Convolution Processing Unit Featuring Adaptive Precision using Dynamic Reconfiguration

WF-IoT 2021 Paper

David Cain, Omar Eldash, Kasem Khalil, Magdy Bayoumi

UNIVERSITY of LOUISIANA LAFAYETTE

Presented by David Cain

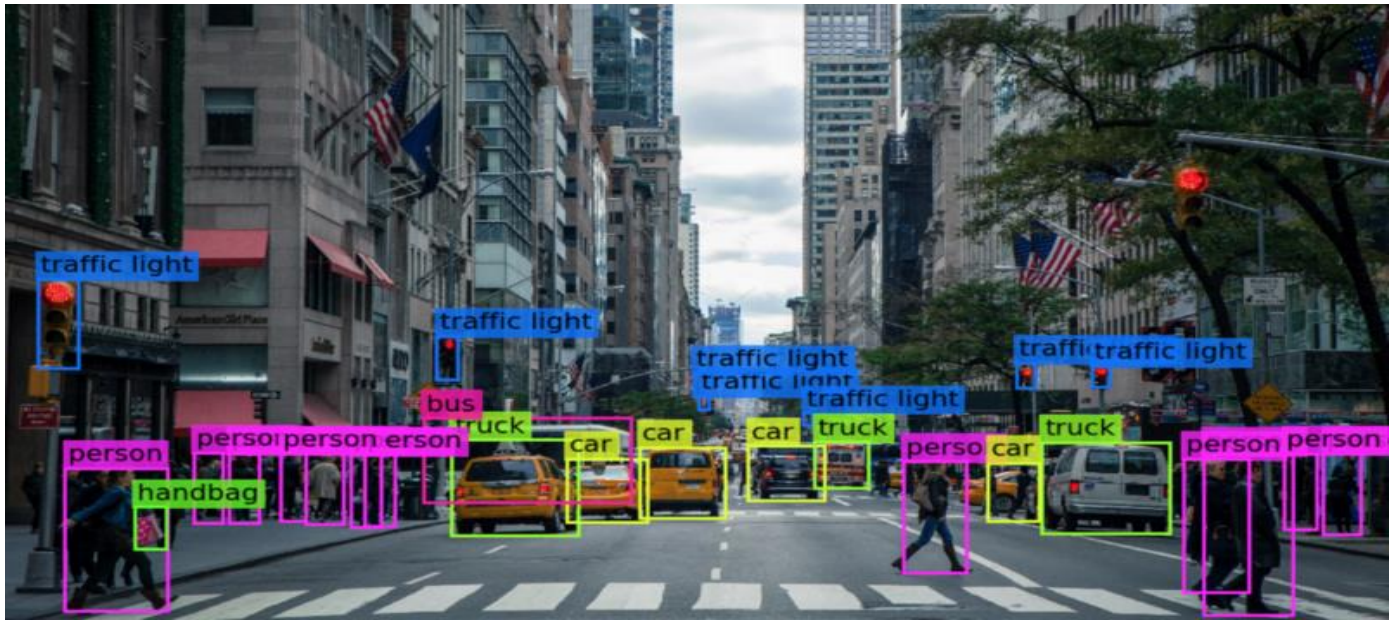
Overview

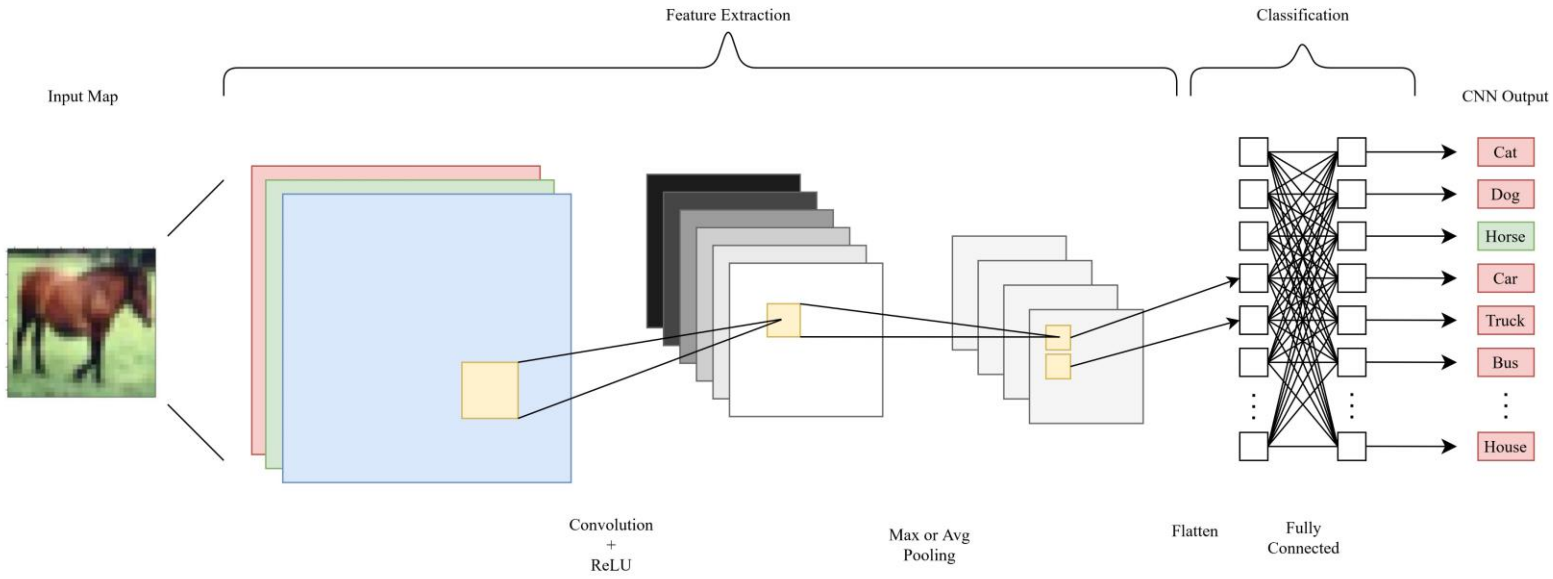
- **Introduction**
  - Computer Vision
  - Traditional IoT CNN Models
  - Hardware Accelerator Types
  - Reconfigurable Hardware
  - Dynamic Partial Reconfiguration
- Related Work
- Proposed Convolution Processing Element
- Hardware Implementation & Comparisons
- Conclusions
- Q&A

