Embedded System Technologies for Deep Learning and Approximate Computing

Under the SPARC Project P:271 – "Approximate Computing Techniques for Resource Constrained Edge Devices"

Prof. Alessandro Cilardo

acilardo@unina.it

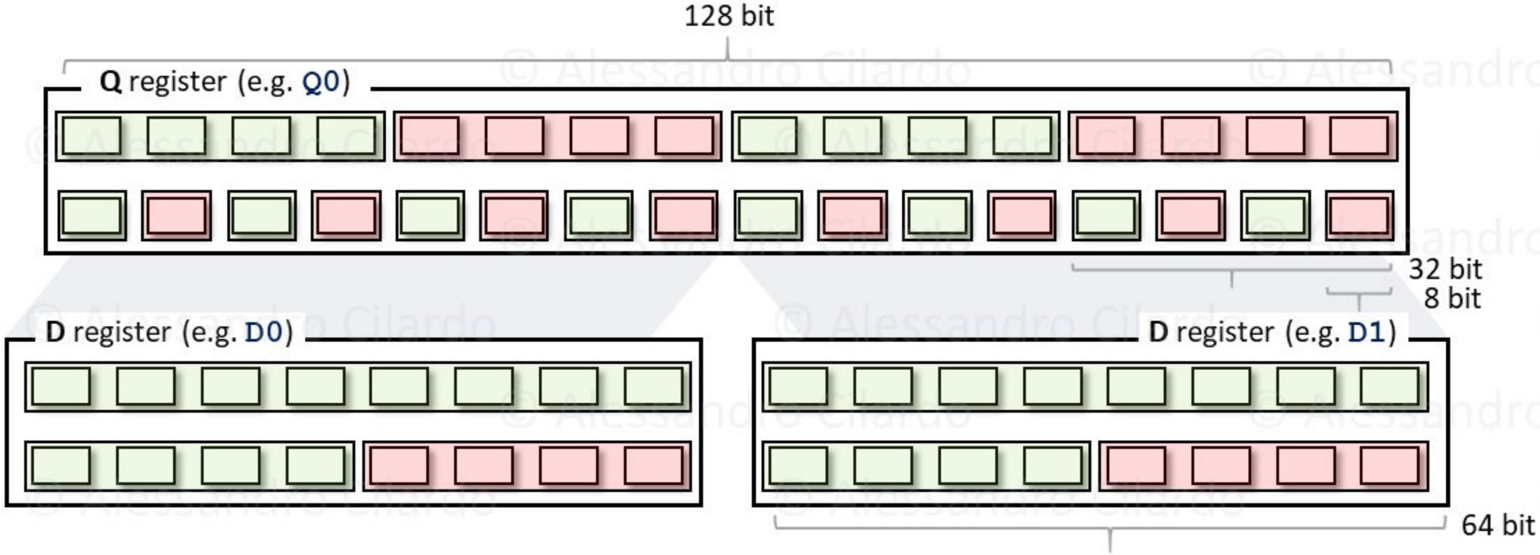
Day 4
June 6th, 2021

Arm Cortex-A-based systems-on-chip: software development

- In the previous classes, we addressed advanced architectural aspects related to Cortex-A systems:
  - e.g. vector extensions (Arm NEON)
  - FPGA-based systems-on-chip relying on Cortex-A9 or Cortex-A53 cores
  - etc..
- Similar to the Cortex-M profile, let's now take a look at the software perspective
  - let's start with a low-level approach
  - ..and then, let's look at software packages which provide an interface to the low-level features

ARM Neon extensions

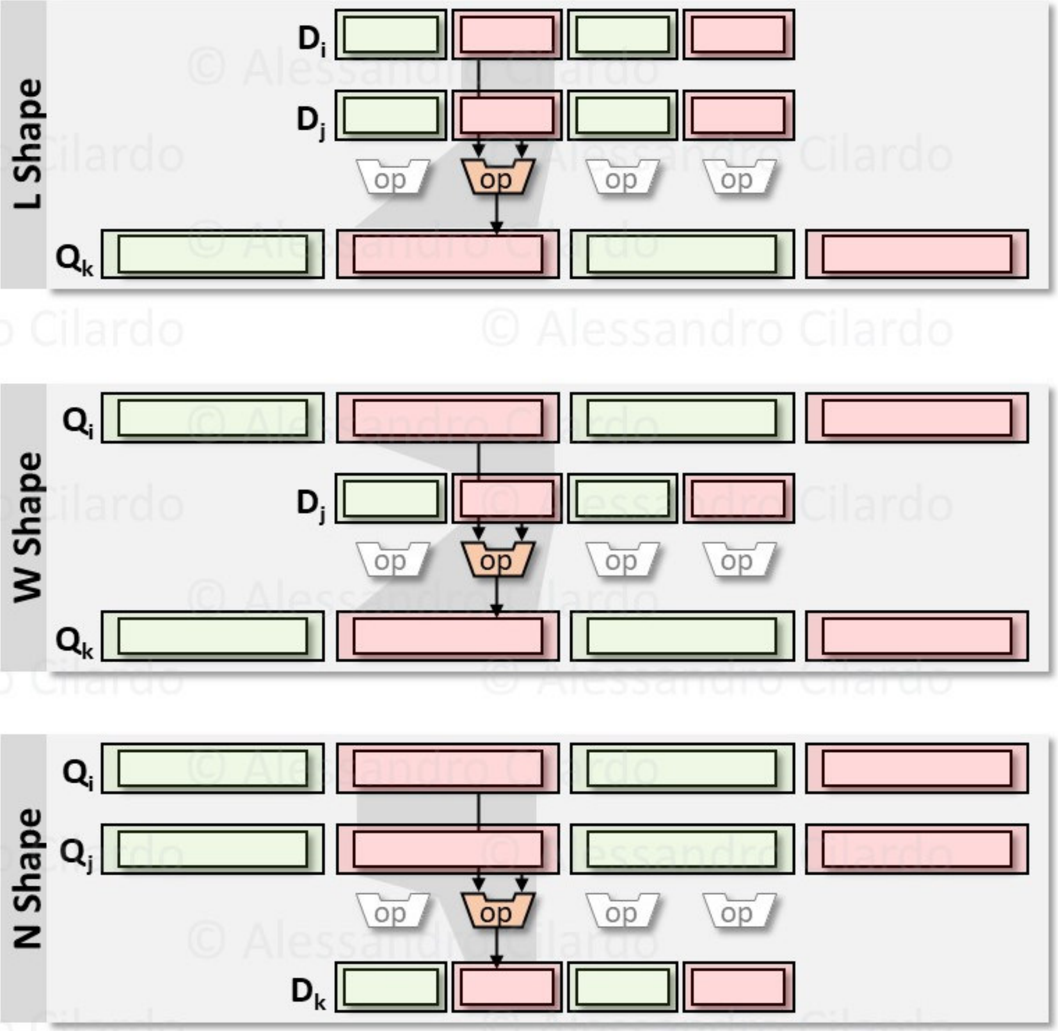
- ARM architectural support for SIMD-like operations (instruction set extensions)
  - ARMv6: introduced support for SIMD instruction (~2002)
  - ARMv7: introduced Neon SIMD extensions (~2009)
  - earlier versions (≤v5) supported Vector Floating Point (VFP) extensions, offering more basic functions
- ARMv7 **Advanced SIMD**, also known as **Neon**
  - 16 additional *128-bit* registers (**Q0-Q15**), which can also be viewed as 32 *64-bit* registers (**D0-D31**)
  - Data types: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit integral types, single precision floating point
  - Additional QC integer saturation summary flag (sticky) in the FP status and control register (FPSCR)



