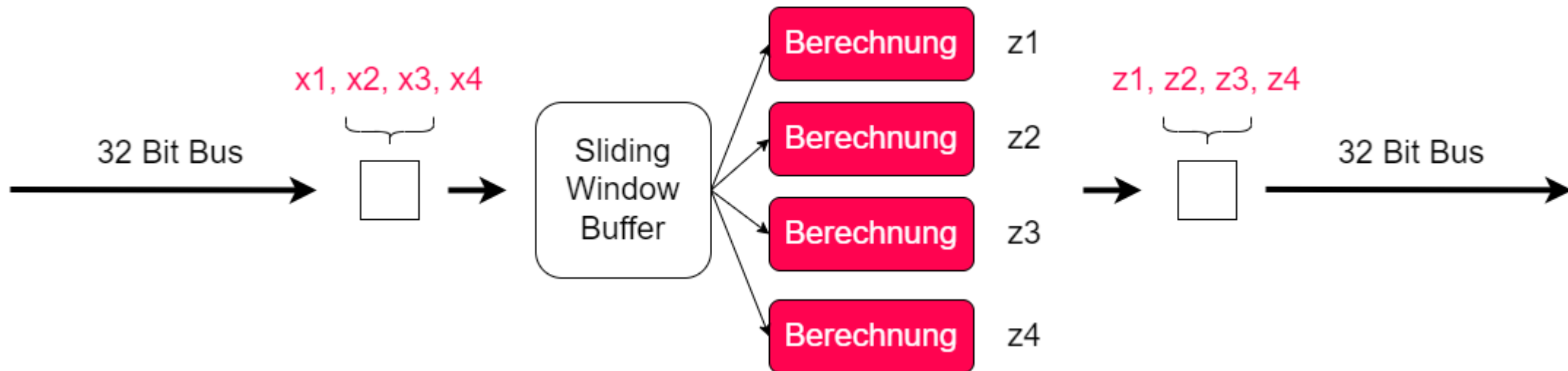
SINGLE ENGINE CORE



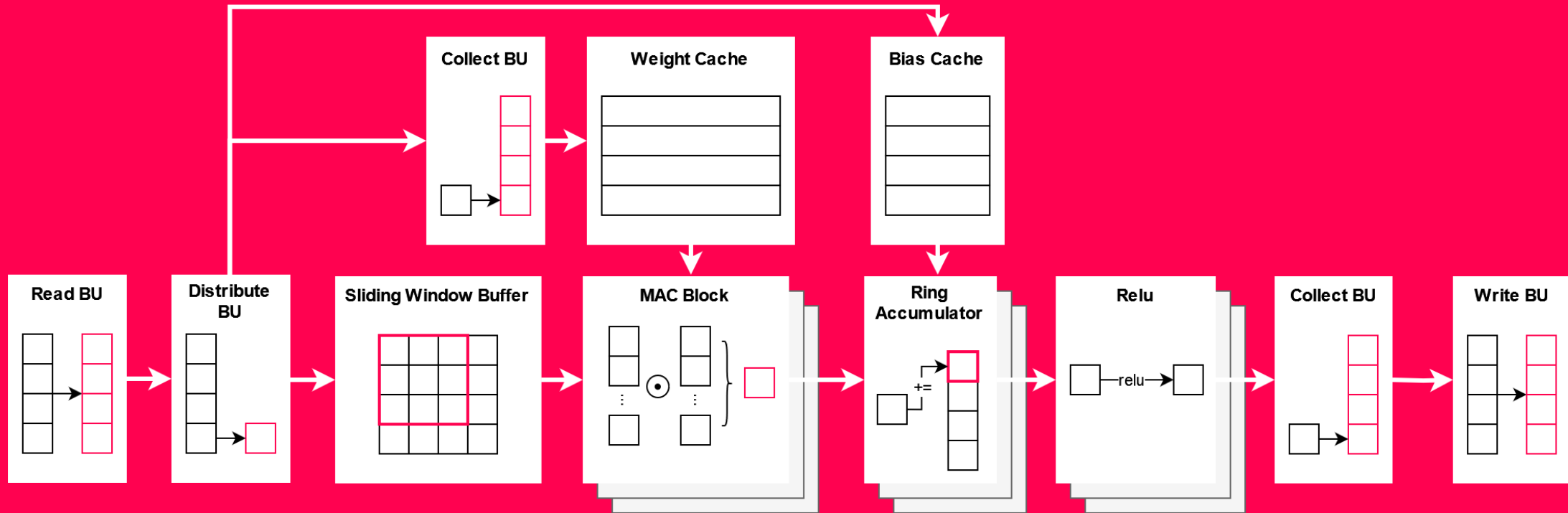
SINGLE ENGINE CORE