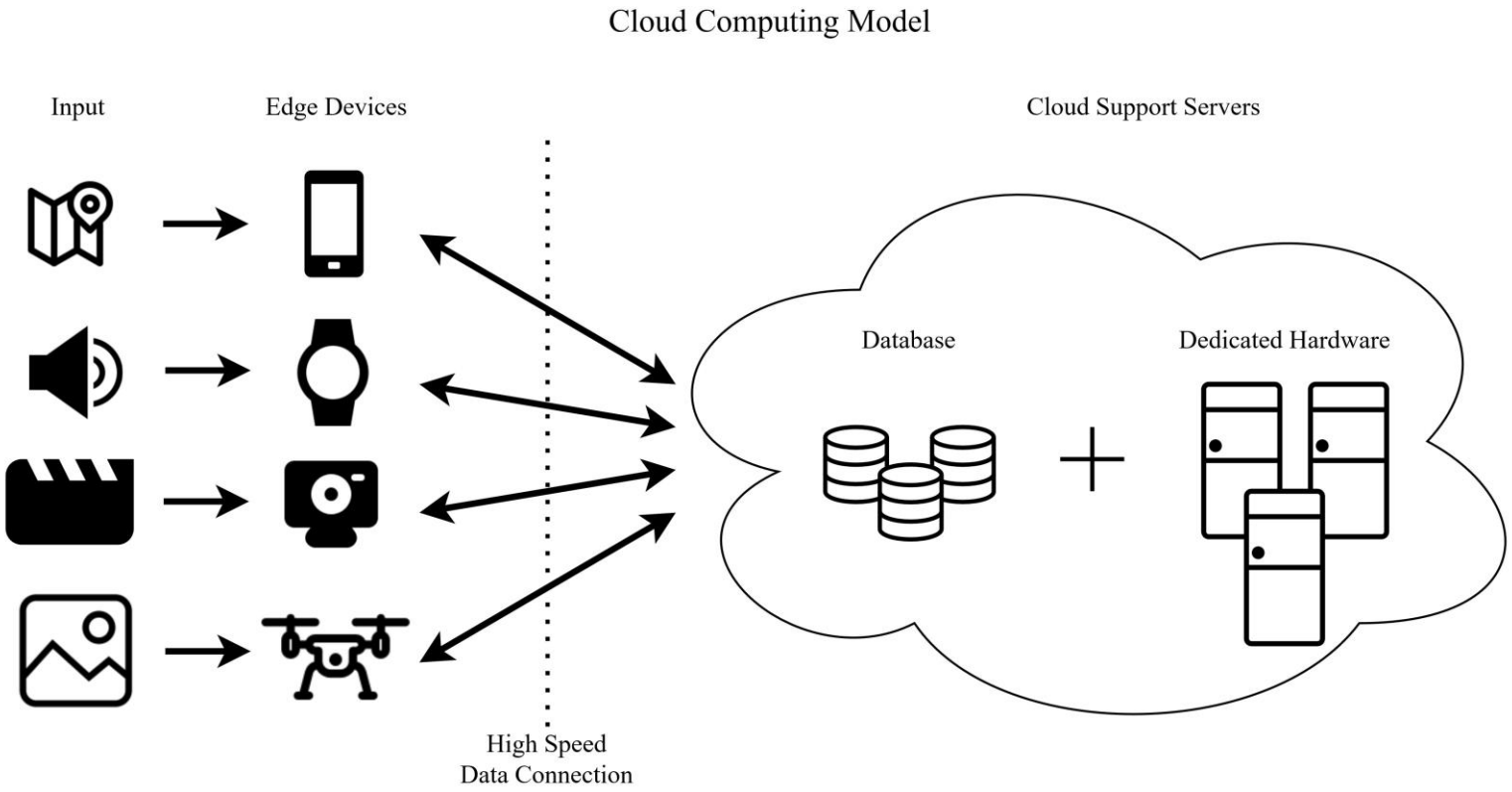
Computer vision is a field which aims to develop models for computers to understand and interpret image data

Unique edge devices utilizing image and video data are now being created

The CNNs used are generally complex and too computationally bulky for IoT edge devices







With traditional cloud computing models, edge devices are "dumb" data aggregators

- Zombie device

Cloud servers perform CNN processes

Dedicated server hardware has much greater processing capabilities and relaxed power constraints compared to edge devices

High connectivity required…

How to get these processes local to the edge device?

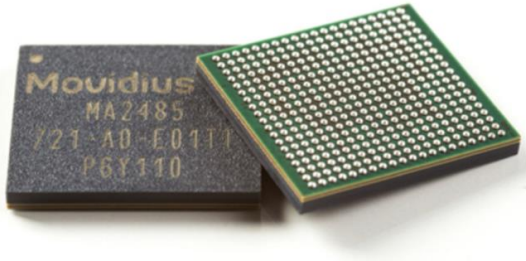
Graphics Processing Unit (GPU)

– ex. nvidia Jetson Nano
– NVIDIA JetPack SDK, TensorFlow, PyTorch

Field Programmable Gate Array (FPGA)