Each of the sixteen 128-bit Neon registers **Q** can be alternatively accessed as a pair of 64-bit registers **D**.
(*Note*: Physically, these name the same physical bits in the Neon register file)

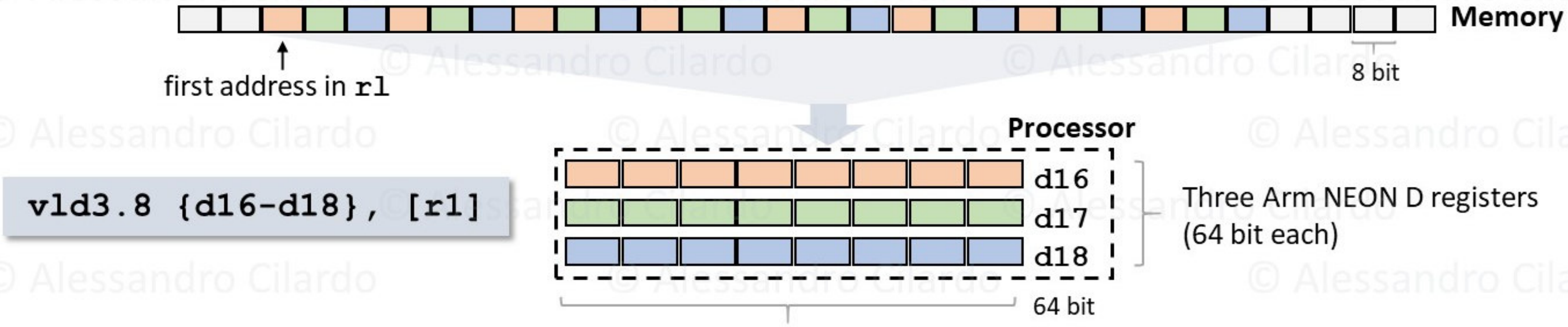
ARM Neon extensions

- Assembly instruction *modifiers* are used to indicate special behaviors:
  - **Q** (operation uses saturating arithmetic, e.g. **`VQADD`**),
  - **H** (operation *halves* the result, e.g. **`VHADD`**),
  - **D** (operation *doubles* the result, e.g. **`VQDMUL`**),
  - **R** (operation performs rounding, e.g. **`VRHADD`**)
- Shape modifiers:
  - **L** (result is double the width of both operands)
  - **W** (result and first operand are twice the width of the last operand)
  - **N** (result is half the width of both operands)
- Instructions also indicate a Conditional (with IT instruction) and a data type



ARM Neon extensions: Loading/Storing arrays of *structures*

- Interleaving/de-interleaving of data to/from memory from/to SIMD registers
- **vld***x.y*: *Structure Load*, **vst***x.y*: *Structure Store* (where *x*: stride, *y*: data size)
  - for example `vld3.8 {d16-d18},[r1]` accesses 8-bit data in memory, strided by 3, and de-interleaves them to the lanes of three **D** registers (3x8 = 24 bytes overall, see the figure below)
- Stride *x* can be one of {1,2,3,4}, data size *y* can be {8,16,32,64}, up to four **D** registers can be indicated as operands, affecting the total number of transferred bytes, e.g.:
  - `vld1.32 {d0},[r0],   vld4.16 {d0,d1,d2,d3},[r0],   vst4.16 {d1,d3,d5,d7},[r10]`
- Can also move a *single element* to a specific lane, e.g.: `vld2.16 {d8[3],d9[3]},[r7]`
- ..or copy the *same* element to *all* the lanes, e.g.: `vld2.32 {d1[],d3[]},[r10:64]`



ARM Neon extensions: vector arithmetic instructions

- Addition, Subtraction (with various modes: Saturating, Halving, Rounding,..)
- Multiplication (VMUL, VMLA, VMLS, VFMA, ..)
- Comparison, Selection (VCEQ, VCGE, VCGT, ..)
- Bitwise logical operations (VAND, VBIC, VEORR, VORN, VORR,..)
  - work on 64-bit and 128-bit registers. Operations are independent of data types
- Bitwise operations: Insert if True, Insert if False, Select (VBIT, VBIF, VBSL)
  - support bit-level manipulation operations. Work on 64-bit and 128-bit registers
- Reciprocal Estimate/Step, Reciprocal Square Root Estimate/Step (FRECPS, ..)
- Further arithmetic operations: MIN, MAX, NEG, MOV, ABS, ABD, ..
- Reduction-like operations (e.g. VPADD: adds adjacent pairs of lanes)
- Miscellaneous: DUP, EXT, CLZ, CLS, TBL, REV, ZIP, TRN, …

ARM Neon extensions used in many software projects..

- Google WebM video codec: around 11,000 lines of code in Neon assembler
- sbc audio encoder in Linux Bluetooth protocol stack (Bluez)
- Pixman library in the cairo 2D graphics library, uses Neon for compositing/alpha blending
- ffmpeg suite, libavcodec codec library (used in LGPL media players of many Linux distros):
  - Neon used for video processing (MPEG-2, MPEG-4 ASP, H.264, VC-1, VP3, Theora, ..) as well as audio processing (AAC, Vorbis, WMA,..)
- Android libraries, e.g. Skia library, S32A_D565_Opaque, ..
  - rely on several Neon optimizations
- Various mathematical and digital signal processing libraries
  - Eigen2 vector mathematics and linear algebra C++ template library
  - Theorarm, also available in a Neon version
  - libjpeg optimized JPEG decode library
  - FFTW Neon-enabled FFT library
- LLVM-based code generation backend in Android Renderscript supports Neon
- An increasing number of ML libraries targeted at ARM systems rely on Neon
- etc ..

