Reliability Analysis of Hardware Accelerators through Compiled Cycle-Based Simulation

Davide Bertozzi

University of Manchester

Collaboration with
Alessandro Veronesi
IHP Microelectronics (Germany)

TWINRELECT
TWINNING FOR EXCELLENCE IN RELIABLE ELECTRONICS

UK Research and Innovation

Funded by the European Union

AI Deployment Scenarios

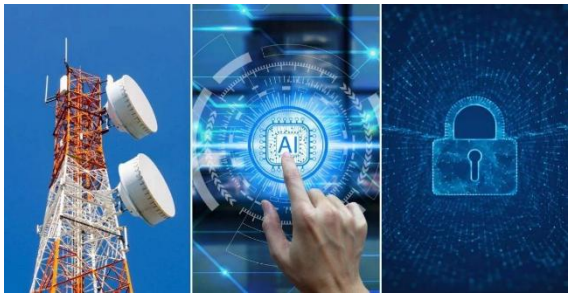
Autonomous driving

AI-assisted surgery

Aerospace & space systems

Industrial automation

Reliability



Telecommunications

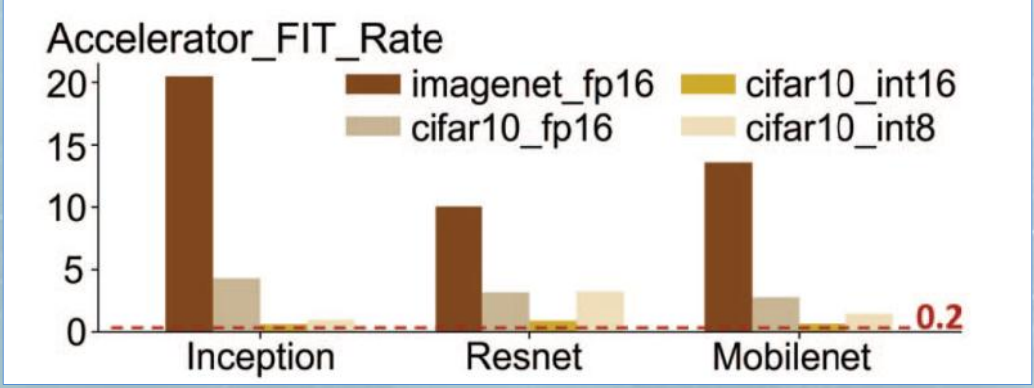
Defense & Military systems

Smart cities

Is it a Problem?

FALSE MYTH: DNNs are inherently robust and reliable, but….

Global control logic protected

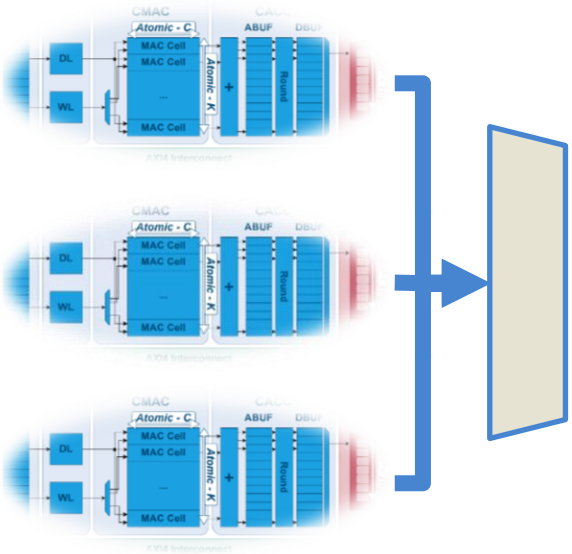


Yi He, Prasanna Balaprakash and Yanjing Li
«FIdelity: Efficient Resilience Analysis Framework for Deep Learning Accelerators», *In Proceeding of MICRO 2020.*

FIT rates of AI hardware accelerators can exceed safety standards, which shows that reliability and error recovery is of paramount concern

Replication

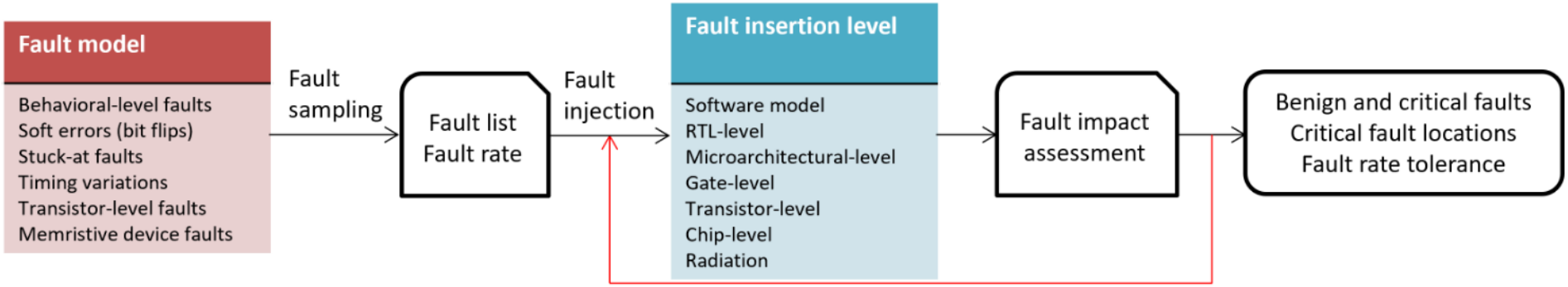
Traditional hardening solutions, based on replication, might not be efficient for DNNs.

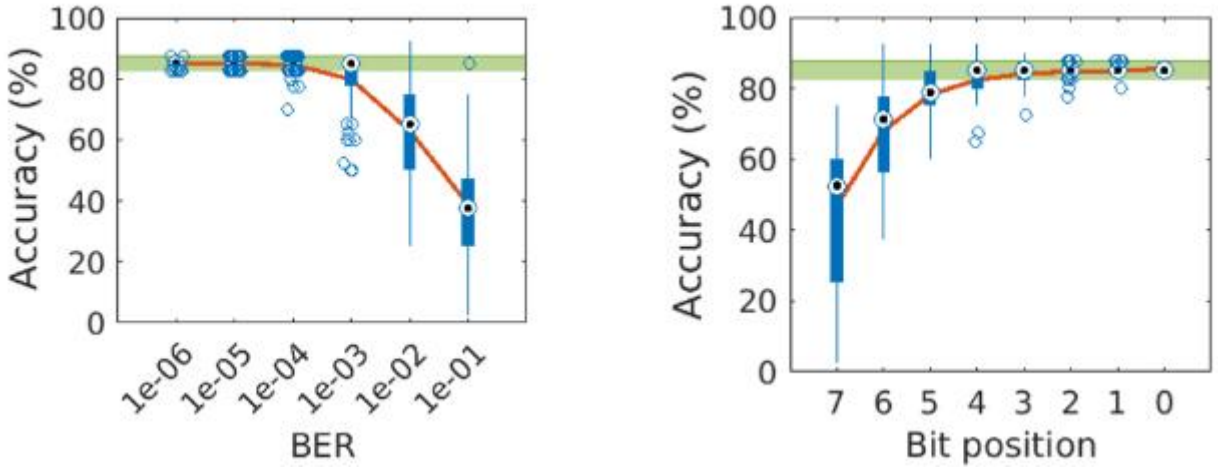


- They fail to meet the primary design goals.
- Not all faults are critical for DNNs.

A reliability analysis is a key enabler toward effective and tailored hardening approaches

Spotting Critical Faults – Fault Injection Flow –

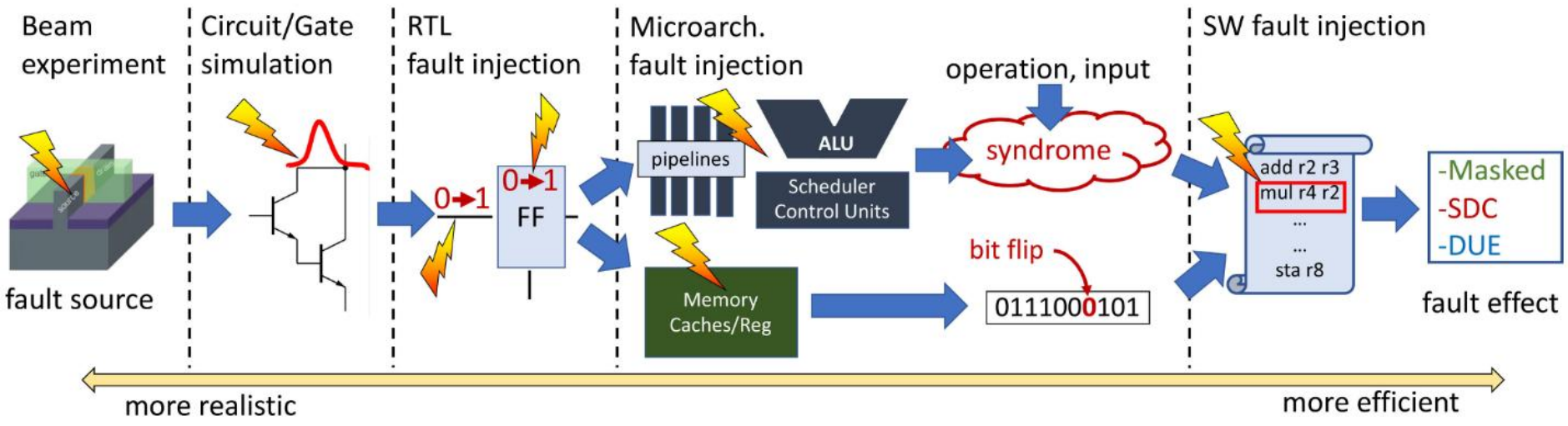




Reliability assessment using bit flips as fault models

What Abstraction Layer?

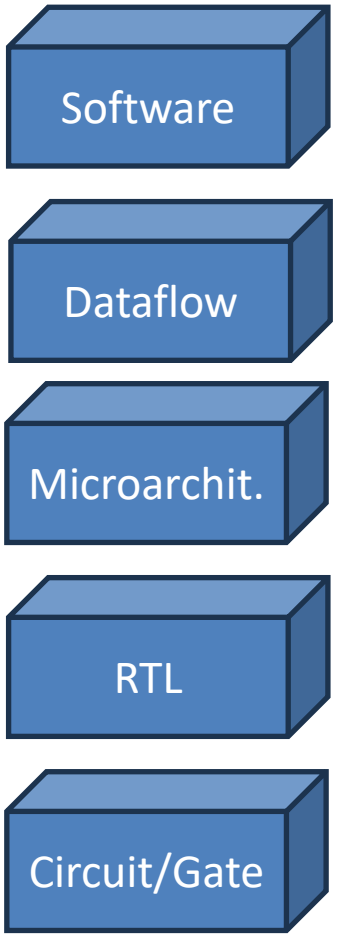
There are various reliability evaluation methodologies to understand fault propagation in computing devices



Abstraction closer to the physical source of the fault

Abstraction closer to the output manifestation of the fault

Hardware Simulation Techniques



Software imperative languages

PROS

- Overly fast simulation times
- Application metrics

CONS

- No microarch. State
- Only memory fault models accurate

Augment C/ simulation t… for flexibility …-driven …bstraction

Heavy event

Poor support

…eduler

variables and faulty m… …oarchitectural states?

Event-driven simulators

PROS

Support for hardware semantics

CONS

Demanding execution times

Accelerator Software Modeling Issues

There is currently no consolidated approach to software modeling of DLAs for reliability analysis

- Software models may not be able to pinpoint all candidate fault locations and to capture fault effects.
- They require accurate fault models from lower abstraction layers (e.g., RTL, gate,..)
- They require accurate post-processing with information from other abstraction layers

What kind of microarchitectural details shall we expose to the software model?

How far shall we go with software modelling?

Up to competing with event-driven simulators on their ground (RTL)?

Outline

Reliability analysis of a DLA datapath

Selecting a suitable software modelling abstraction

Cycle-based simulation for <u>fast and complete</u> reliability analysis of a DLA

Outline

Select a suitable software modelling abstraction

Case Study: NVDLA

NVIDIA Deep Learning Accelerator is an open-source Convolutional Neural Network Accelerator for Inference operations.

- **Open DLA architecture** (RTL, Software Environment, Virtual Prototyping platform)
- **Highly parametrized** and modular, both in hardware and software
- **Industry-level design** (Verified and implemented)

NVIDIA

NVDLA.org

Jetson AGX Xavier

Robotics, autonomous vehicles

Jetson Xavier NX

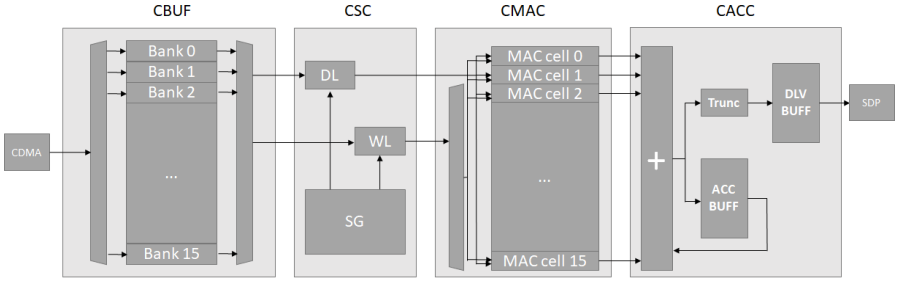
Drones, robots, edge AI

NVIDIA Drive AGX

Autonomous vehicles

Our focus will be on **SEU injection** into the **Datapath of NVDLA**

- **Convolutional Input Buffer** (CBUF)
- **Data Loading Units from buffer to MAC array** (CSC)
- **MAC pipeline** (CMAC)
- **Accumulation Unit** (CACC)



How to simulate this?