– ex. Xilinx Virtex UltraScale+
– Verilog, VHDL, HDL

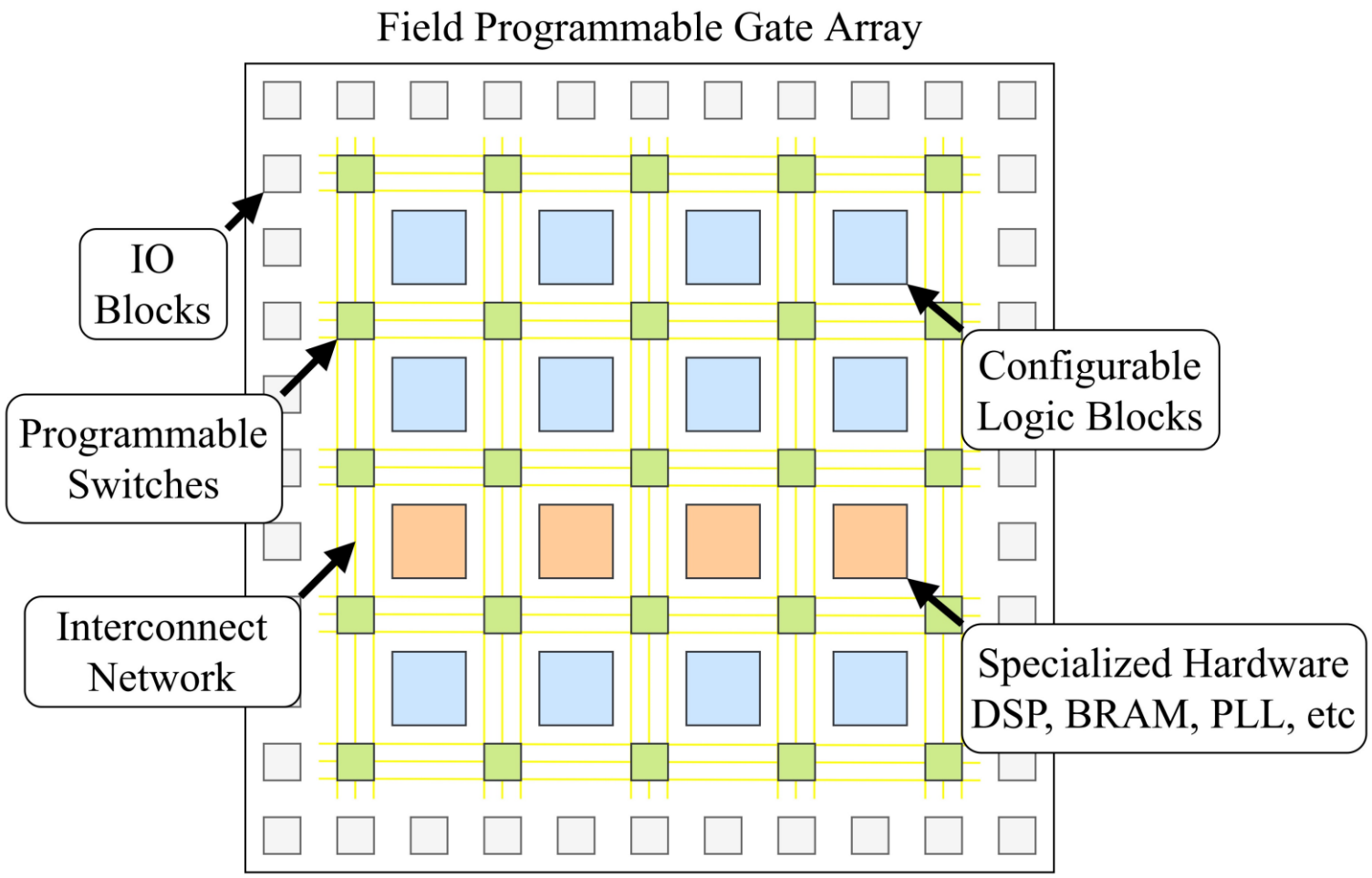
Vision Processing Unit (VPU)



– ex. Intel Movidius Myriad X
– Propriety Intel MDK

Application-Specific Integrated Circuit (ASIC)