SIMD software development

- Optimized libraries
  - e.g. NEON OpenMAX DL (Development Layer): APIs contain a comprehensive set of audio, video and imaging functions that can be used for a wide range of accelerated codec functionality such as MPEG-4, H.264, MP3, AAC and JPEG
  - Broad open source support for NEON
- Vectorizing compilers
  - Exploit SIMD automatically with existing C source code
- Intrinsics
  - C function call interface to SIMD operations
  - Supports all data types and operations supported by the SIMD extensions
- Assembler
  - for those who really need to optimize code at the lowest level

SIMD software development

- vectorizing compilers vs. intrinsics

```
void add_int(int * __restrict pa, int * __restrict pb,
             unsigned int n, int x){
  unsigned int i;
  for(i = 0; i < (n & ~3); i++)
    pa[i] = pb[i] + x;
}
```

```
void add_int(int *pa, int *pb, unsigned n, int x){
  unsigned int i;
    for (i = ((n & ~3) >> 2); i; i--){
      *(pa + 0) = *(pb + 0) + x;
      *(pa + 1) = *(pb + 1) + x;
      *(pa + 2) = *(pb + 2) + x;
      *(pa + 3) = *(pb + 3) + x;
      pa += 4; pb += 4;
    }
}
```

```
. . .
float32x4_t A0;
float32x4_t A1;
float32x4_t A2;
float32x4_t A3;

A0 = vld1q_f32(A);
A1 = vld1q_f32(A+4);
A2 = vld1q_f32(A+8);
A3 = vld1q_f32(A+12);
. . .
```

Arm Compute Library

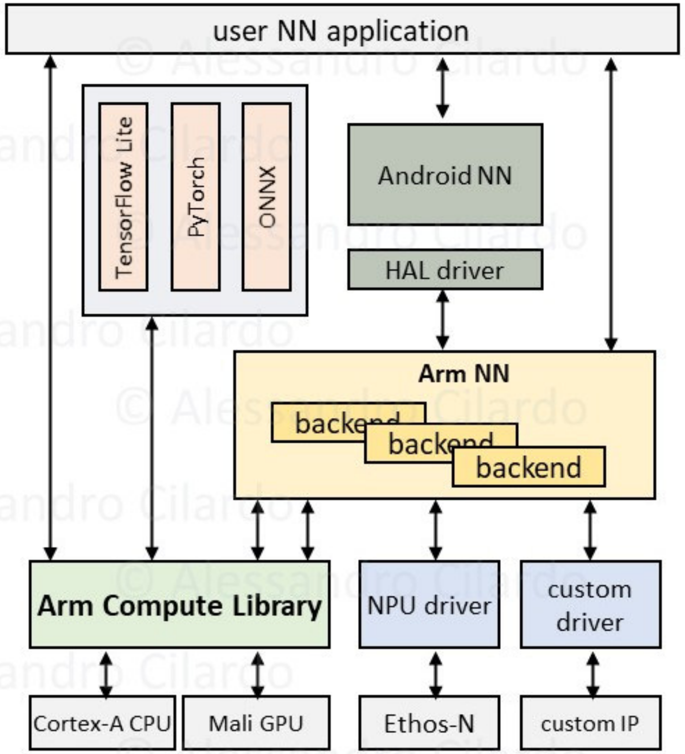
https://github.com/ARM-software/ComputeLibrary

- Low-level machine learning functions optimized for
  - Cortex-A CPU, *including NEON*
  - Mali GPU
- provides superior performance to other open source alternatives and immediate support for new Arm technologies e.g. SVE2
- Key Features:
  - 100+ machine learning functions for Cortex-A CPU and Mali GPU
  - multiple data types supported, i.e. FP32, FP16, int8, uint8, BFloat16
  - multiple convolution algorithms: GEMM, Winograd, FFT and Direct
  - microarchitecture optimization for key Machine Learning primitives
  - configurable build options enabling lightweight binaries
  - advanced optimizations for kernel fusion, fast math enablement, texture utilization, ..
  - OpenCL tuner and GEMM optimized heuristics for device- and workload-specific tuning
- Agnostic to Operating System and portable to Android, Linux and bare metal systems
- Used in countless smartphones, smart cameras, automotive applications, etc.
- Open source software (under permissive MIT license)

Arm NN

https://github.com/ARM-software/armnn

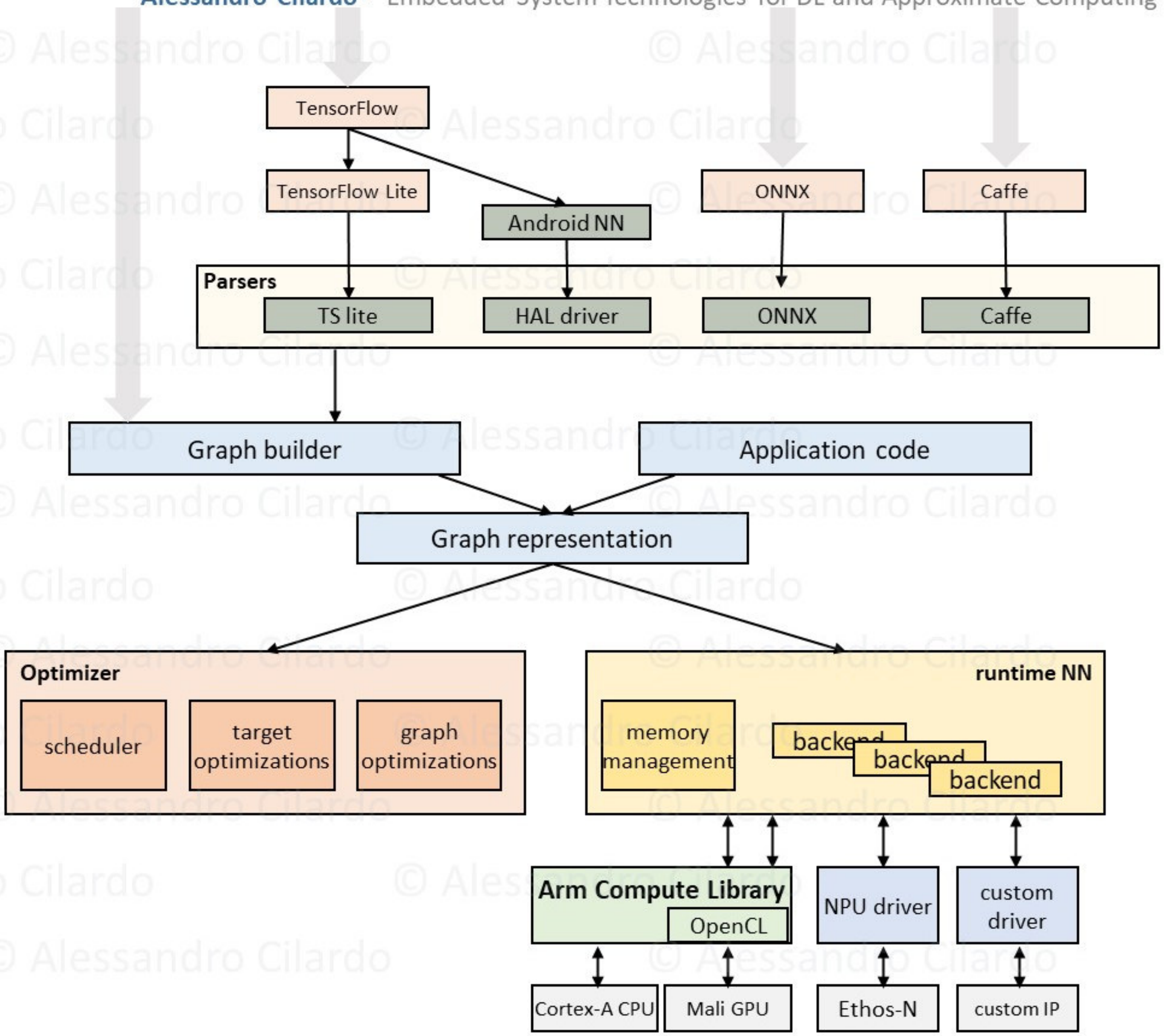
- Provides an interface between popular NN frameworks and Cortex-A CPUs, Arm Mali GPUs, or Arm Ethos NPUs
  - relies on the Arm Compute Library for exploiting Cortex-A CPUs (v7, v8) and Mali GPUs
  - custom IP and associated *backend* can be provided
  - (backend can be dynamically loaded)
  - Ethos-N Driver Stack for the NPU IP (yet not available commercially)
- Can also interface with Google's Android NNAPI
  - uses the HAL driver targeted at the Arm IP
- Takes in a given model and replaces the underlying operations with optimized versions for the target platform
  - scripts used for translation are part of the framework
  - (can accept TensorFlow Lite, ONNX, PyTorch, and Caffe models)
- Open-source software and tools
  - written in portable C++14