Simplest Software Model: Hardware-Agnostic

- Capturing DNN algorithmic aspects regardless of hardware implementation
- Performing fault injection into weights, feature data and neuron outputs

Algorithm 1 Algorithm-oriented model for FC layer.

```
1: wt_mat ← Weight Matrix
2: ft_vec ← Feature Vector
3: S ← ft_vec length
4: for all k ∈ MAC_cells do
5:     for i : 0 → S − 1 do
6:         cacc[k] += ft_vec[i] × wt_mat[k][i]
7:     end for
8:     dlv[k] = cacc[k]
9: end for
```

Datapath: DNN layer's Inputs and Outputs are injectable;

Local Control Path: Faults modelled as randomization of affected neuron outputs;

Global Control Path: Overlooked.

- Very common
- Essentially models memory errors
- Captures the algorithmic execution flow, not the accelerator execution dataflow

One Step Further: Dataflow Visibility

- Next step: a timed-functional scheduling-oriented MAC array model
- Computation scheduling on the available hardware neurons (MAC cells)
- The model gains visibility of elementary hardware atomic operations

Algorithm 2 Scheduling-oriented model for FC layer.

```
1:  wt_mat ← Weight Matrix
2:  ft_vec ← Feature Vector
3:  Atomics ← Number of Atomic Operations
4:  Muls ← Number of Multipliers per each MAC cell
5:  for all k ∈ MAC_cells do
6:     for atm : 0 → Atomics − 1 do
7:        for i : 0 → Muls − 1 do
8:           psums[k] += ft_vec[i] × wt_mat[k][i]
9:        end for
10:       cacc[k] += psums[k]
11:    end for
12:    dlv[k] = saturation(cacc[k] ≫ Shift)
13: end for
```

- Key microarchitectural state captured by software variables

But:

- Real datapaths are more elaborated
- Local control path grossly oversimplified

Datapath: Partial Results are injectable; Modelling of the Accumulation Unit and of the delivery buffer;
Local Control Path: randomization of affected neuron outputs;
Global Control Path: Overlooked.

One Step Further: More Microarchitectural State

- Modelling of retiming components along the datapath (MAC pipeline model)
- Local control path accurately modelled

Algorithm 3 Hardware-oriented model for FC layer.

```
1:  wt_mat ← Weight Matrix
2:  ft_vec ← Feature Vector
3:  Atomics ← Number of Atomic Operations
4:  Muls ← Number of Multipliers per each MAC cell
5:  rt_num ← Retiming registers in the adder tree
6:  for all k ∈ MAC_cells do
7:    for atm : 0 → Atomics − 1 do
8:      for i : (atm × Muls) → ((atm+1) × Muls) − 1 do
9:        ft_load[i] = ft_vec[i]
10:     end for
11:     for i : (atm × Muls) → ((atm+1) × Muls) − 1 do
12:       wt_load[i] = wt_mat[k][i]
13:     end for
14:     for i : 0 → Muls − 1 do
15:       mul_out[k] += ft_load[i] × wt_load[i]
16:       rt_reg[i ÷ rt_num] += mul_out[i]
17:     end for
18:     for j : 0 → rt_num − 1 do
19:       psums[k] += rt_reg[j]
20:     end for
21:     cacc[k] += psums[k]
22:   end for
23:   dlv[k] = saturation(cacc[k] ≫ Shift)
24: end for
```

- Still a dataflow model (only suitable for *what-if* analysis)
- Not an RTL model yet:
  - ✓ Algorithmic step is the atomic hardware operation, not the clock cycle
  - ✓ No hardware concurrency
  - ✓ Some RTL registers hidden
- Not suitable to model global control or variable lifetimes

Datapath: Added Pipeline and Retiming registers, which are potential fault locations

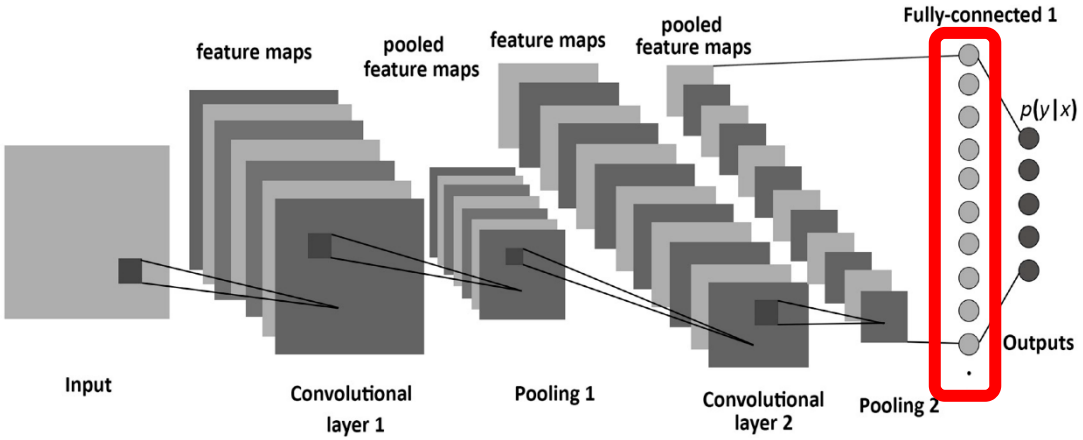
Local Control Path: Accurate modelling of buffer addressing and valid signals; local control signals are potential fault locations

Global Control Path: Overlooked.

Which Reliability Analysis with each Model?

TESTBENCH

For the resilience analysis of the MAC pipeline, we used a Fully-Connected (FC) layer from a CNN model based on LeNet

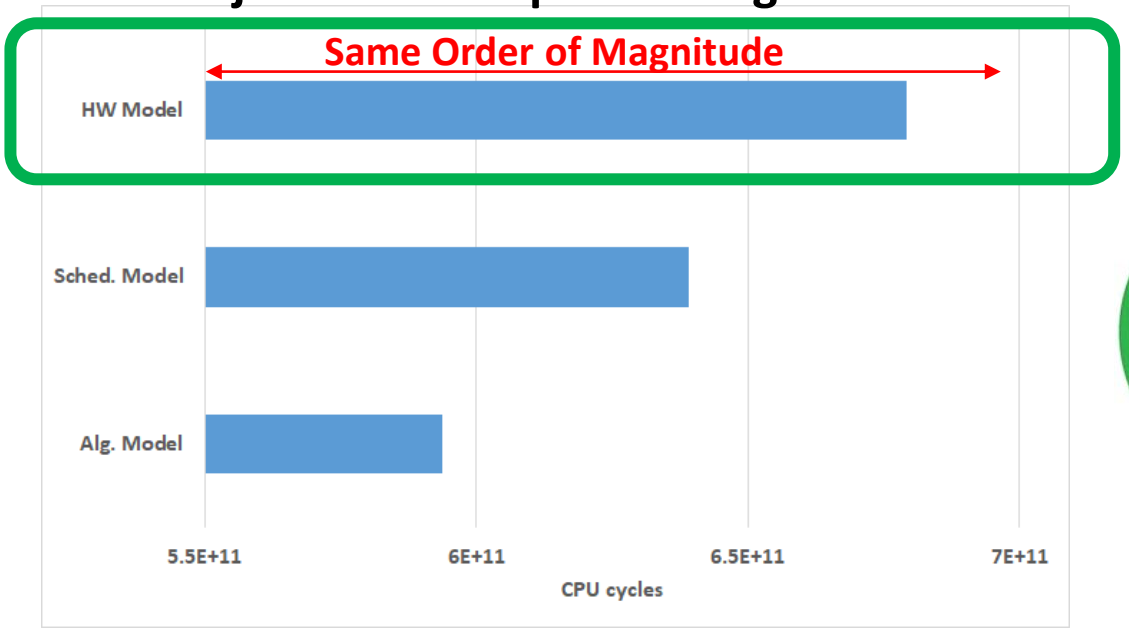


- 1 hardware atomic operation: 64 multiplications per hardware neuron
- Processing the target 1024-inputs FC layer takes place through 16 consecutive Atomic Operations in each used MAC cell.

- **LeNet variant for CIFAR-10 trained with FP32**
- **Post-training INT16 uniform and symmetric quantization**
- **Fully Connected layers turn out to be the most sensitive layers** in the network (cf. previous work)

Execution Time

Exhaustive fault injection while performing inference on one sample

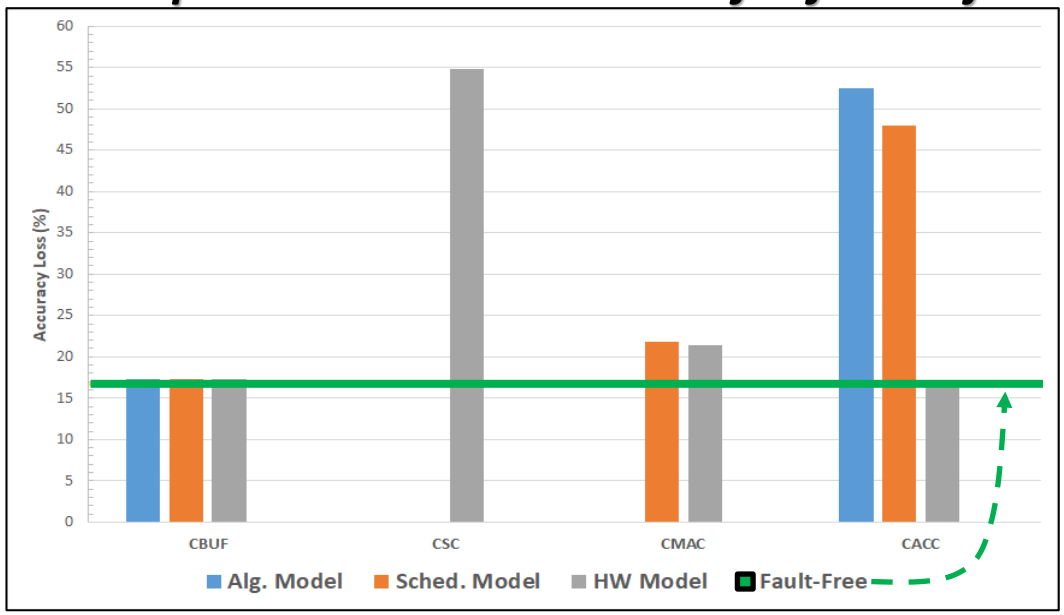


Intel® Xeon® E5-2637 CPU with a 128GB DRAM.

- Simulation time dominated by fault injection into CBUF (99% of Alg. Model)
- Without buffer injection, Sched. Model takes 51x the time of Alg. Model
- HW model causes less than 2x overhead over Sched. Model

What is the accuracy drop due to a fault in an accelerator sub-unit?

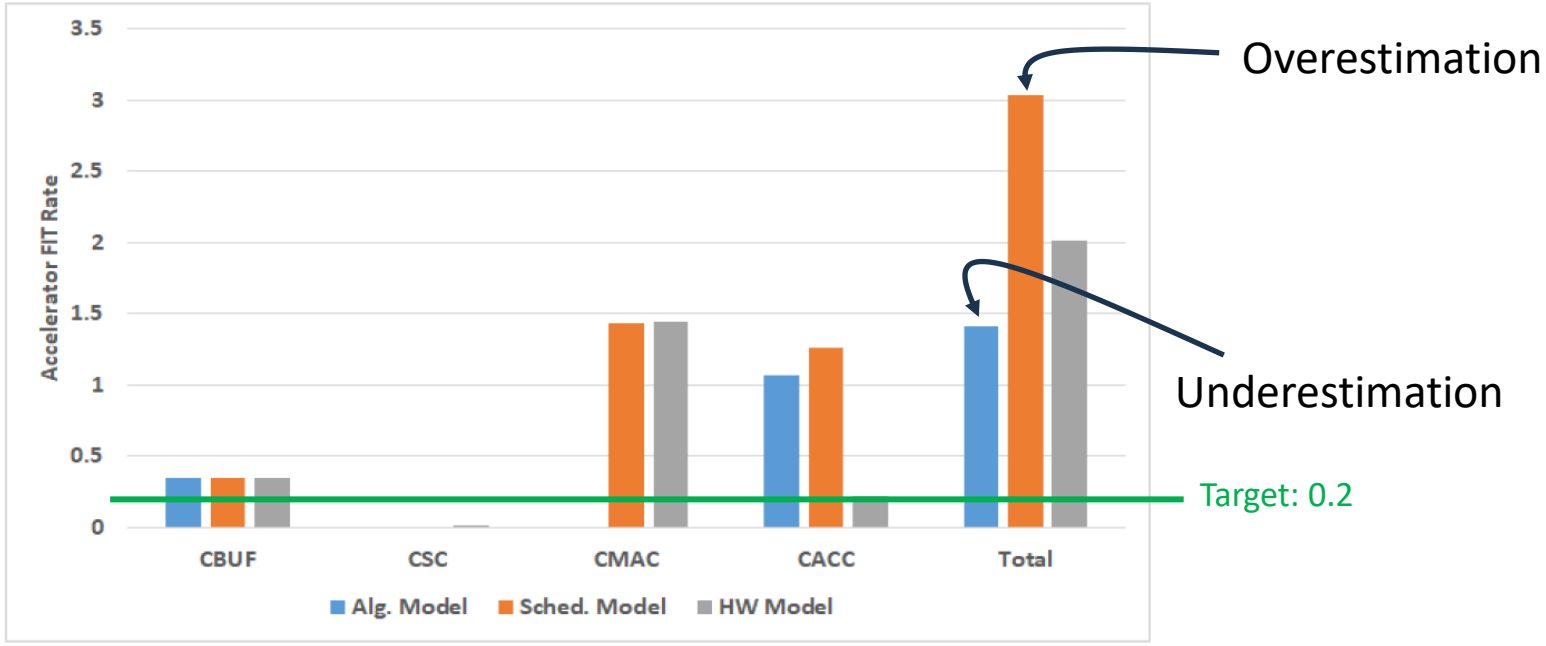
The answer depends on the model used for fault injection!!