BUS BASIERTE SKALIERUNG

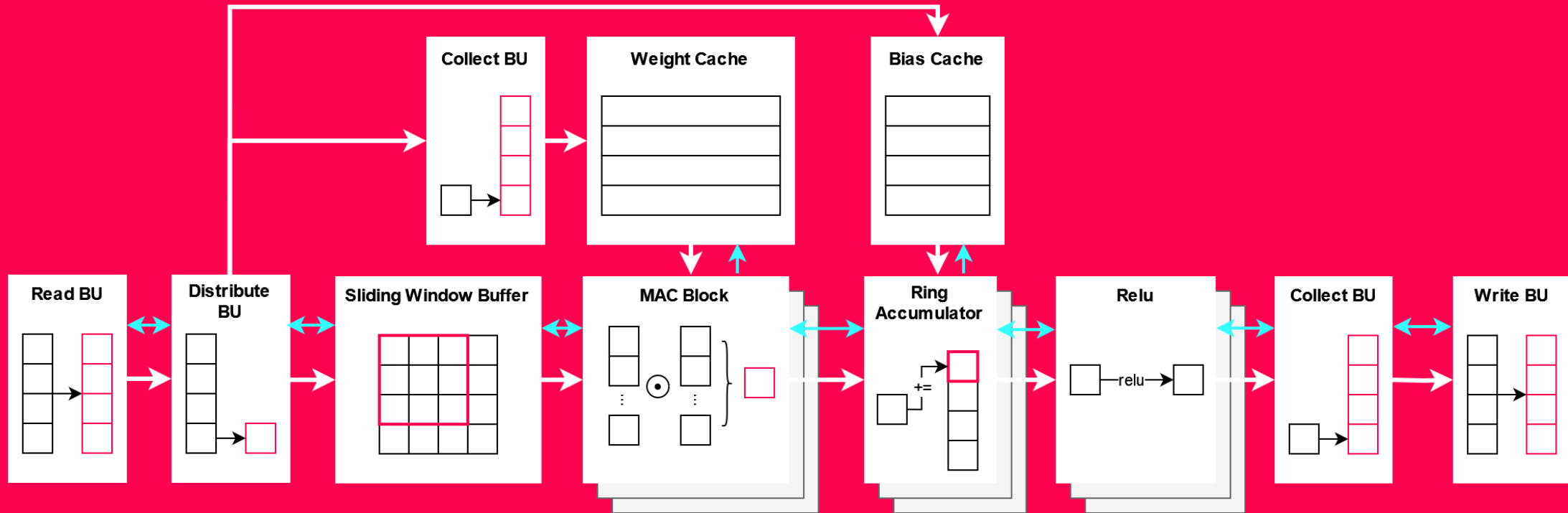
- Skalierung der Parallelisierung auf Basis der Busbreite des FPGAs