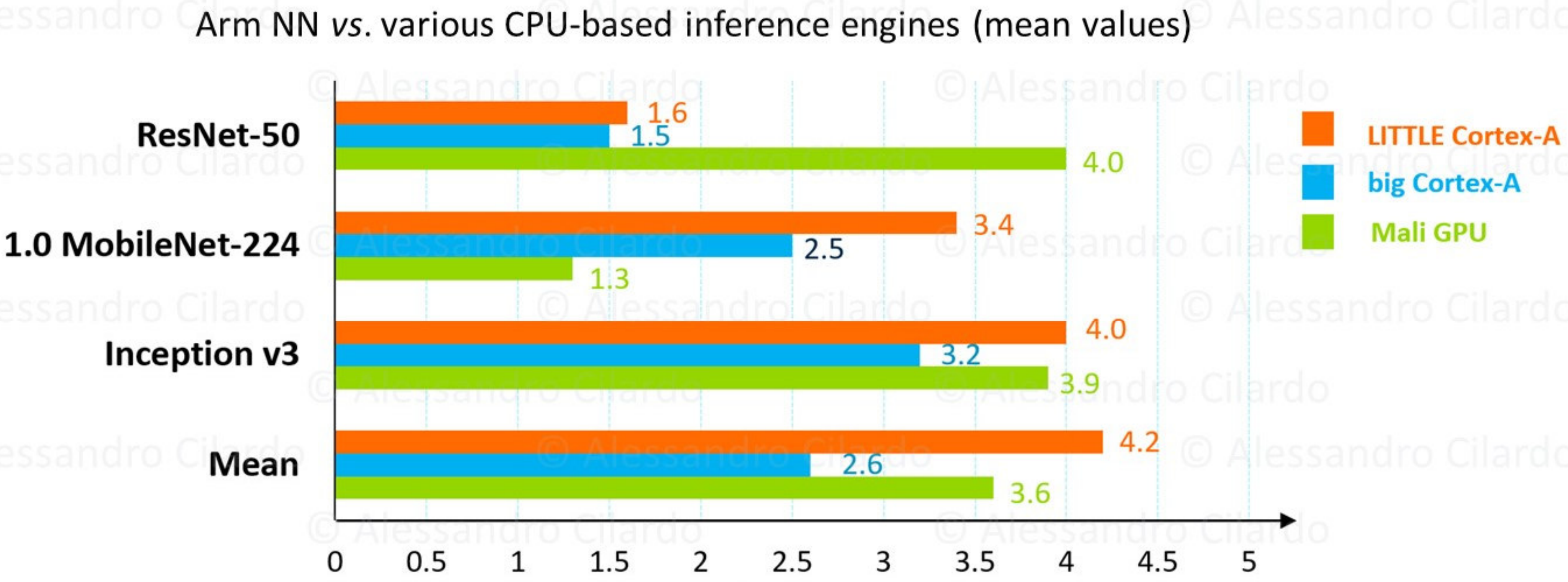
- *Note* (1): only inference is supported
- *Note* (2): no support for Cortex-M

Arm NN: modes of use

- Three modes of use supported:
  - Direct use of the graph builder API
  - ML frameworks (through parsers)
  - Android NN API integration