- **Alg. Model** accounts only for **storage errors of inputs and weights;**
- **Sched. Model** introduces **MAC array visibility** (roughly 5% accuracy drop)
- **Both Alg. and Dataflow Models** grossly overestimate **severity of accumulator faults**
- **HW Model** accurately captures **buffer addressing errors** (roughly 40% accuracy drop)

How likely are such faults?

Again, the answer depends on the model used for fault injection!!



- **FIT rate** constraint for all FFs in NVDLA must be **< 0.2 (chipset for autonomous cars)**
- The total FIT rate **largely exceeds the target**
  - Mainly determined by CMAC faults; CACC faults alone hit the target
- The **Alg. Model** ignores CMAC faults, thus underestimating total FIT rate
- **The Sched. Model** overestimate CACC FIT rate, and total FIT rate as well

Good enough for reliability analysis of the datapath

Algorithm 1 Algorithm-oriented model for FC layer.

```
wt_mat ← Weight Matrix
ft_vec ← Feature Vector
S ← ft_vec length
for all k ∈ MAC_cells do
    for i : 0 → S − 1 do
        cacc[k] += ft_vec[i] × wt_mat[k][i]
    end for
    dlv[k] = cacc[k]
end for
```

Algorithm 2 Scheduling-oriented model for FC layer.

```
wt_mat ← Weight Matrix
ft_vec ← Feature Vector
Atomics ← Number of Atomic Operations
Muls ← Number of Multipliers per each MAC cell
for all k ∈ MAC_cells do
    for atm : 0 → Atomics − 1 do
        for i : 0 → Muls − 1 do
            psums[k] += ft_vec[i] × wt_mat[k][i]
        end for
        cacc[k] += psums[k]
    end for
    dlv[k] = saturation(cacc[k] ≫ Shift)
end for
```

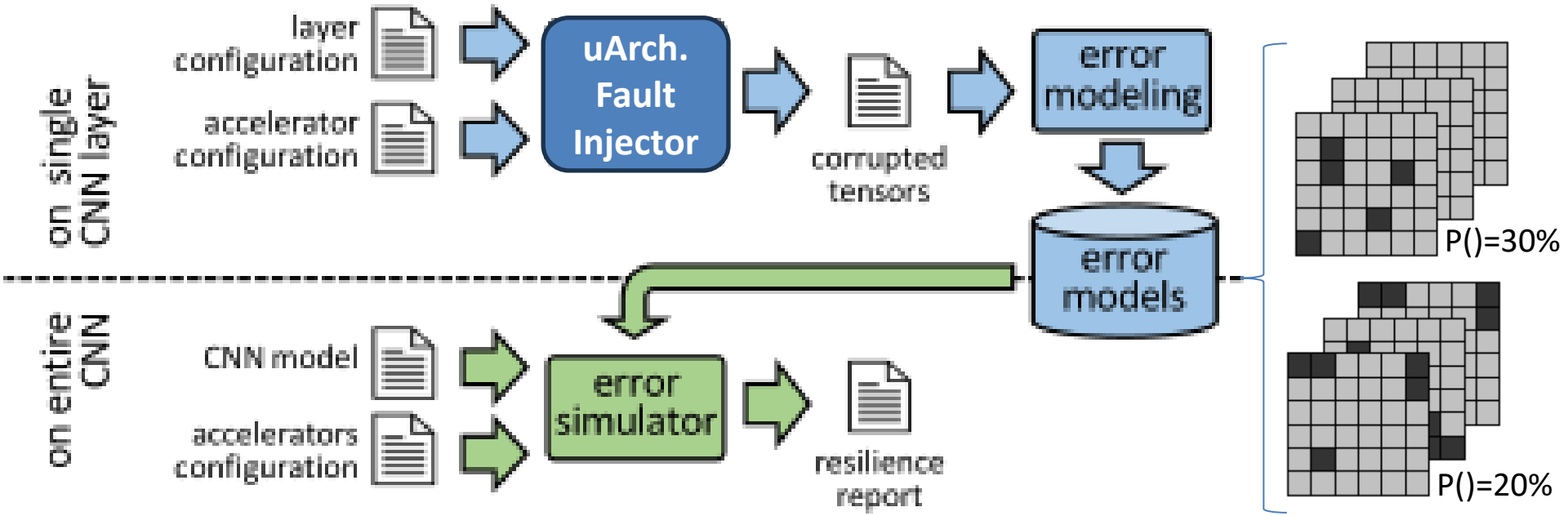
Algorithm 3 Hardware-oriented model for FC layer.

```
wt_mat ← Weight Matrix
ft_vec ← Feature Vector
Atomics ← Number of Atomic Operations
Muls ← Number of Multipliers per each MAC cell
rt_num ← Retiming registers in the adder tree
for all k ∈ MAC_cells do
    for atm : 0 → Atomics − 1 do
        for i : (atm × Muls) → ((atm+1) × Muls) − 1 do
            ft_load[i] = ft_vec[i]
        end for
        for i : (atm × Muls) → ((atm+1) × Muls) − 1 do
            wt_load[i] = wt_mat[k][i]
        end for
        for i : 0 → Muls − 1 do
            mul_out[k] += ft_load[i] × wt_load[i]
            rt_reg[i ÷ rt_num] += mul_out[i]
        end for
        for j : 0 → rt_num − 1 do
            psums[k] += rt_reg[j]
        end for
        cacc[k] += psums[k]
    end for
    dlv[k] = saturation(cacc[k] ≫ Shift)
end for
```

+ Simulation speed within the same order of magnitude of algorithmic models
+ Spatial coverage and fault propagation have been validated against RTL
+ Unmodelled datapath registers can be still accounted for through RTL-based post-processing
- Still, part of the local control and the whole global cannot be modelled

Cross-Layer Reliability Analysis

How to correlate error syndromes of output tensors to DNN model accuracy?



- **Integrated cross-layer analysis framework**
  - **Microarchitectural simulator**
  - **Application-level error simulator**, e.g., CLASSES
- Exchange format**: error model database with associated occurrence probabilities**

Outline

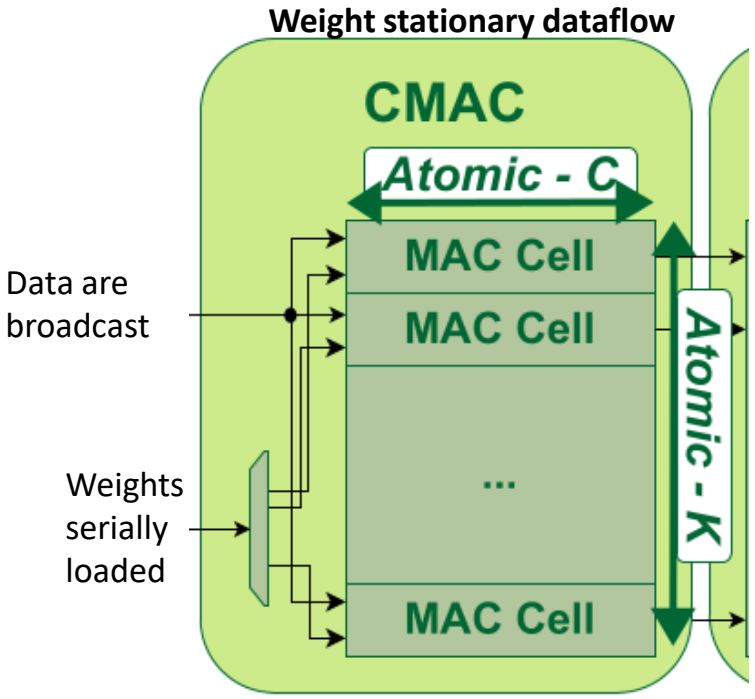
Perform reliability analysis of a state-of-the-art DLA datapath

We've got the tool!

Fault Propagation through Data Reuse

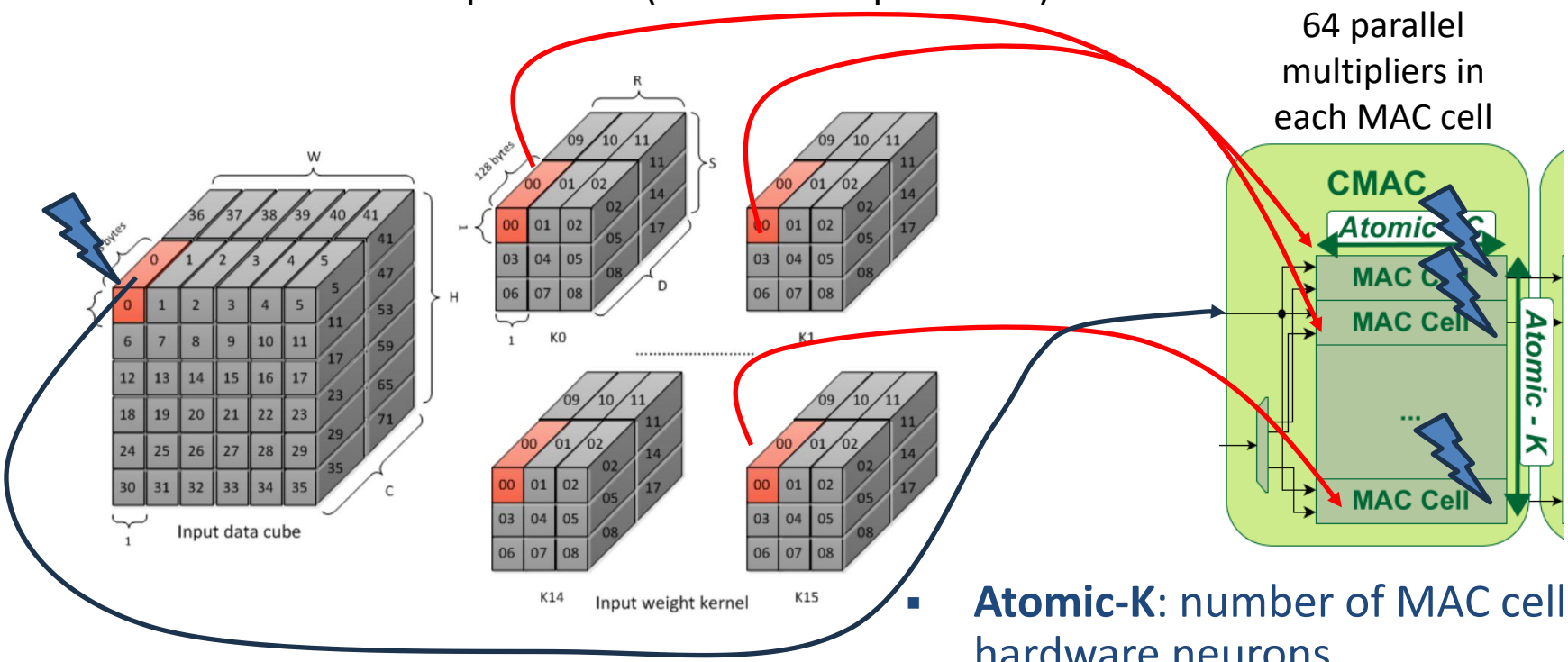
- **Fault propagation in a deep learning accelerator depends on the Data Reuse Policy**
- **Such propagation is mediated by hardware configuration parameters**

Correlation between **hardware configuration space** and **fault propagation** has been **poorly explored**



Key Parameters for Resilience to SEUs

NVDLA - The convolution is split into a number of basic vector-matrix multiplications (i.e. atomic operations)

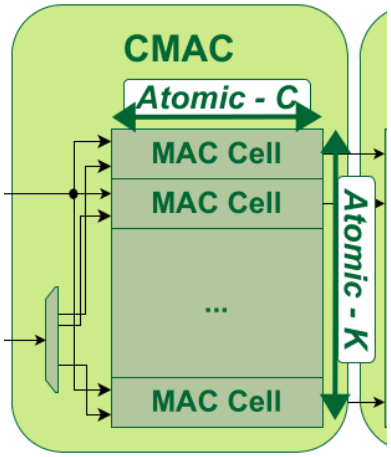


- A **corrupted input** may be **shared among several output neurons**
- The **number of affected output neurons** is directly **determined by the size of the MAC array** itself