SINGLE ENGINE CORE

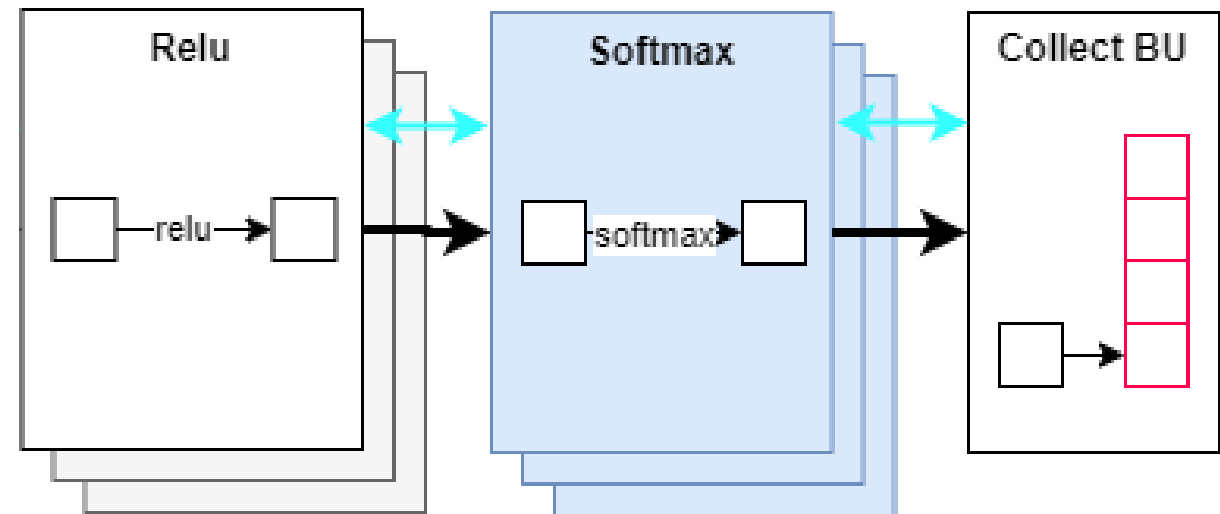


SINGLE ENGINE CORE

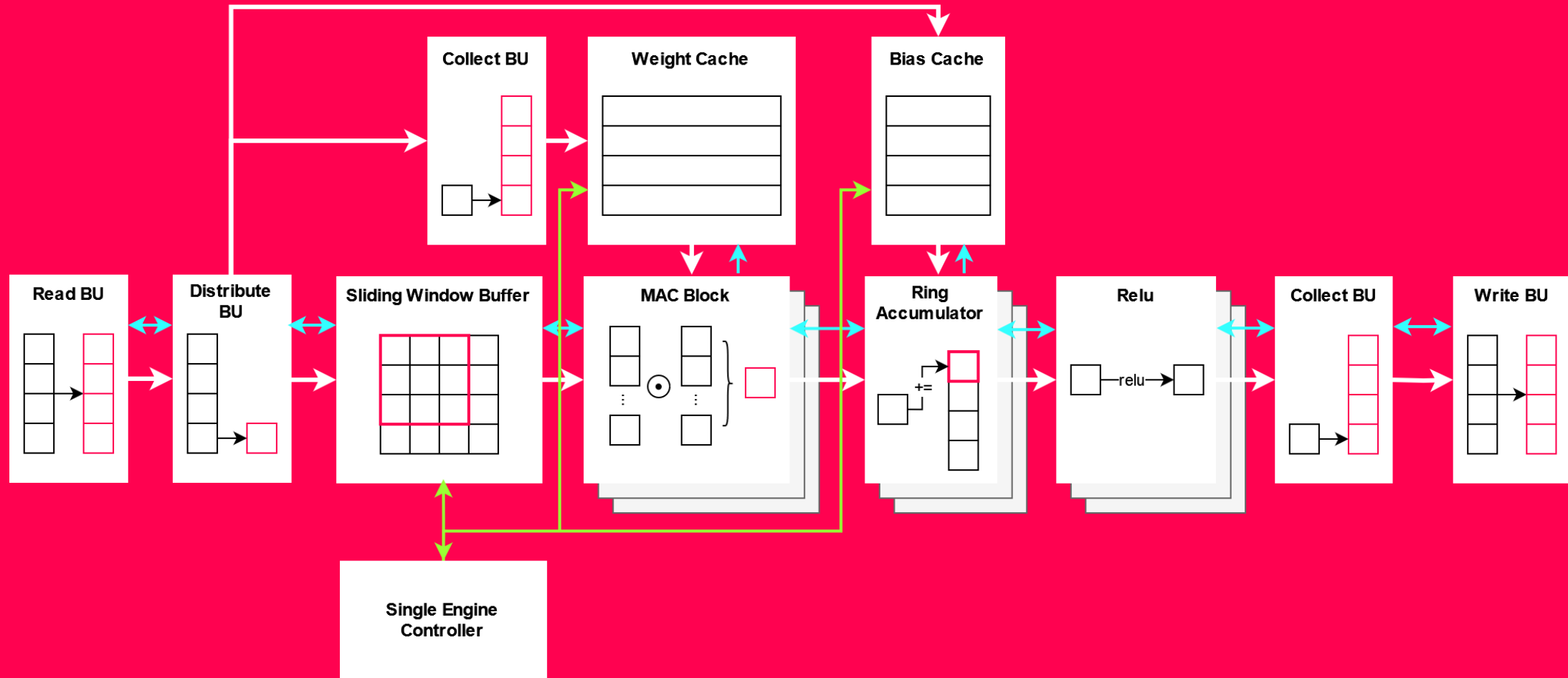


SINGLE ENGINE CORE ERWEITERBARKEIT

- Konfigurierbarere
Einreihung neuer Module



SINGLE ENGINE CORE



SINGLE ENGINE CORE FEATURES

- Berechnung von Fully Connected Layern und Convolutions über dieselben **Hardwareeinheiten**
- Geringer Verbrauch von **Logik Ressourcen**