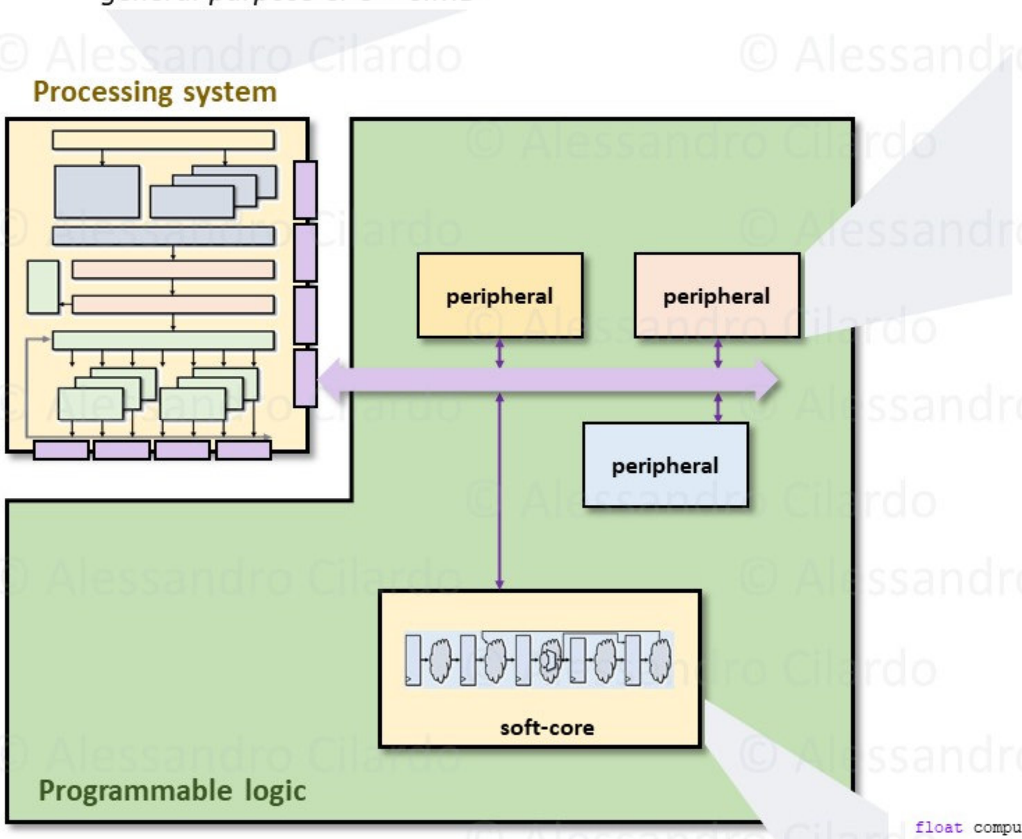
Arm NN

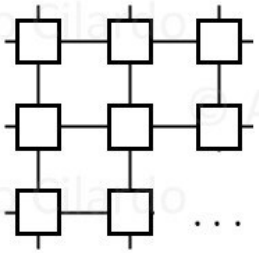


Source: https://github.com/ARM-software/armnn

FPGA design flows

general-purpose CPU + SIMD



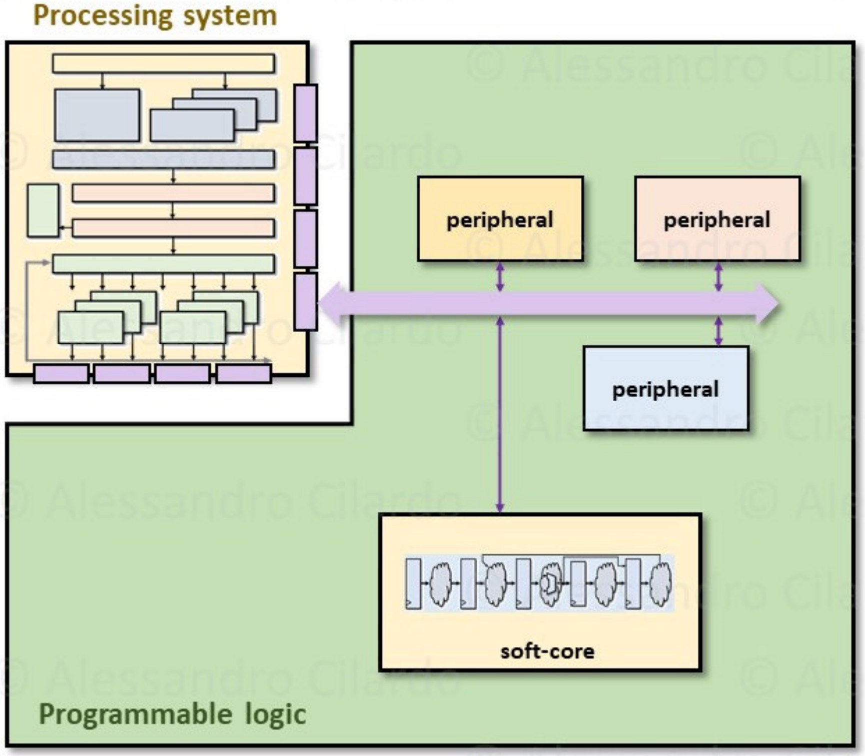


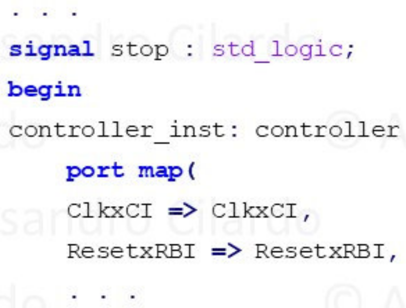
for example, some layer of a Binarized NN or a Quantized NN

control logic written in C logic running on a custom soft core

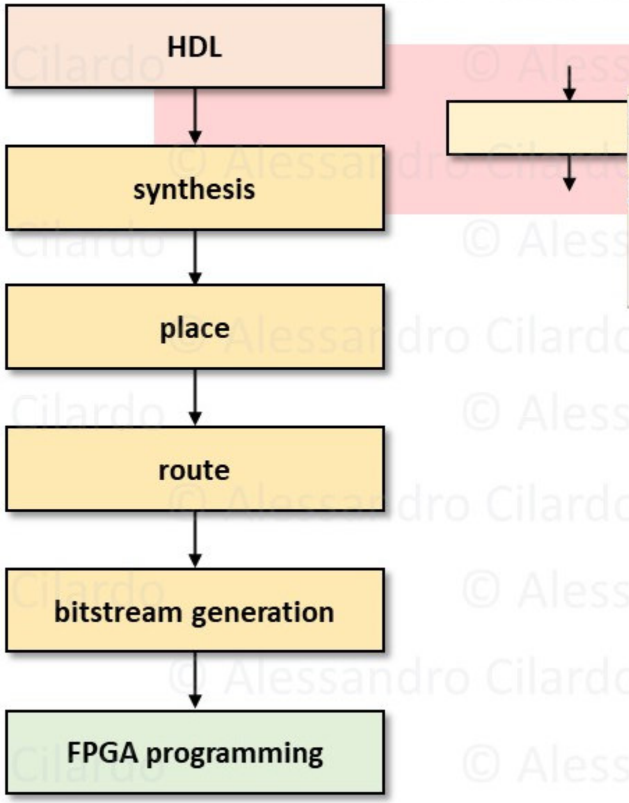
```
float computeNeighbor(float v){
    if (v==0) return 0.0;
    int sgn=false;
    if (v<0) { v*=-1; sgn=true; }
    int e;
    . . .
```

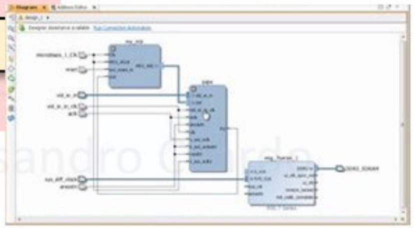
FPGA design flows





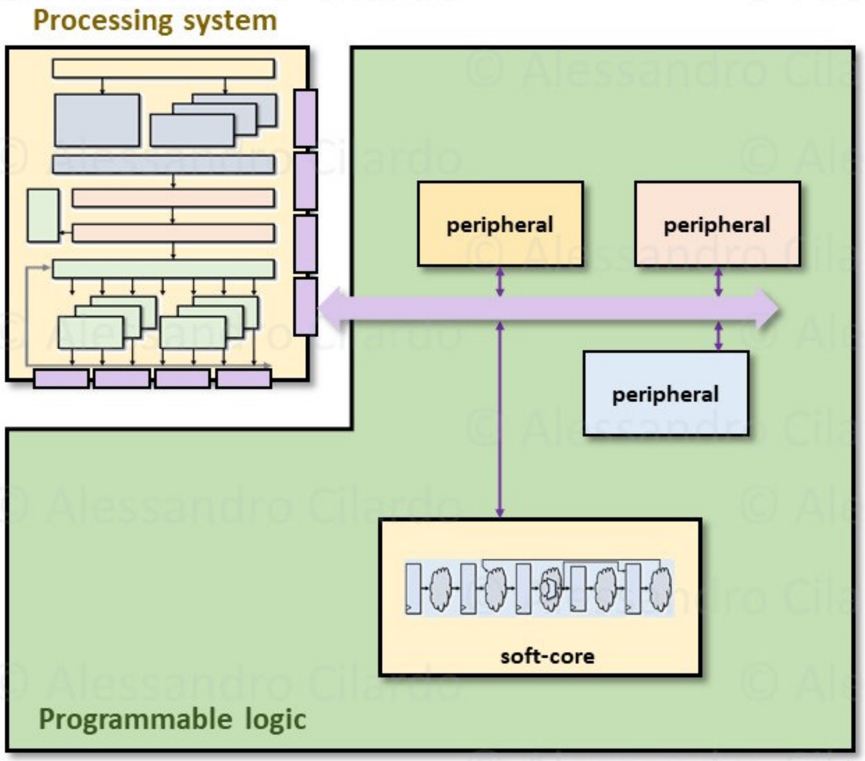
```
. . .
signal stop : std_logic;
begin
controller_inst: controller
    port map(
    ClkxCI => ClkxCI,
    ResetxRBI => ResetxRBI,
    . . .
```