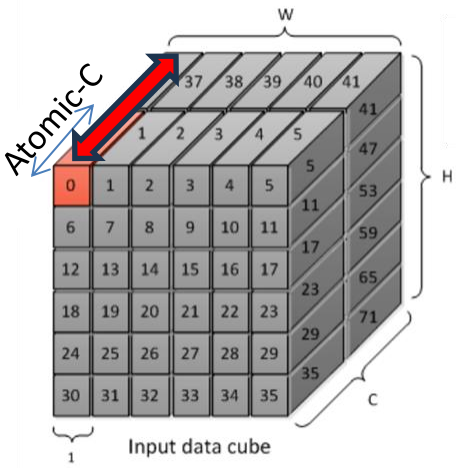
- **Atomic-K**: number of MAC cells, aka hardware neurons
  - Affects fault propagation to output channels

Key Parameters for Resilience to SEUs (contd)

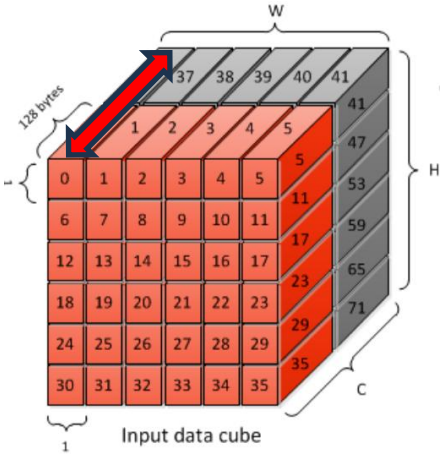
Atomic operations are repeated several times to process the input data cube



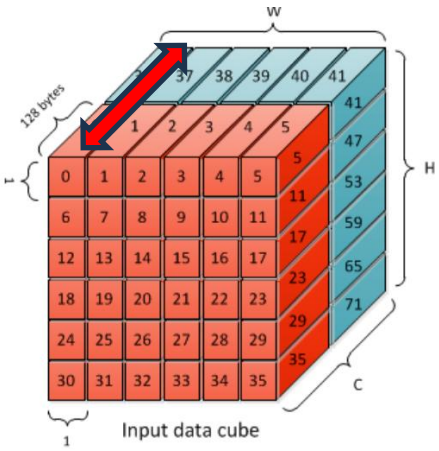
Atomic operation



Block operation



Channel operation



- **Atomic-C**: number of multipliers per MAC cell
  - Affects convolution buffer lifetime (i.e., exposure to faults)

With a 2x deeper MAC Cell, the whole cube could be processed with half the number of atomic operations (i.e., of the time)

Goal

- Let us capture the reliability implications of key accelerator parameters
  - Atomic-C
  - Atomic-K
  - Data precision

Tested Hardware Configurations

	CBUF*	Atomic-C	Atomic-K	ABUF*	SDP-Thpt	Bitwidth
nv_small64	16384	8	8	1024	1	Int8, int16, int32
nv_small256	8192	32	8	1024	1	Int8, int16, int32
nv_medium512	8192	32	16	2048	4	Int8, int16, int32
nv_medium1024	8192	32	32	2048	8	Int8, int16, int32
nv_large2048	4096	64	32	2048	16	Int8, int16, int32

*: Number of buffer entries

To ensure a fair comparison, we considered an **operative frequency** of **250MHz** for a **40nm technology**.

Tested DNN Models

	AlexNet	ResNet-18	YOLO-v3
Dataset:	CIFAR-10	CIFAR-100	COCO
Metric:	Top-1	Top-1	mAP*
int32:	79.1%	79.3%	53.9%
int16:	79.1%	79.3%	53.9%
int8:	78.0%	69.0%	30.9%**

*: As error metric, the Jaccard index has been considered instead.

**: Due to the poor performance, YOLO-v3 quantized at int8 won't be further considered.

Inference time

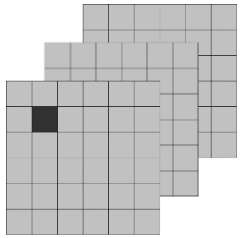
	AlexNet	ResNet-18	YOLO-v3
nv_small64	22.7 ms	190.1 ms	1082.3 ms
nv_small256	16.2 ms	142.7 ms	1013.8 ms
nv_medium512	7.46 ms	69.6 ms	546.0 ms
nv_medium1024	7.1 ms	35.3 ms	273.3 ms
nv_large2048	2.5 ms	5.0 ms	37.3 ms

The higher the amount of resources the lower the execution time

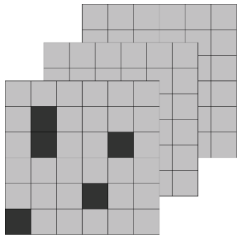
Is there a trade-off with reliability?

Observed Fault Models

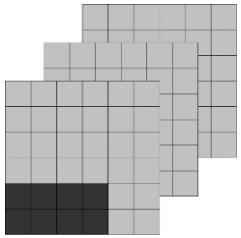
We grouped the observed **error geometries in the output tensors of single DNN layers** into 6 different **error patterns**.



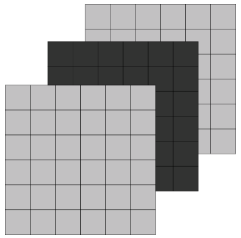
Single Point



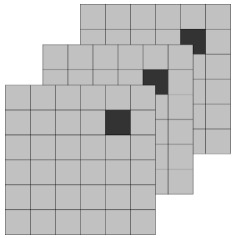
Single Channel Random



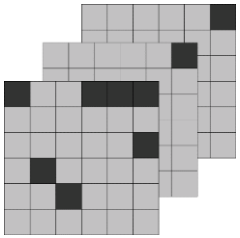
Single Block



Full Channel



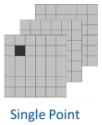
Bullet Wake

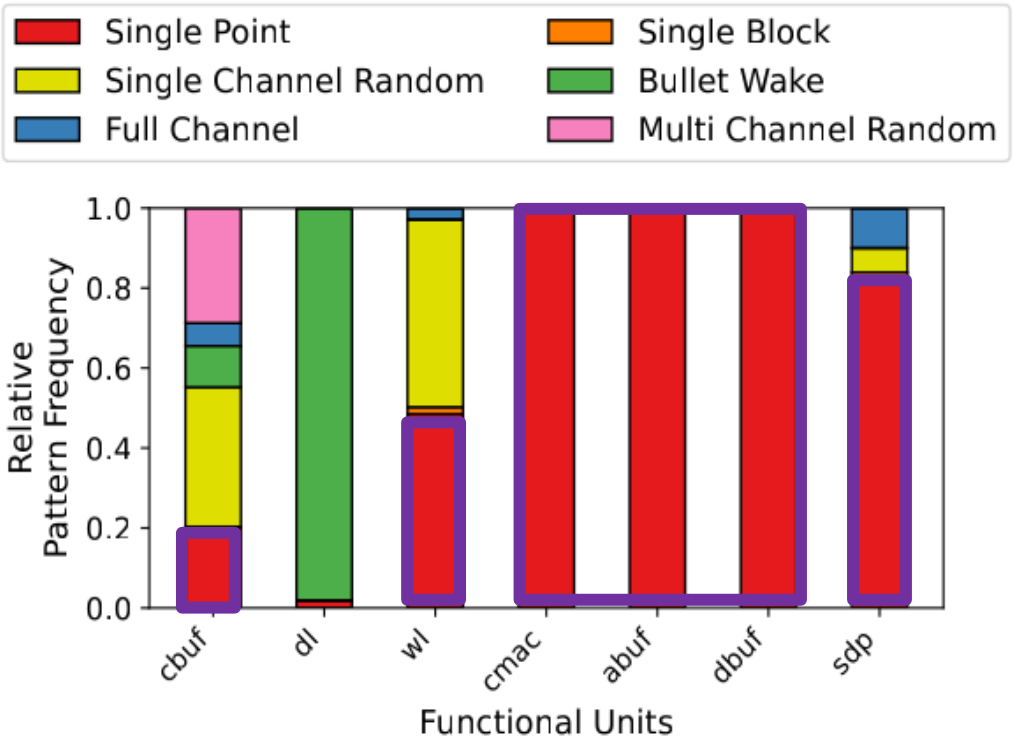


Multi Channel Random

What is their relative frequency?

SEU-Induced Errors Breakdown



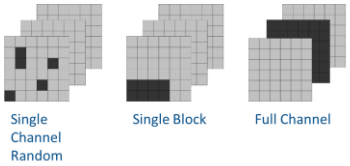


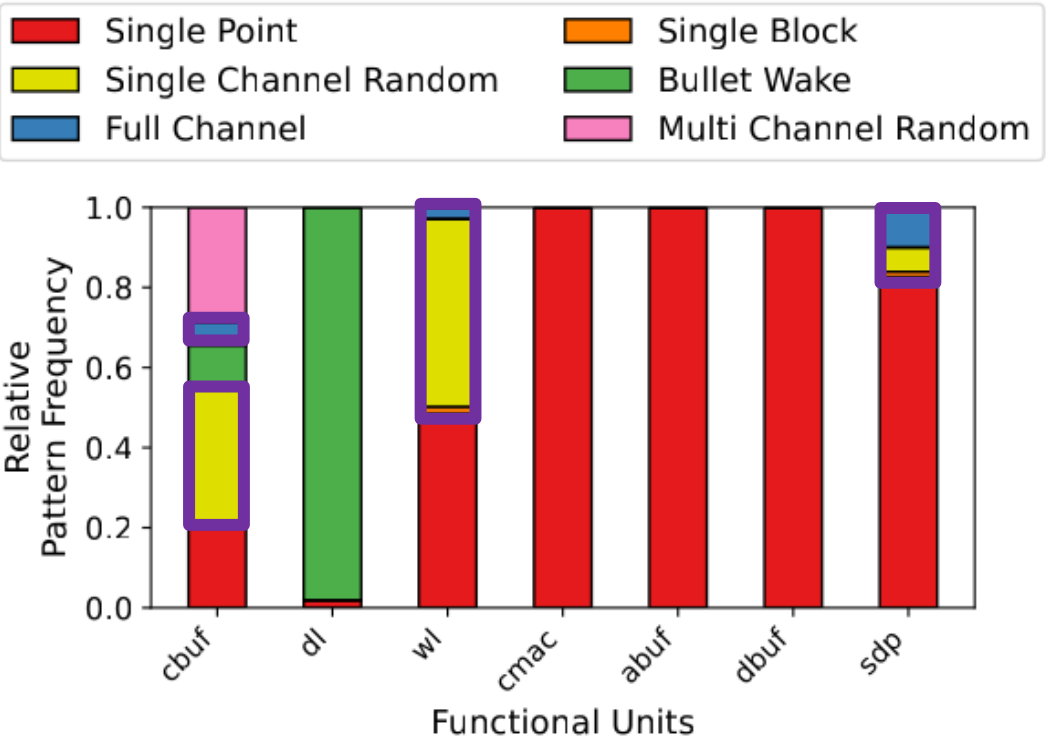
Reference architecture: *nv_medium512 with int16*

Avg occurrence probab. across all layers of all DNN models

- **Single-Point errors**
  - **Most common**
  - **Faults neither masked nor amplified**
  - **100% in CMAC and CACC**
- **Originated in the SDP by corrupted activation tensors**

SEU-Induced Errors Breakdown

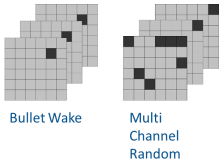


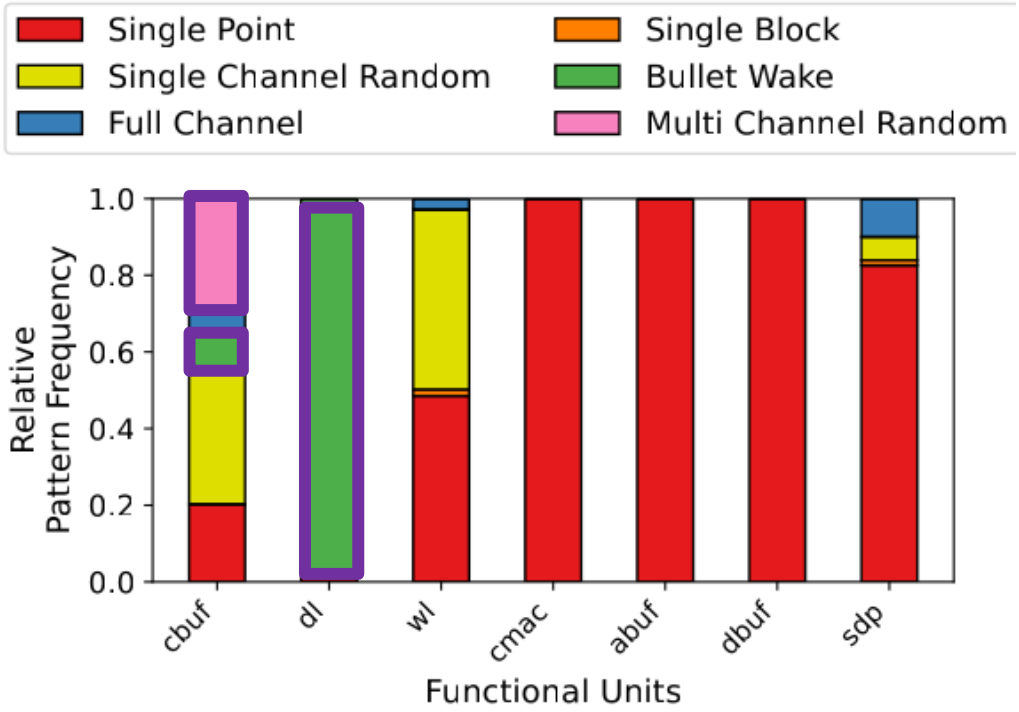


Reference architecture: *nv_medium512 with int16*

- **Full Channel and Single Block**
  - Associated with faults corrupting weigths
  - Weights are loaded into a single MAC cell
  - May occur either in CBUF or in WL
  - Sparse activations may convert them into *Single-Channel Random*
- **Full Channel and Single-Channel Random as an effect of bias corruption in SDP**