- **Bus basierte Skalierung**
- Einfach **erweiterbar** dank einheitlicher Interfaces
- **Parametrisiertes** Design

GENERIERUNG DER HARDWARE

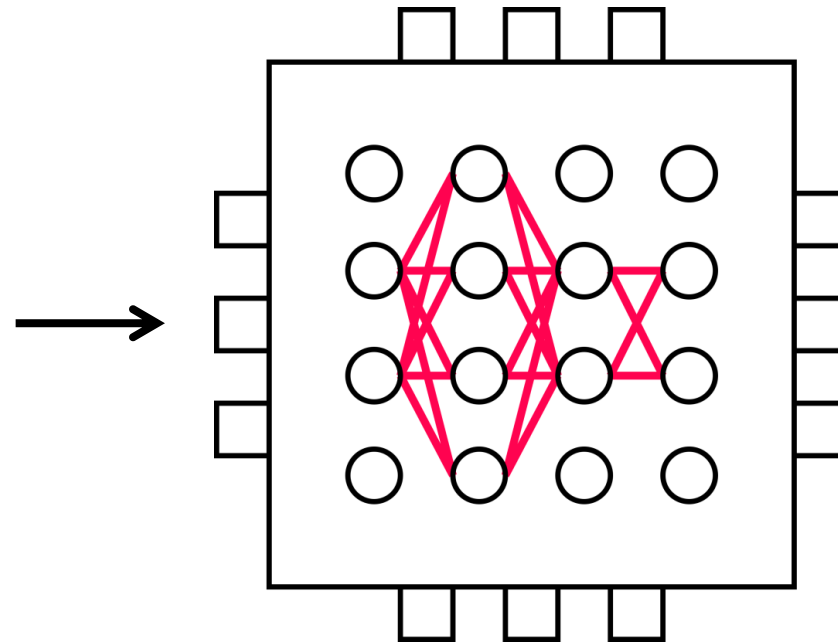
	LeNet	VGG-16
Input Bitbreite	5	7
Gewichte Bitbreite	5	7
Parallelisierungsfaktor	2	4
Bildgröße	28x28	224x224
Filtergröße	5x5	3x3
Speichereinheiten	20	713642
Pipelinetiefe	5	9