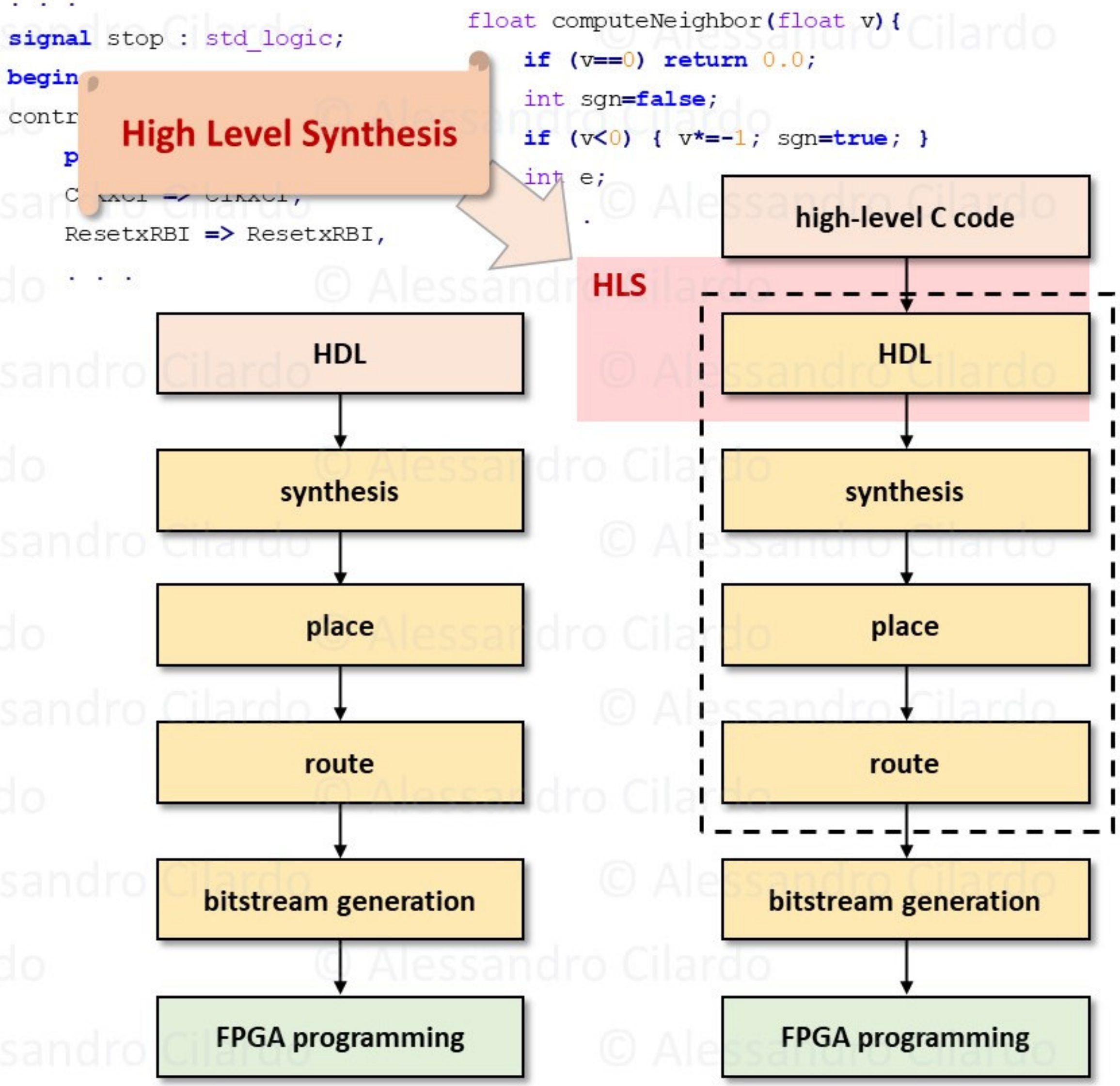




IP integration

FPGA design flows

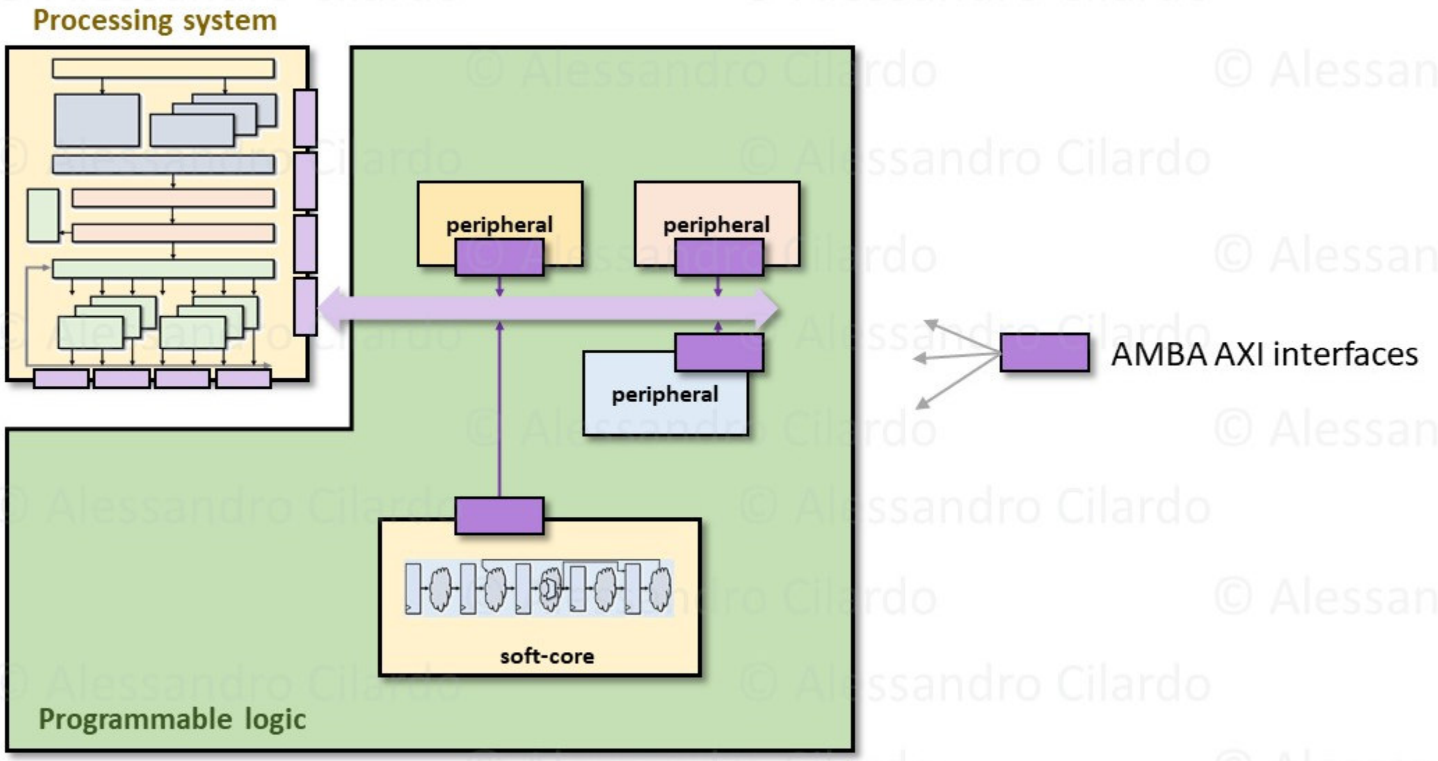




```
. . .
signal stop : std_logic;
begin
contr
   p
   ClkxCI => ClkxCI,
   ResetxRBI => ResetxRBI,
   . . .
```