SEU-Induced Errors Breakdown

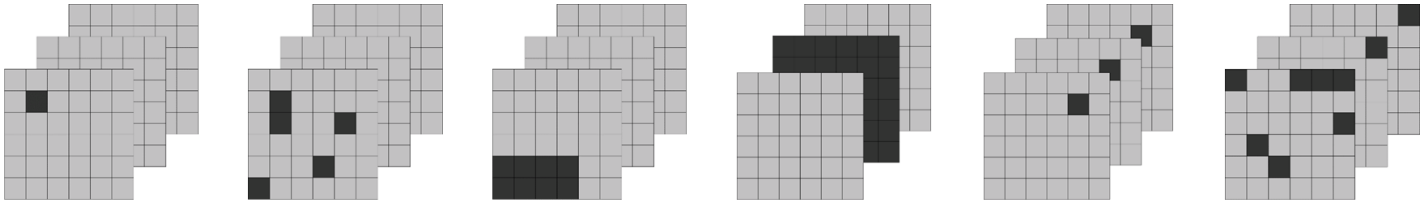




Reference architecture: *nv_medium512 with int16*

- **Bullet Wake and Multi-Channel Random**
  - **Generated by corrupted feature data**
  - **The fault location does play a role**
    - **CBUF: mainly Multi-Channel Random**
    - **DL: mainly Bullet Wake**

Key Takeaways



Single Point | Single Channel Random | Single Block | Full Channel | Bullet Wake | Multi Channel Random

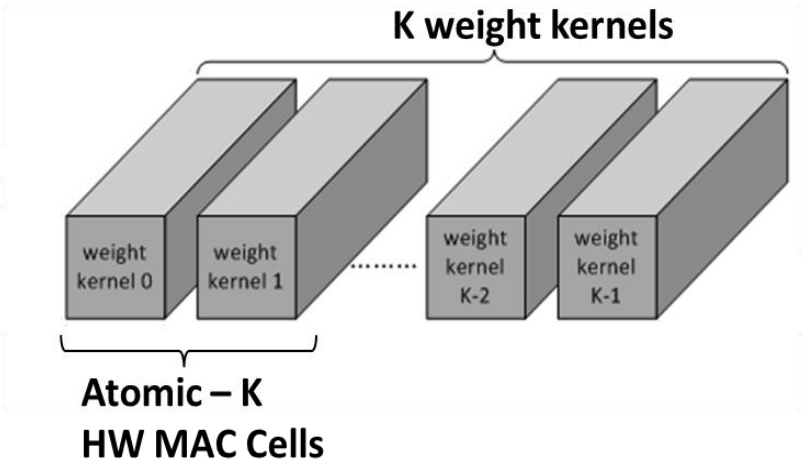
- **Faults in weights** turn into **single-channel corruption**
- **Faults in input feature data** turn into **cross-channel** output tensor **corruption**
- **The Single-Point error pattern is pretty common**
- **Bullet Wake is consistently the least common error pattern**

How "dangerous" are such error patterns for the running DNN model?

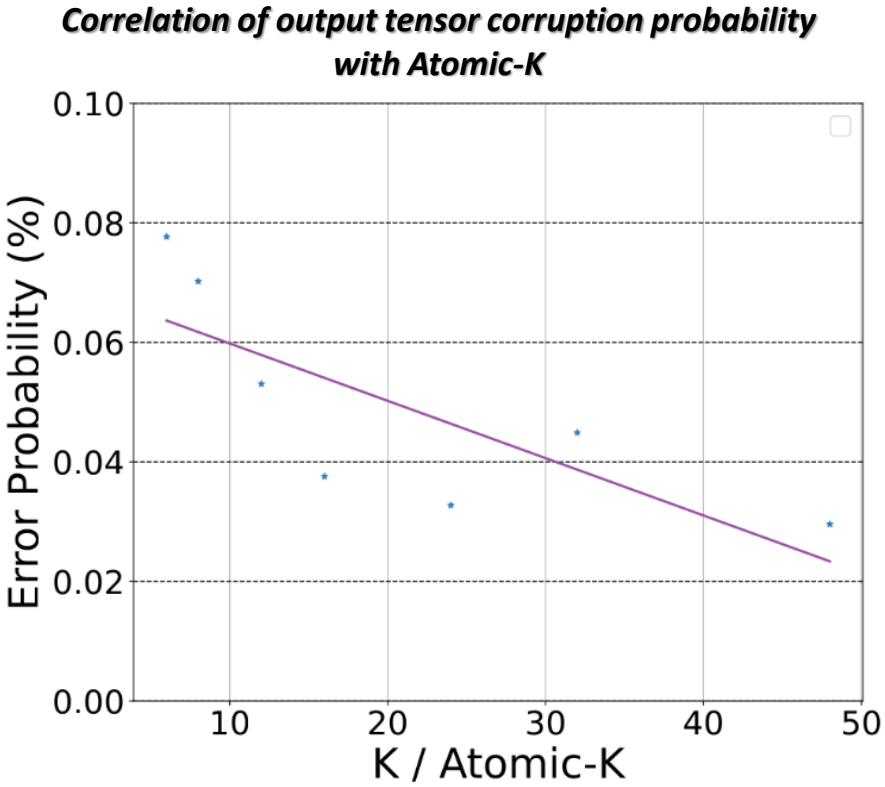
The error patterns are all ways to corrupt output tensors of DNN layers….

….how is the probability of such corruption related to MAC Array instantiation parameters?

Configuration Space Analysis: Atomic - K

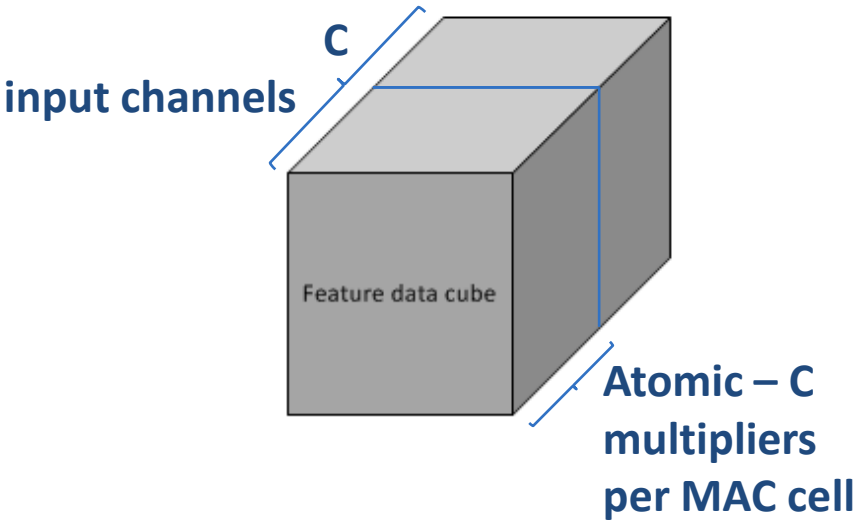


- The **more** the **hardware convolutions**, the **smaller** they are..
- …**the lower the prob. that a fault propagates!**



K / Atomic-K indicates the **number of hardware convolutions** the layer is decomposed into

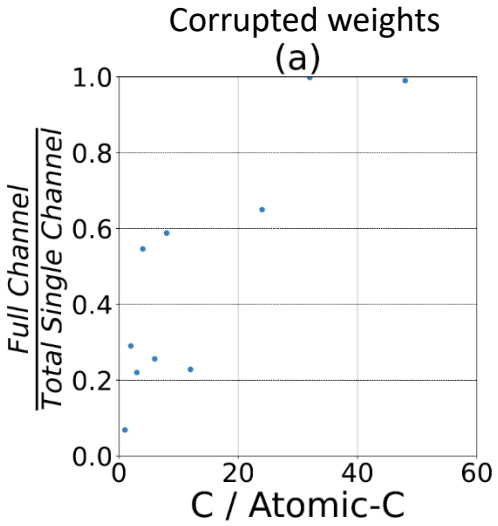
Configuration Space Analysis: Atomic - C



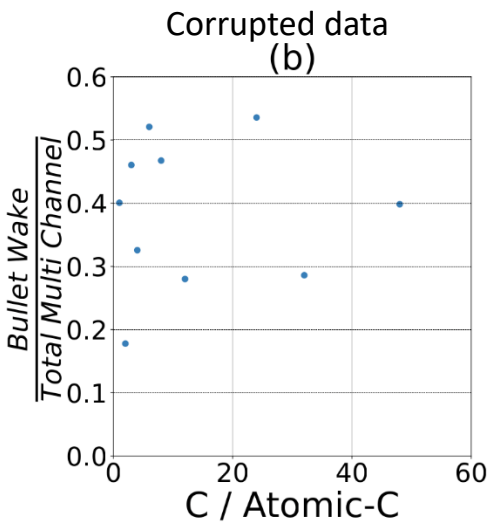
- A **small Atomic-C** causes **way more atomic operations** to process the input data cube..
- ..augmenting the input **buffer exposure**
- The relative impact of **the most serious error patterns (Bullet Wake and Full Channel) increases**

Correlation of high-impact error patterns probability with Atomic-C

With a Full Channel pattern, YOLOv3 has a 73% probability to fail



With a Bullet Wake pattern, YOLOv3 has a 90% probability to fail

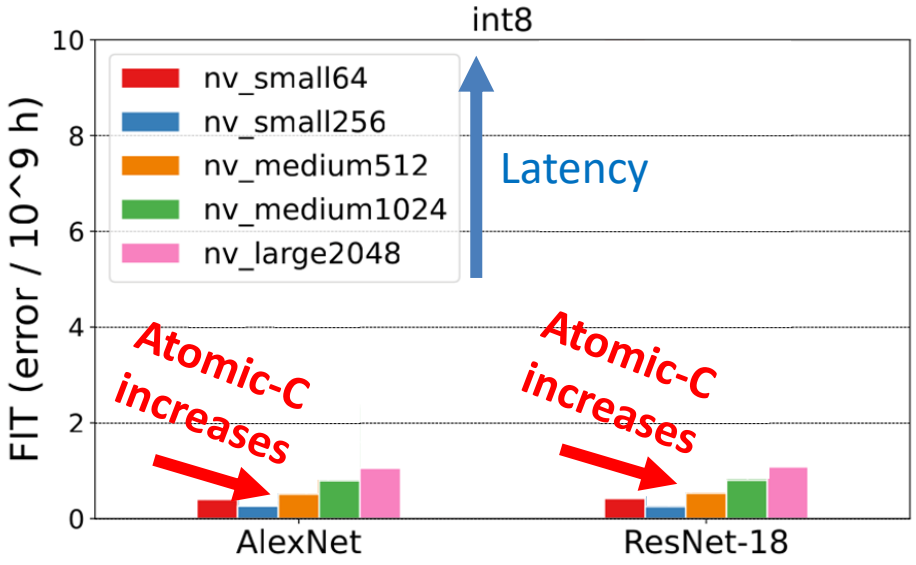


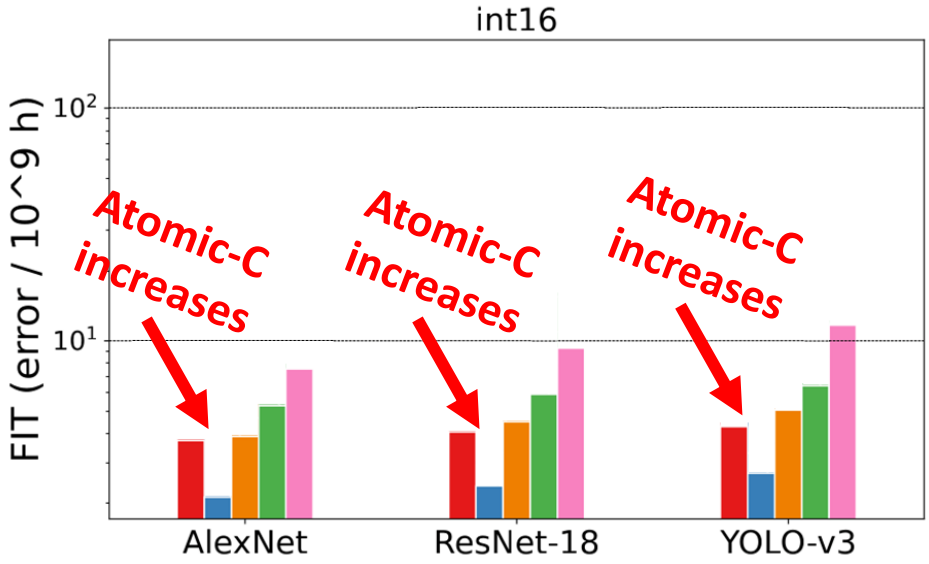
C / Atomic-C determines the relative occupancy of the input buffer