1. Neuronale Netze und Optimierung
2. THANNA Quantizer
3. THANNA Prozessor
4. THANNA Treiber

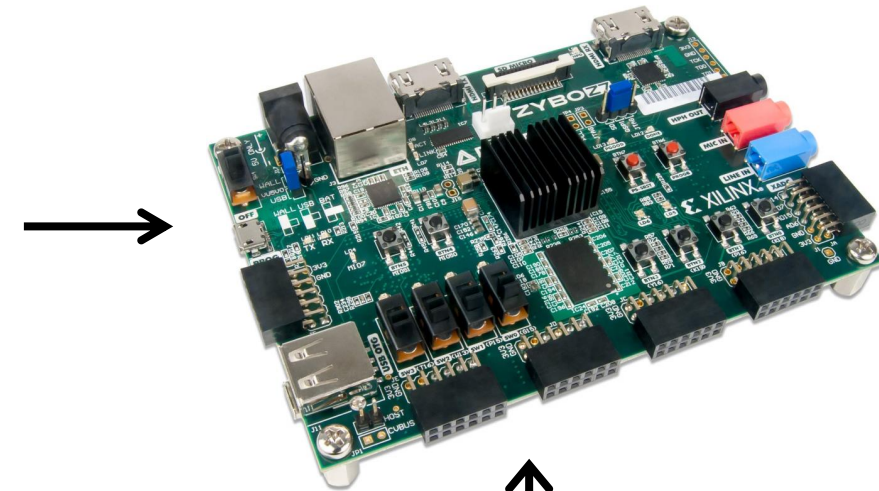
THANNA FRAMEWORK

```
31 self.file = None
32 self.fingerprints = set()
33 self.logdups = True
34 self.debug = debug
35 self.logger = logging.getLogger('THANNA')
36 if path:
37     self.file = open(os.path.join(job_dir, 'THANNA.log'), 'a')
38     self.file.seek(0)
39     self.fingerprints.update(fingerprint)
40
41 @classmethod
42 def from_settings(cls, settings):
43     debug = settings.getbool('debug')
44     return cls(job_dir(settings), debug)
45
46 def request_seen(self, request):
47     fp = self.request_fingerprint(request)
48     if fp in self.fingerprints:
49         return True
50     self.fingerprints.add(fp)
51     if self.file:
52         self.file.write(fp + os.linesep)
53
54 def request_fingerprint(self, request):
55     return request_fingerprint(request)
```