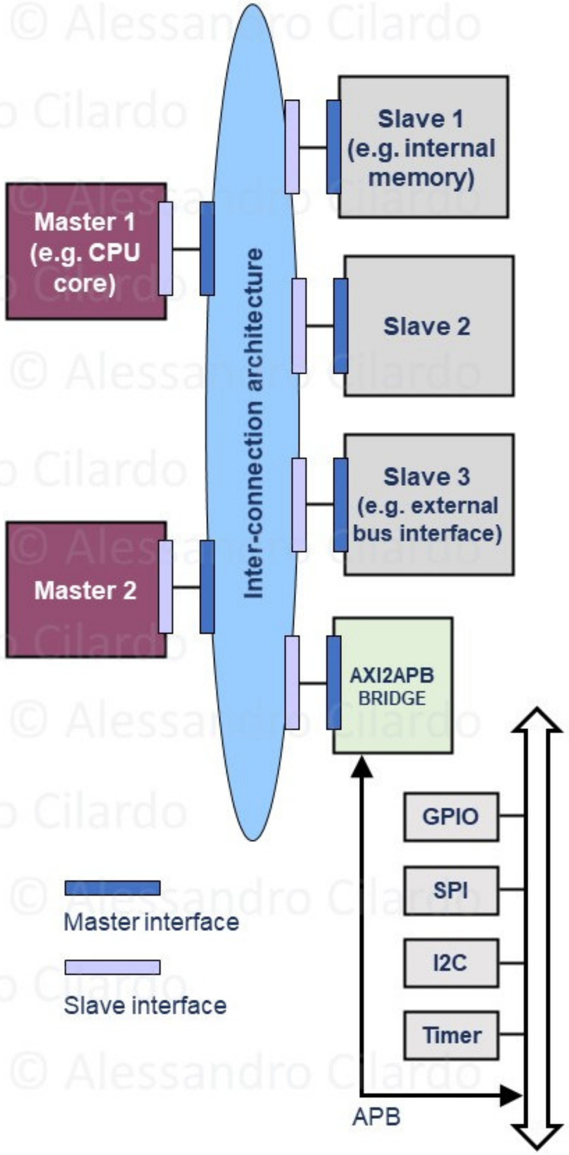
```
float computeNeighbor(float v){
   if (v==0) return 0.0;
   int sgn=false;
   if (v<0) { v*=-1; sgn=true; }
   int e;
   .
```

FPGA design flows



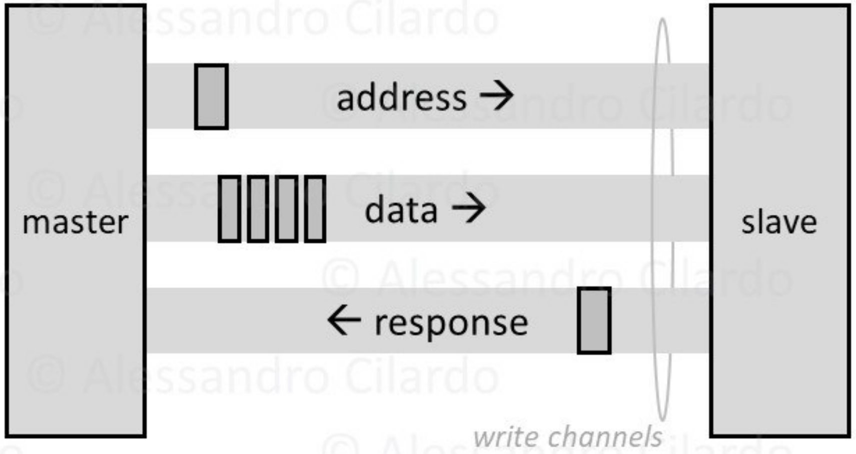
AMBA 3 and 4: Advanced Extensible interface (AXI)

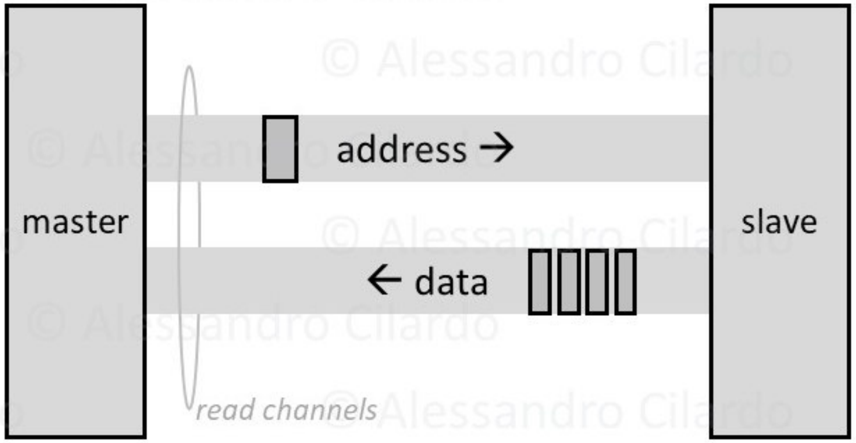
- Advanced Extensible interface (AXI)
  - used for high bandwidth and low latency interconnects
  - *point-to-point* interconnect, avoid the limitations of a shared bus
  - the "interconnect" can be a custom crossbar or a switch design or even a *Network on Chip* (NoC)
  - enhances AHB: supports multiple outstanding data transfers (pipelined), burst data transfers (up to 256 cycles), separate read/write paths and different bus widths
- AXI-lite protocol: a simplified version of AXI
  - no support for burst data transfers
- AXI-stream protocol
  - supports only streaming of data from a master to a slave
  - no separate read/write channels in the stream protocol unlike a full AXI or AXI-lite
  - only streams in one direction
  - multiple streams can be transferred, possibly with interleaving, across a master and slave, e.g. for video streaming applications.