Now we are in a position to appreciate the Performance-Reliability Trade-Off spanned by the Accelerator Configurations under test

Performance-Reliability Trade-Off

Number of misclassifications per 1 billion hours operation (1 FIT)

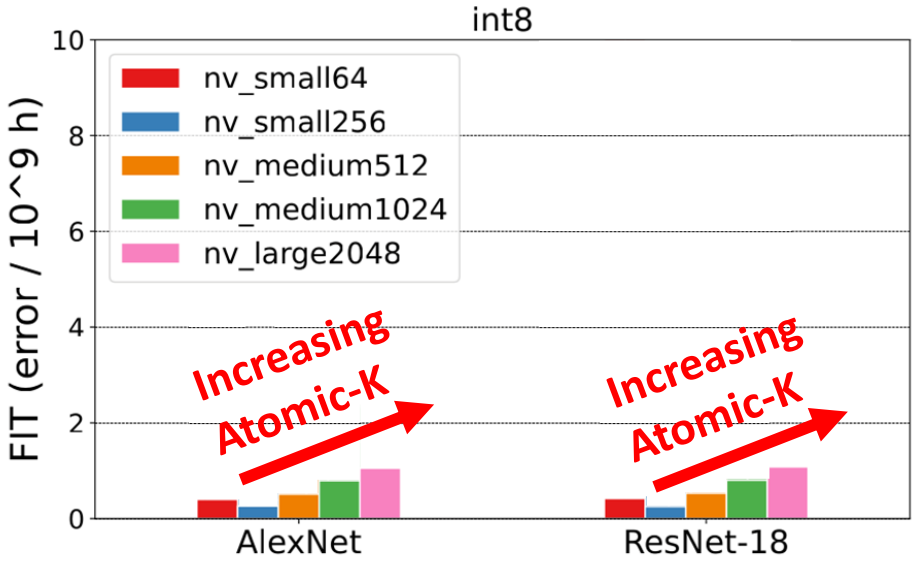


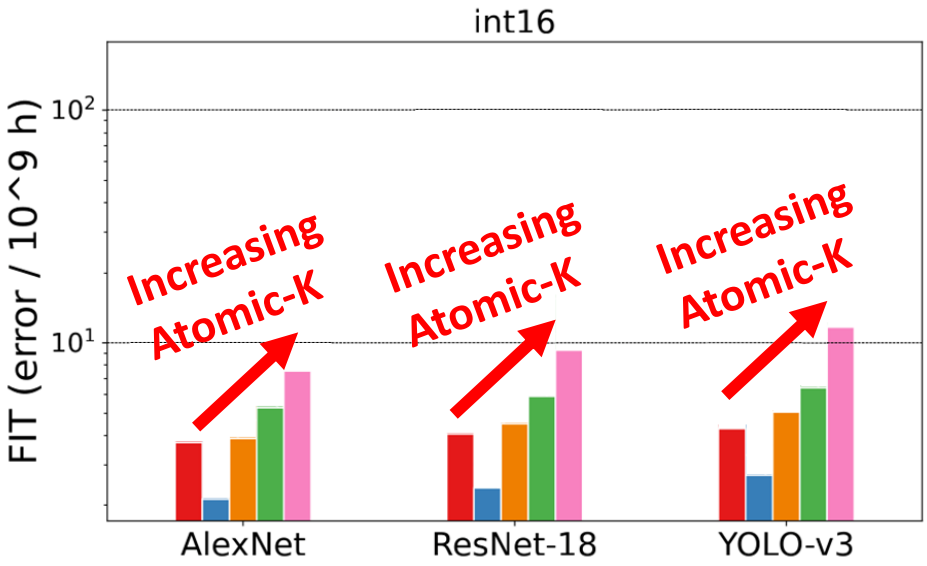


- An increasing Atomic-C causes lower exposure of the input buffer and less dangerous error patterns..
- …hence a lower FIT rate!

Performance-Reliability Trade-Off

Number of misclassifications per 1 billion hours operation (FIT)





- An increasing Atomic-k causes wider error propagation…
- …hence a higher FIT

Key Takeaway

Higher performance and higher implementation cost do not necessarily lead to lower reliability….

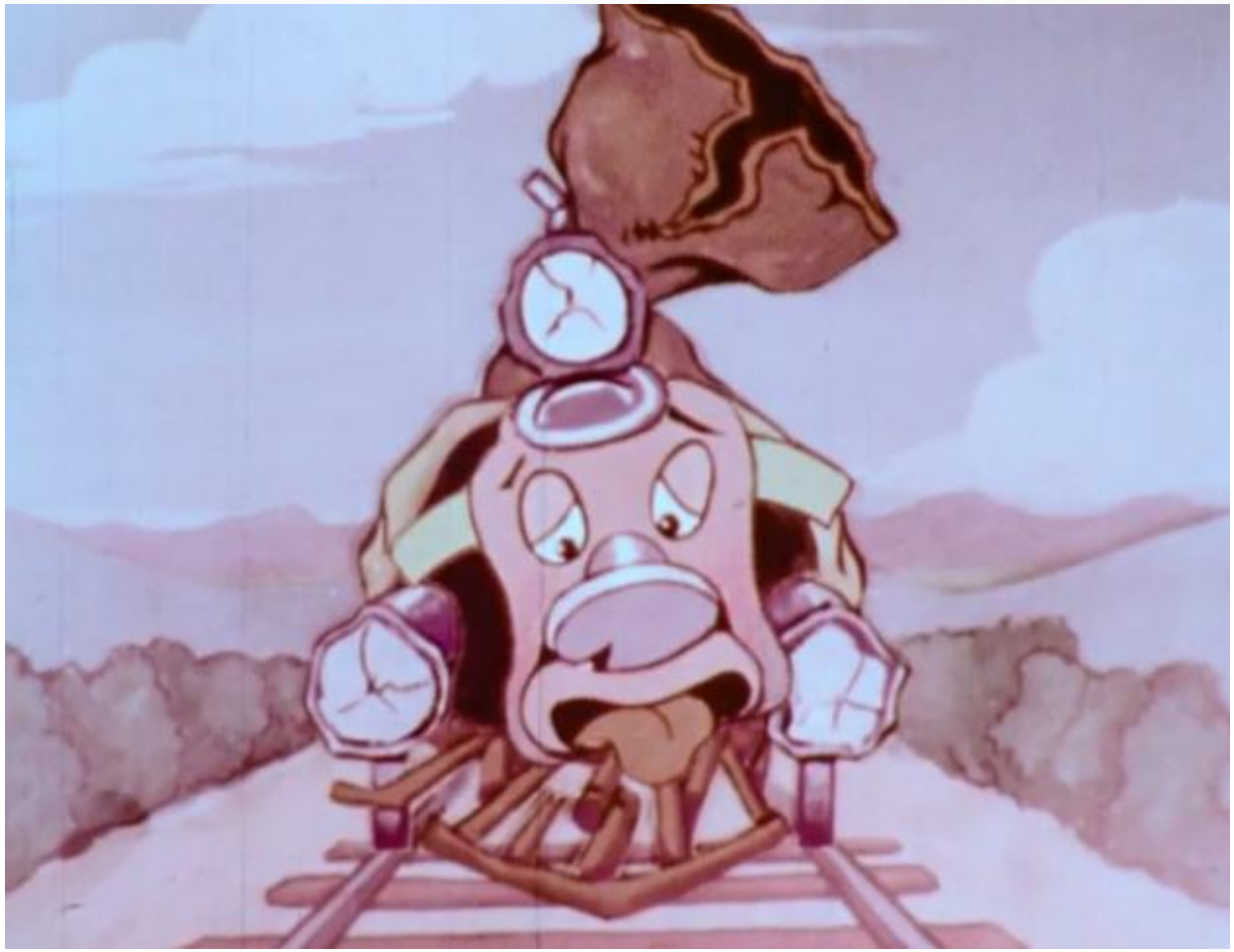
…pretty much depends on which resources are instantiated

Outline

We've got the tool!

DLA datapath has been analyzed (reliability-wise)

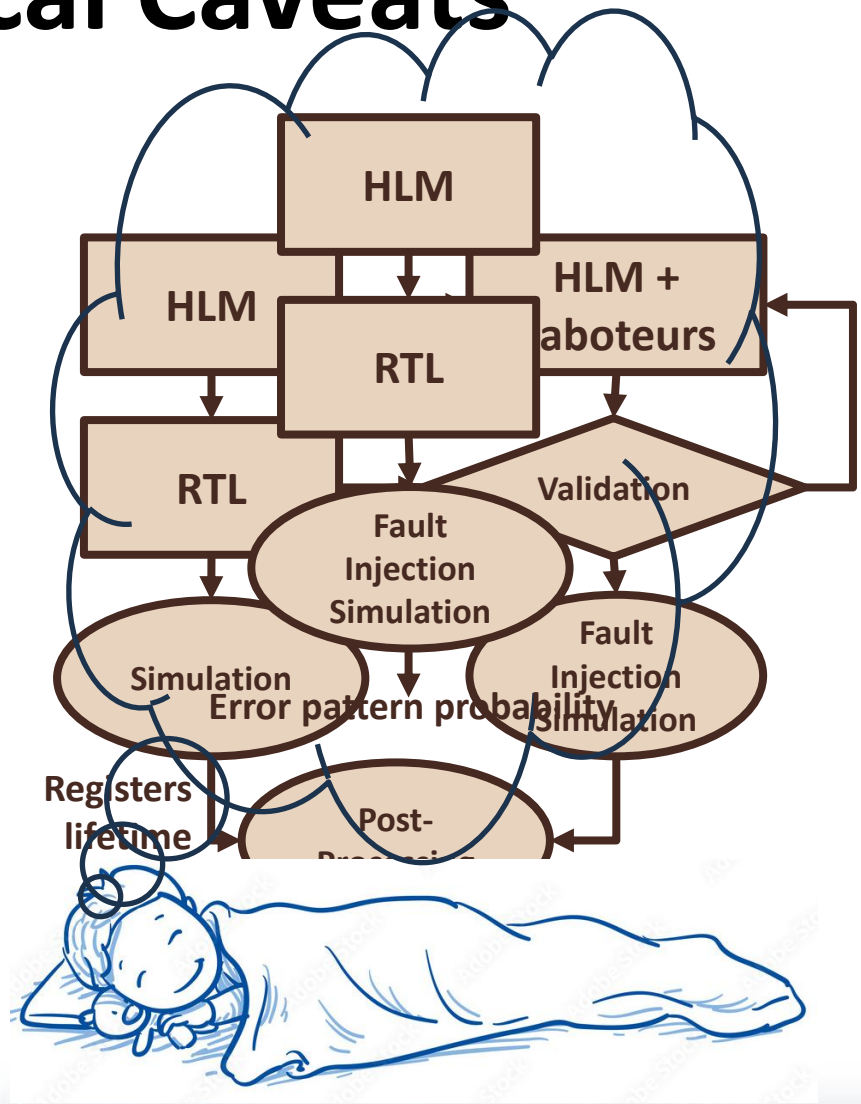
Are we using the right methodology? What about the control path?



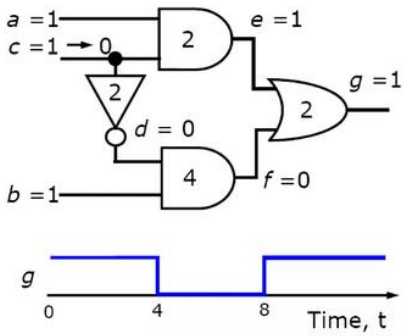
We’ve done the right things, but took so much effort, and we are not done yet!