THANNA Quantizer



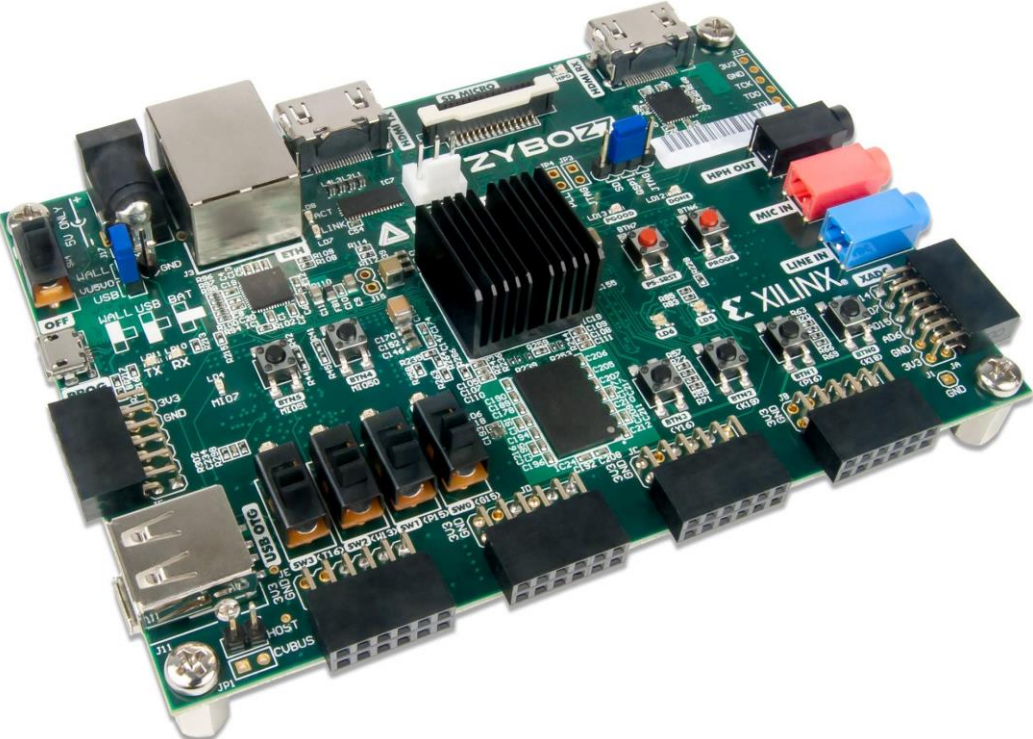
THANNA Prozessor



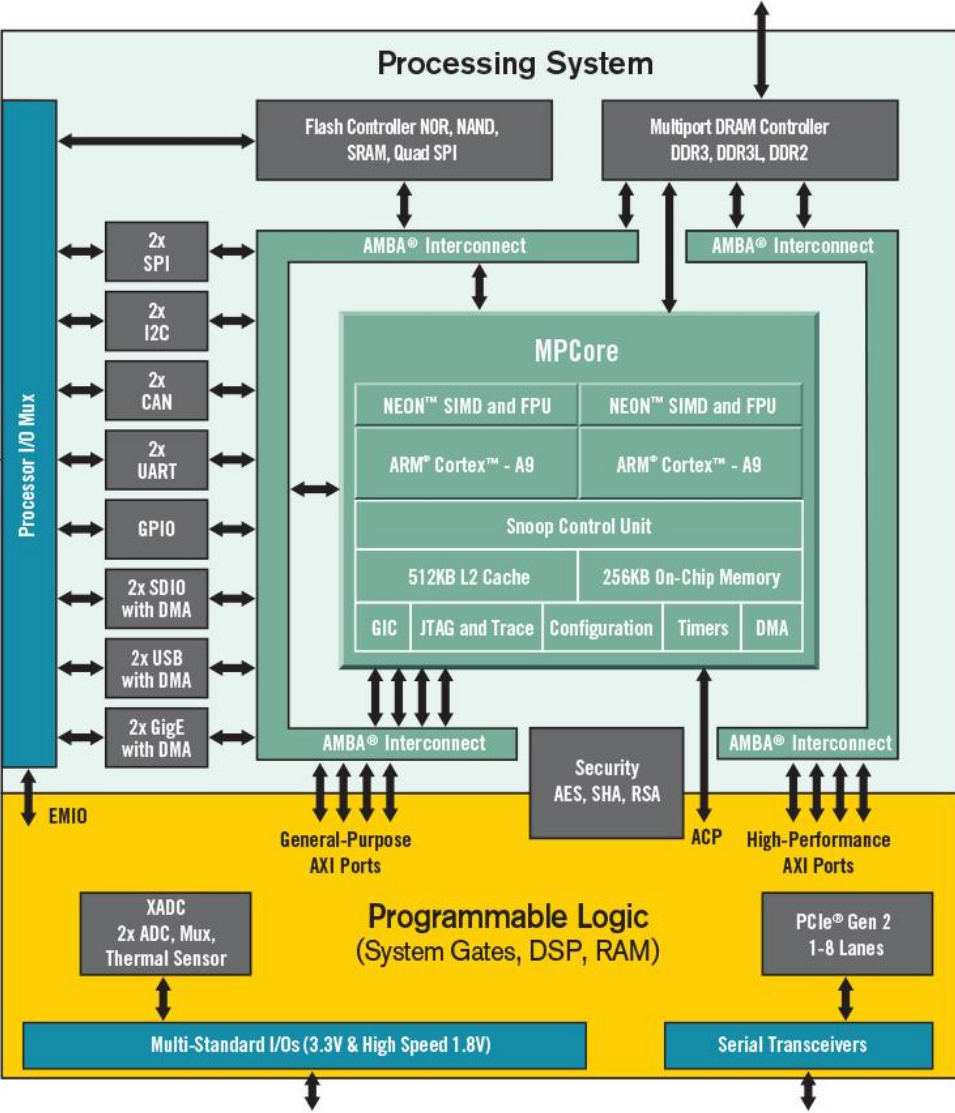
PetaLinux
THANNA Treiber
THANNA Library



DAS ZYBO Z7 MIT ZYNQ-7020



Quelle: digilent.com



Quelle: xilinx.com

WARUM LINUX VERWENDEN?

- Full-Stack Python/Tensorflow Lite
- Treiber für Peripherie (Kamera, Tastatur)
- Einfaches Debuggen der Hardware

→ **Weniger Entwicklungsaufwand**

LINUX TREIBER

KERNEL TREIBER

- Umsetzung über Character-Device
- Steuerung des THANNA-Cores über Register
- DMA für Gewichtsmatrizen



THANNA LIBRARY

- C-Library
- Python Wrapper

DEMONSTRATOR

PROJEKTSTAND

- ✓ THANNA Quantizer aus QKeras Code
- ✓ THANNA Prozessor mit Convolution, Fully-Connected und Relu
- ✓ THANNA Treiber

PROJEKTSTAND

- x Post Training Quantisierung mit Trainingsdaten
- x Pooling & mehr Layer in THANNA Prozessor
- x Pruning Unterstützung in THANNA Prozessor



THANNA

Technische Hochschule Augsburg

Neural Network Accelerator