Detail of the AXI protocol

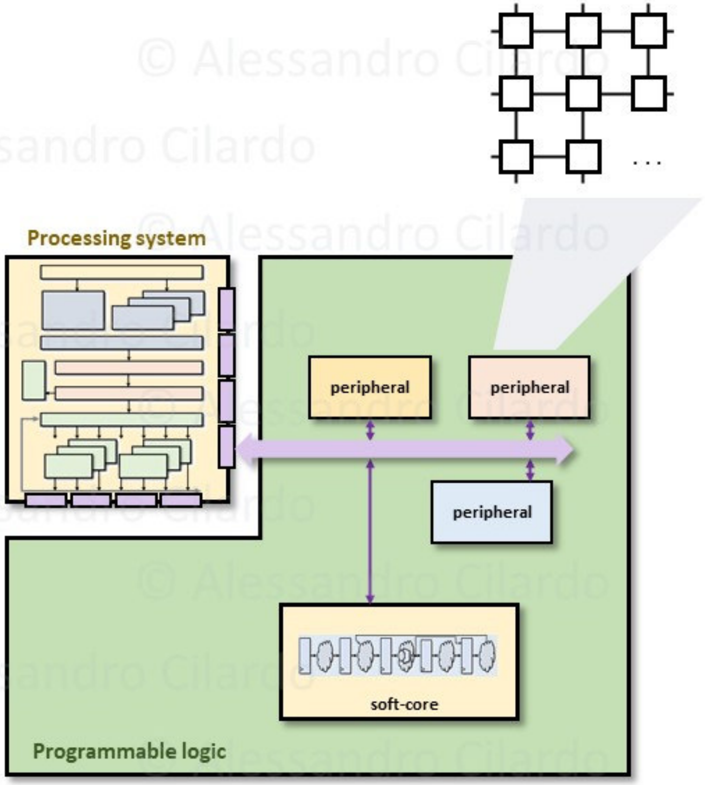
- Protocol based on independent "*channels*"
- Address Channels (R/W)
  - separate for reads and writes, provide all the address and control information regarding a transaction. They control:
  - bursts, varying from 1 to 16 data transfers per burst
  - burst transfer size: 8-1024 bits
  - wrapping, incrementing and non-incrementing bursts
  - caching and buffering control at the system-level
  - atomic operations with exclusive or locked accesses
  - secure and privileged access
- Write Data Channel
  - a data bus ranging from 8-1024 bits wide
  - a byte lane strobe for every eight bits of data, used for identifying valid bytes in the data bus
- Read Data Channel
  - includes a data bus with an identical range but also a read response which indicates the completion of a read transaction
- Write Response Channel
  - allows signalling of a completion to a write transaction from the slave by sending a completion signal once for each burst





Challanges for porting an NN to an FPGA

- Models may be extremely demanding
  - e.g. ResNet-50 requires +7 billion operations for every input image
  - but, operations are highly parallel:
  - parallel branches and topology parallelism across consecutive layers and (GoogLeNet, DNN ensambles, ..)
  - within a layer: multiple input/output feature map channels and pixels in convolutional layers
  - bit-level parallelism
- Approaches to FPGA acceleration:
  - optimized "streaming" custom architecture
  - reusable processing engine /vector processor



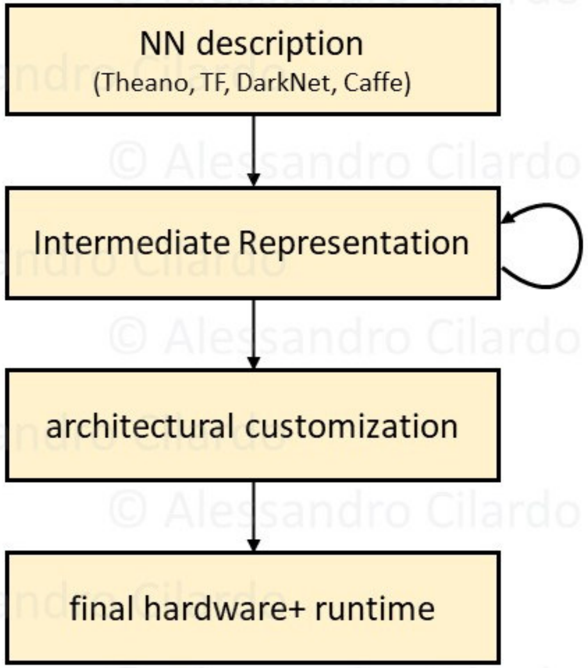
Xilinx FINN-R experimental framework (*)

- Support architectural choices and mixed/variable precisions beyond binary
- Feed-forward dataflow implementations for fully binarized neural networks
- The flow relies on a quantization-aware *intermediate representation*
  - enables QNN-specific optimizations
- Originally only supported binary quantization for input and weight values
  - resort to XNOR and bitcount (a.k.a. Hamming weight, or popcount) for multiply-accumulate
  - only feed-forward dataflow (DF)
- Now, supports Multilayer Offload (MO)
  - only some of the layers are implemented in the PL
  - automatically generates a runtime schedule for the MO case
  - identifies an optimized folding parameter for the DF case
- Templated HLS library + examples
  - implements convolutional, fully-connected, pooling and LSTM layer types as streaming components

(*) T. Preusser, G. Gambardella, N. Fraser and M. Blott, “Inference of Quantized Neural Networks on Heterogeneous All-Programmable Devices”, *Proceedings of the Design Automation & Test in Europe Conference & Exhibition* (DATE), Dresden, Germany, March 2018.

Xilinx FINN-R experimental framework (*)

- LLVM-based flow
- Front-end
  - responsible for interfacing with a selection of training frameworks (Caffe, DarkNet and Tensorflow) and translating trained QNNs into the IR
- Quantization-aware intermediate representation (IR) used for performance modelling
- Back-end: code generation


NN description
(Theano, TF, DarkNet, Caffe)
Intermediate Representation
architectural customization
final hardware+ runtime

(*) T. Preusser, G. Gambardella, N. Fraser and M. Blott, "Inference of Quantized Neural Networks on Heterogeneous All-Programmable Devices", *Proceedings of the Design Automation & Test in Europe Conference & Exhibition* (DATE), Dresden, Germany, March 2018.