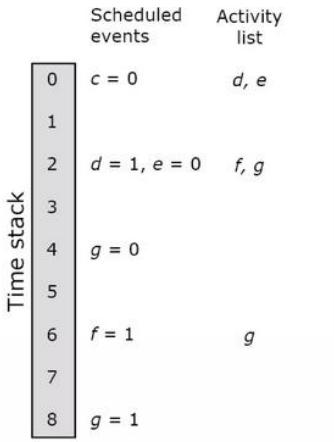
Methodological Caveats

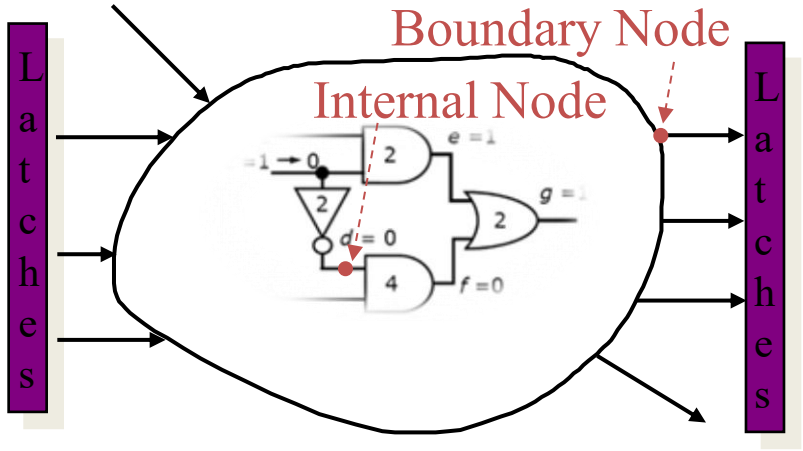
- **Lack of full hardware implementation awareness**
  - **Requires** extensive **saboteurs validation**
  - **Requires post-processing** with HW implementation information
  - **Good for „what-if“ scenarios**, but not for full design analysis
- **Unable to model the control path**



Overcoming the Event-Driven Sim. Barrier







- **Both timing and functional verification**
  - **All nodes visible**
  - **Glitches are detected**
- **Time wheel for complete ordering**

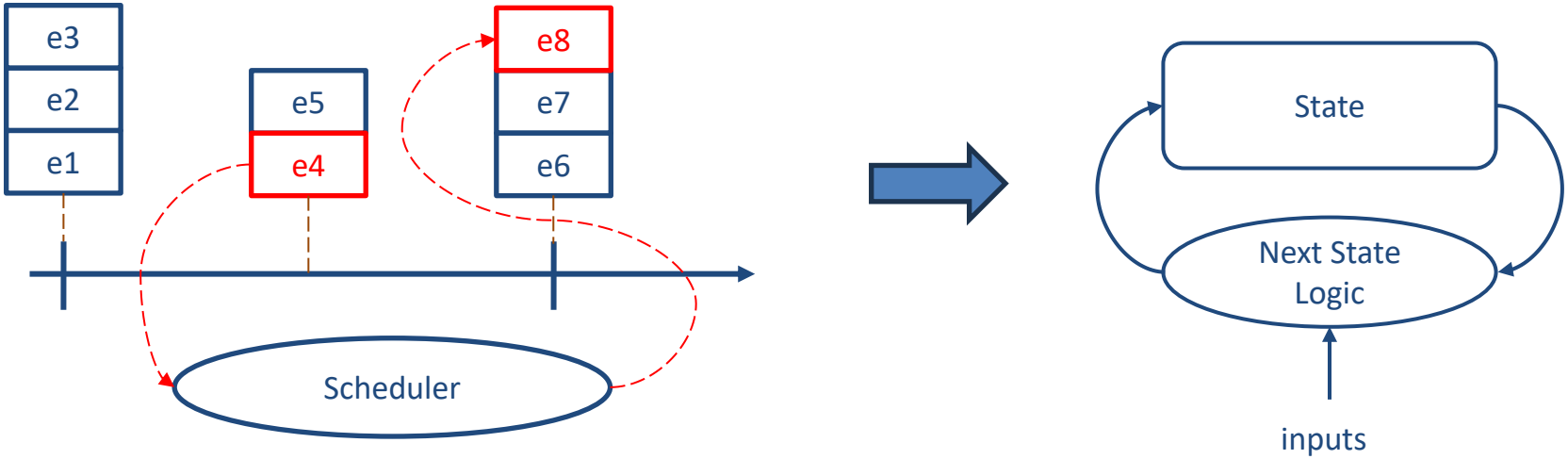
Take advantage of the fact that most digital designs are largely synchronous

- **State elements change value on active edge of clock**
- **Only boundary nodes are evaluated**

- ❑ Only boundary nodes visible
- ❑ Relies on zero gate delay assumption (OK for RTL only)
- ❑ 10x-100x faster than event-driven simulation (and uses less memory)

Event-Driven vs. Cycle-Based: Why So Fast?

The **"cycle-based" paradigm** means that simulation progresses in discrete clock steps, rather than simulating all events and interactions at the time-accurate level, as traditional event-driven simulators do.



The **performance boost** cycle-based simulation exposes is **caused by** the **lower number of EV&UPs** in the simulation **and by** the lack of an **event scheduler**

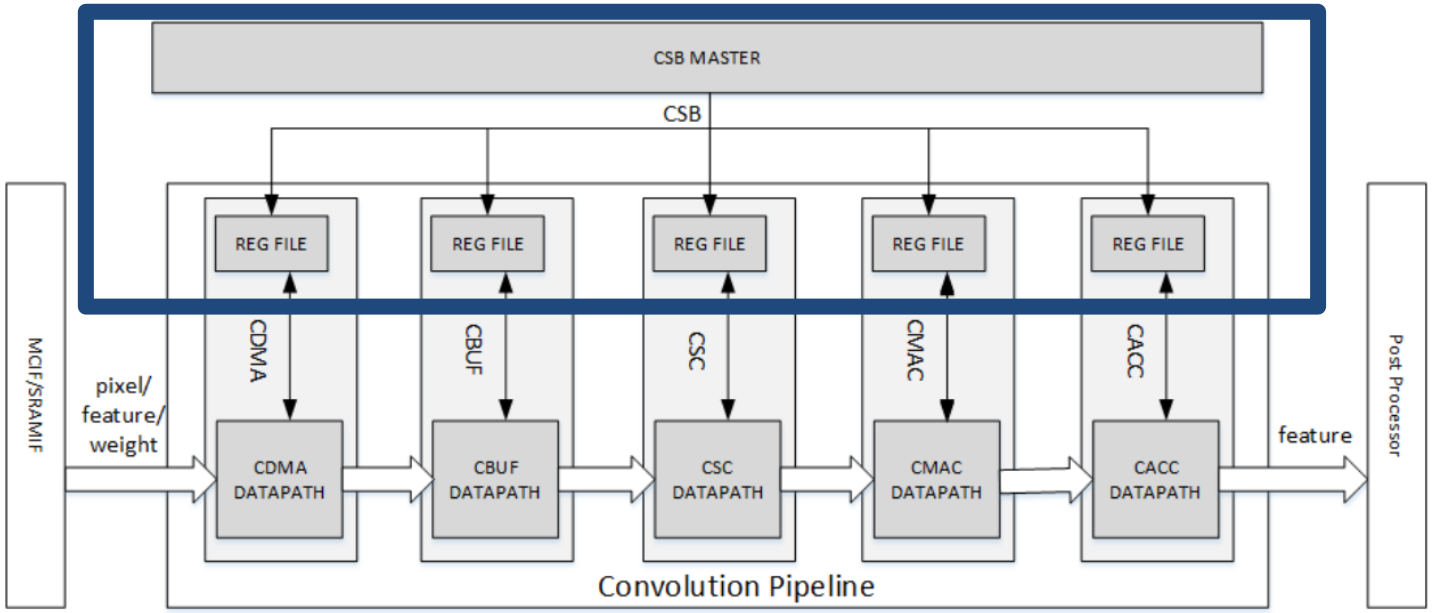
Fast Fault Simulation with EMBER

The Extensible Microarchitectural Benchmark for Error Resilience (EMBER) is a C++ RTL modelling library with cycle-based simulation paradigm, and built-in fault-injection support.

- **Cycle-Based simulation paradigm**
  - **Faster than event-driven RTL** simulation
- **Exploits C++ for flexibile hardware description**
  - Exploits **C++ meta-language** features **for datapath** description
  - Design‘s **configurability parameters** are **evaluated at** class‘ **initialization time**
- **Supports Fault-Injection through built-in Saboteurs**
  - **No need for design customization** for saboteurs insertion
  - **Decouples RTL design** from **fault model design**
  - **Designed to be extensible** for new fault models

Current Work

Currently extending the reliability analysis to the global controller

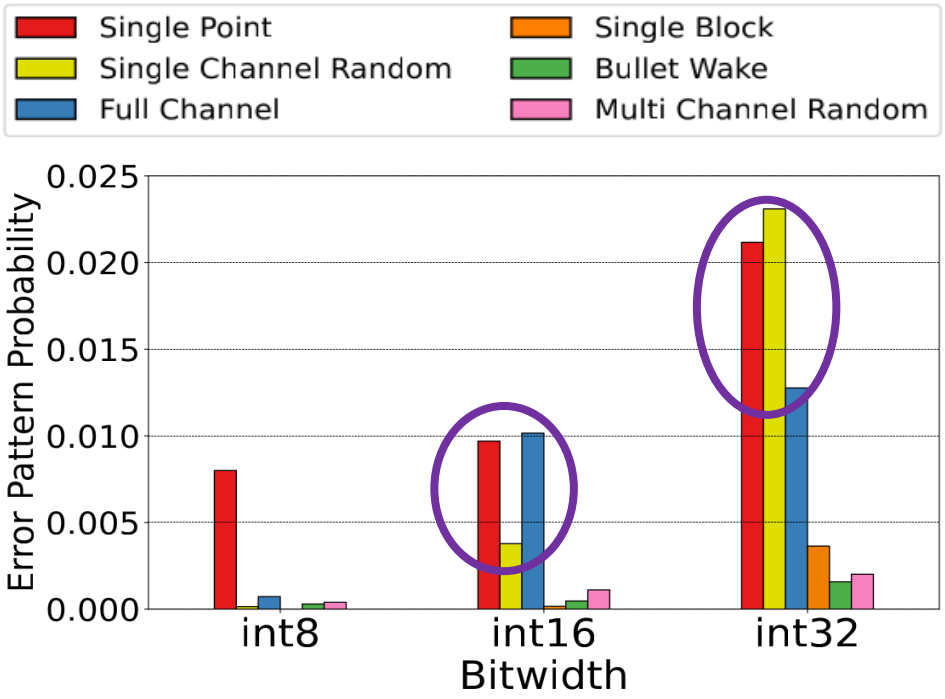


Conclusions

- Understanding fault behavior is key for the selective hardening of deep learning accelerators
- Dataflow-level injection helps early “what-if” analysis but needs validated saboteurs and heavy post-processing
- Microarchitectural parameters have to be explored, since they affect the performance vs robustness vs cost trade-off
- Cycle-based simulation through imperative software languages allows fast RTL-accurate exploration.
- EMBER combines fast cycle-based simulation, flexible design, and built-in support for fault injection.
- Future work: reliability analysis of the control path of the DLA

Sensitivity to the Bitwidth

Is the relative frequency of error patterns dependent on datapath bitwidth?



- Steep increase of single-channel error patterns
- Effect of the unified CBUF architecture
- Wider pipelines expose more sign bits, especially for weights
- Bullet Wake is the least frequent error pattern
- **Why using INT32 at all?**

Model Vulnerability to Error Patterns

Once they occur, what is the application failure rate?

Susceptivity of the considered DNN model to observed error patterns:

- Despite being the most common, **Single Point** patterns are the **least dangerous** for DNN models under test
- **Bullet Wake** may **significantly corrupt** the network's performance
- **YOLO-v3** is the most vulnerable DNN model to all error patterns

Pattern	Bitwidth	AlexNet	ResNet-18	YOLO-v3
Single Point	Int8	35.9%	28.9%	---
	int16	27.7%	27.4%	40.7%
Single Channel	Int8	51.5%	62.3%	---
	int16	50.7%	63.7%	63.8%
Single Block	Int8	51.9%	59.5%	---
	int16	53.8%	62.9%	58.9%
Full Channel	Int8	57.8%	66.1%	---
	int16	59.3%	69.0%	73.4%
Bullet Wake	Int8	69.0%	84.6%	---
	int16	69.6%	85.1%	90.6%
Multi Channel	Int8	57.4%	64.7%	---
	int16	58.0%	66.0%	62.8%

Model Vulnerability to Error Patterns

Once they occur, what is the application failure rate?

Susceptivity of the considered DNN model to observed error patterns:

- Despite being the most common, **Single Point** patterns are the **least dangerous** for DNN models under test
- **Bullet Wake** may **significantly corrupt** the network's performance
- **YOLO-v3** is the most vulnerable DNN model to all error patterns

Pattern	Bitwidth	AlexNet	ResNet-18	YOLO-v3
Single Point	Int8	35.9%	28.9%	---
	int16	27.7%	27.4%	40.7%
Single Channel	Int8	51.5%	62.3%	---
	int16	50.7%	63.7%	63.8%
Single Block	Int8	51.9%	59.5%	---
	int16	53.8%	62.9%	58.9%
Full Channel	Int8	57.8%	66.1%	---
	int16	59.3%	69.0%	73.4%
Bullet Wake	Int8	69.0%	84.6%	---
	int16	69.6%	85.1%	90.6%
Multi Channel	Int8	57.4%	64.7%	---
	int16	58.0%	66.0%	62.8%

Model Vulnerability to Error Patterns

Once they occur, what is the application failure rate?

Susceptivity of the considered DNN model to observed error patterns:

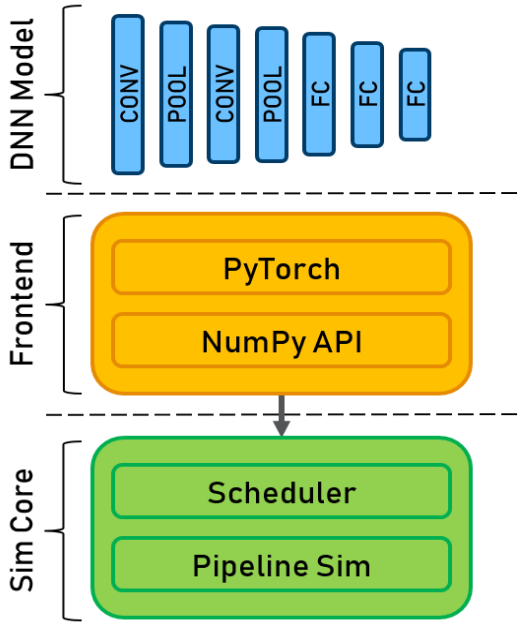
- Despite being the most common, **Single Point** patterns are the **least dangerous** for DNN models under test
- **Bullet Wake** may **significantly corrupt** the network's performance
- **YOLO-v3** is the most vulnerable DNN model to all error patterns

Pattern	Bitwidth	AlexNet	ResNet-18	YOLO-v3
Single Point	Int8	35.9%	28.9%	---
	int16	27.7%	27.4%	40.7%
Single Channel	Int8	51.5%	62.3%	---
	int16	50.7%	63.7%	63.8%
Single Block	Int8	51.9%	59.5%	---
	int16	53.8%	62.9%	58.9%
Full Channel	Int8	57.8%	66.1%	---
	int16	59.3%	69.0%	73.4%
Bullet Wake	Int8	69.0%	84.6%	---
	int16	69.6%	85.1%	90.6%
Multi Channel	Int8	57.4%	64.7%	---
	int16	58.0%	66.0%	62.8%

Most dangerous error patterns

Case Study: NVDLA

EMBER for modelling **NVDLA** and testing its robustness agains faults




```
name: "nv_small"
dtype: "int8"
cbuf:
  num-banks: 16
  bank-depth: 512
  wordsize: 64
cmac:
  batch-size: 1
  atomic-c: 8
  atomic-k: 8
cacc:
  extra-bits: 6
  num-banks: 8
  bank-depth: 512
  wordsize: 136
sdp:
  bs-enable: "True"
  bn-enable: "True"
  ew-enable: "False"
```

- **Python & PyTorch Wrapper**:

Bridging C++ APIs to mainstream frameworks

- **YAML Configuration file**

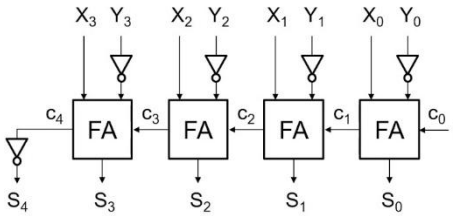
Configuration settings

Datapath as Template Parameter

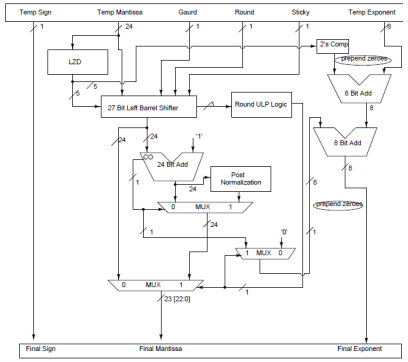
EMBER exploits **C++ meta-language** features for **quick datapath design.**

```
1 void adder::update()
2 {
3     C.write(A.read() + B.read());
4 }
```

adder<flare::int8_t>

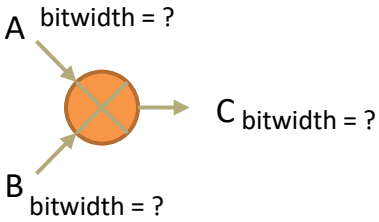


adder<flare::fp32_t>

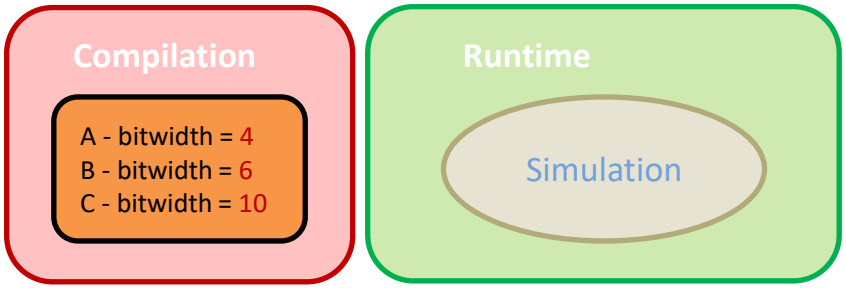


Configuration Initialization Time

Together with **DUT Compilation** and **Simulation**, **EMBER introduces** a **DUT Initialization** step.

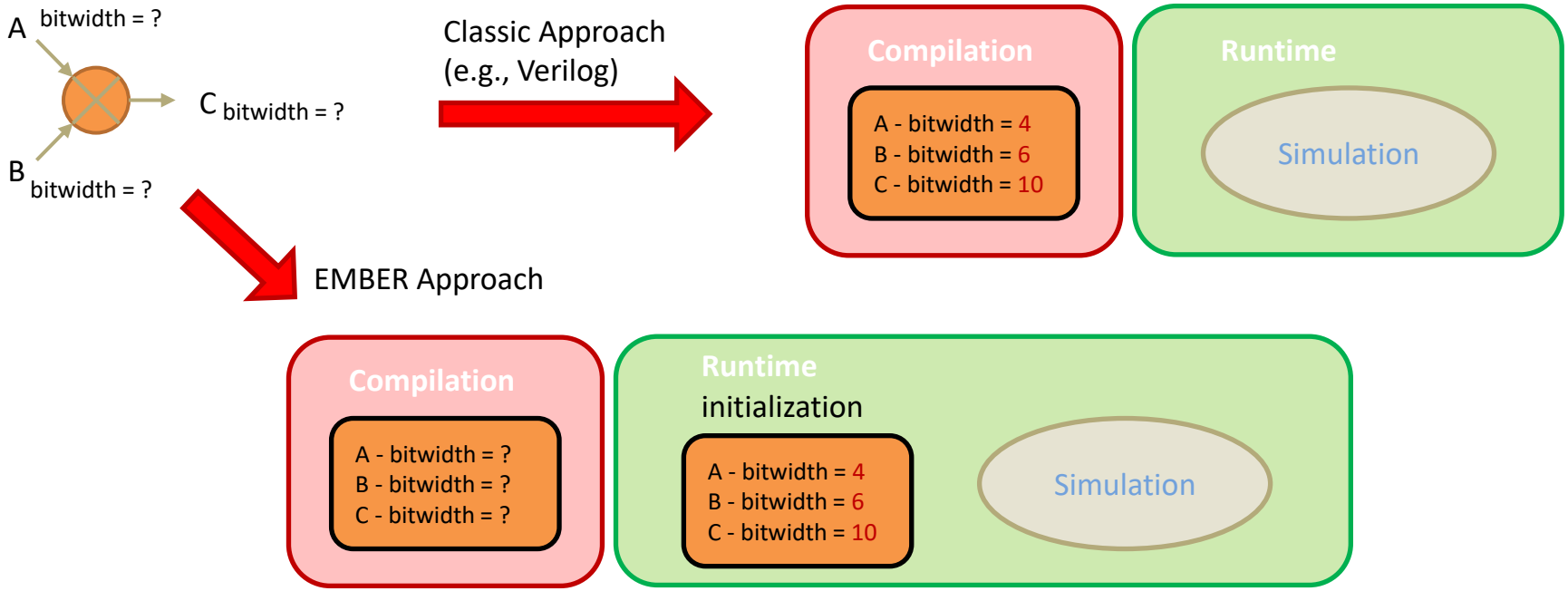


Classic Approach
(e.g., Verilog)



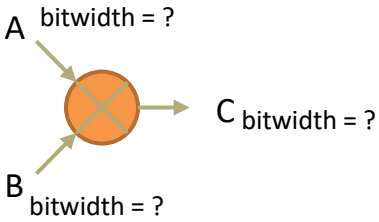
Configuration Initialization Time

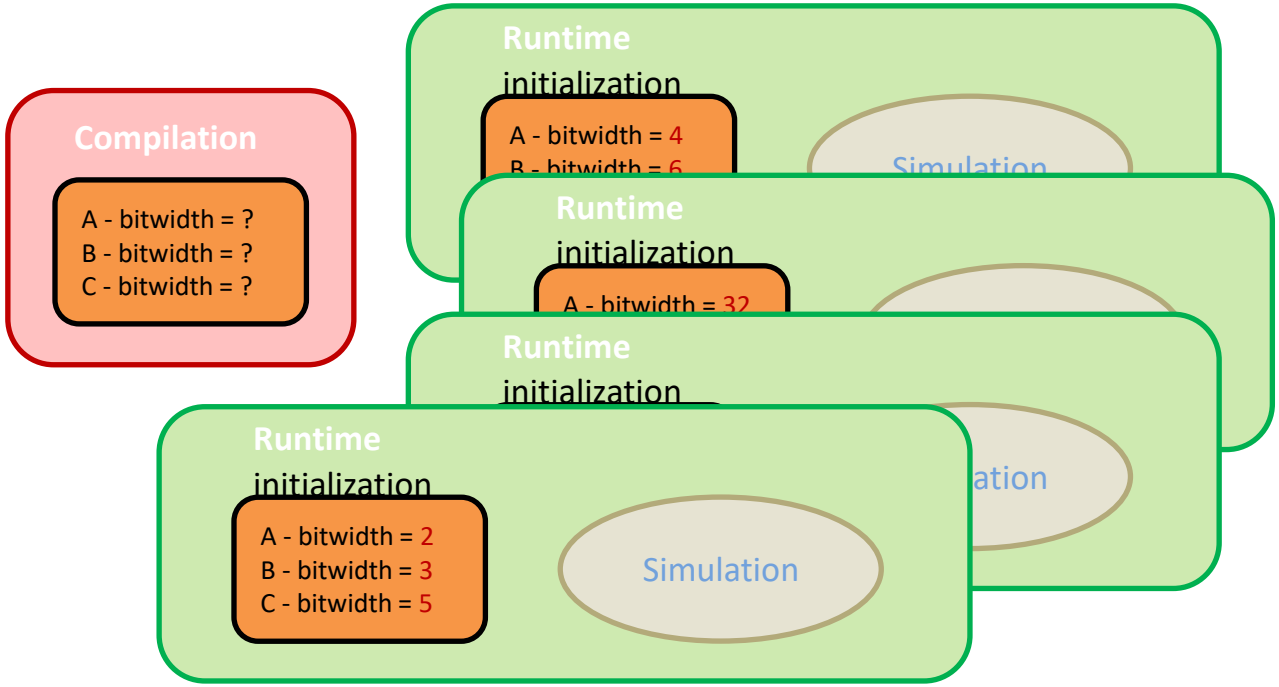
Together with **DUT Compilation** and **Simulation**, **EMBER introduces** a **DUT Initialization** step.



Configuration Initialization Time

Together with **DUT Compilation** and **Simulation**, **EMBER introduces** a **DUT Initialization** step.





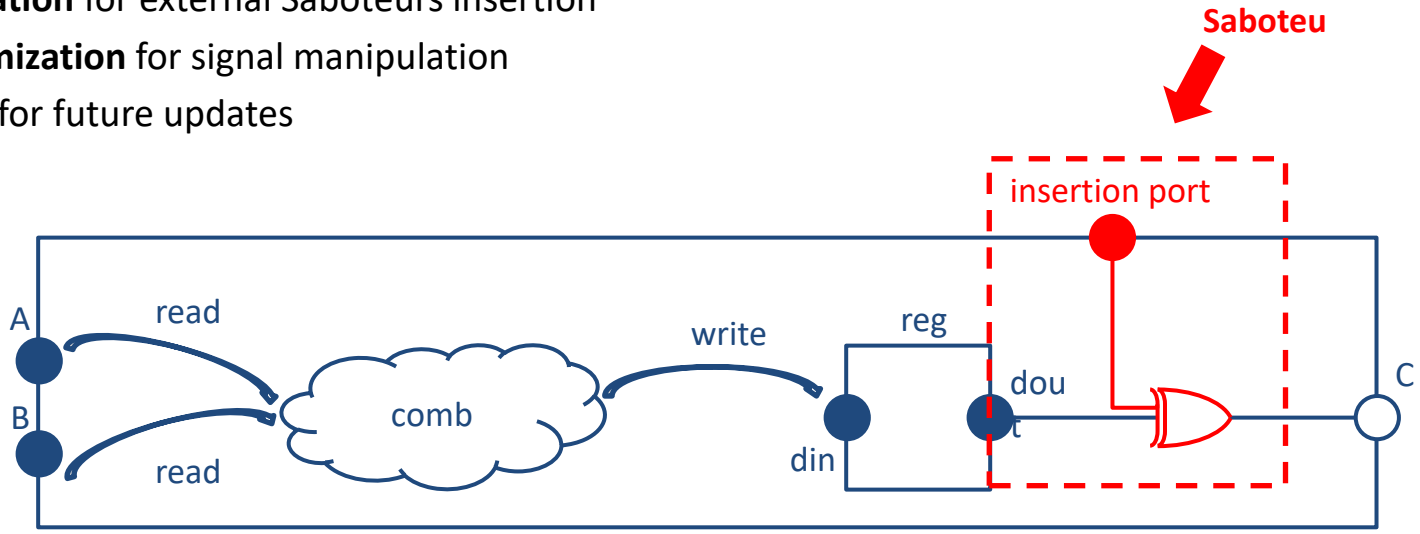
Compile Once, Run Many!

Ideal to explore the configuration space of DLAs

Fault Injection through built-in Saboteurs

EMBER adopts **built-in hardware saboteurs** for supporting **fault injection** capabilities

- **No RTL customization** for external Saboteurs insertion
- **No Scripts customization** for signal manipulation
- **Easily extensible** for future updates



Fault Injection through built-in Saboteurs

EMBER adopts **built-in hardware saboteurs** for supporting **fault injection** capabilities

- **No RTL customization** for external Saboteurs insertion
- **No Scripts customization** for signal manipulation
- **Easily extensible** for future updates