– Verilog, VHDL, HDL

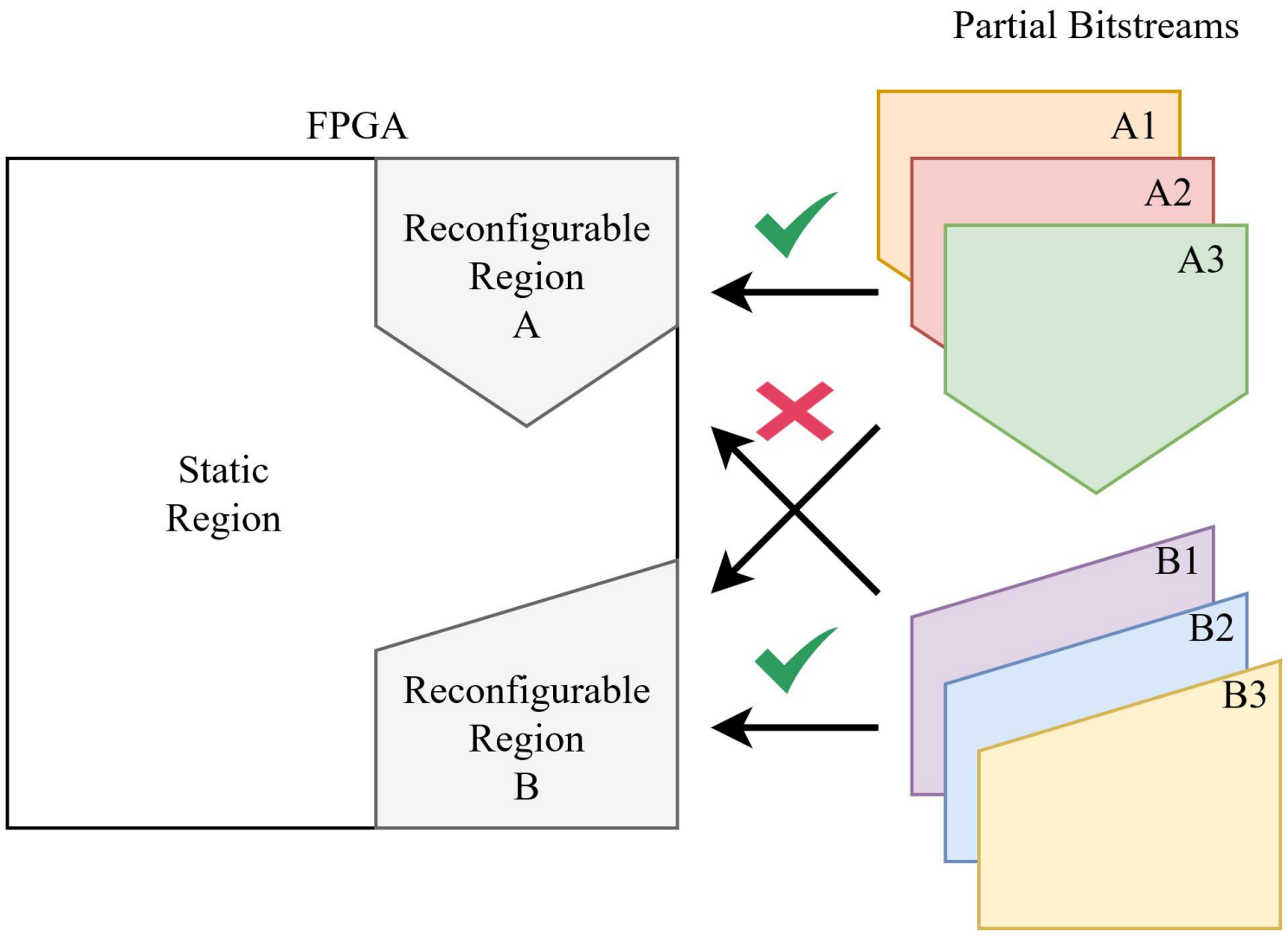


FPGAs are classified as reconfigurable hardware by changing **behavior or functionality** at the hardware logic level

Here, FPGAs are being investigated for **hardware accelerating** the CNN process at the edge devices

- Low development overhead
- Rapid Prototyping
- Application Specific Processors

FPGAs offer **Dynamic Partial Reconfiguration (DPR)**



DPR allows the device to partially reconfigure while the rest of the system remains unchanged

Modern DPR design flows use **reconfigurable regions** with **partition definitions** to be loaded after full configuration

DPR can minimize system downtime as only individual blocks of the design can be updated

Related Work

Deploying CNN Models on FPGA

FINN is developed by a Xilinx research group and specializes in low latency architectures

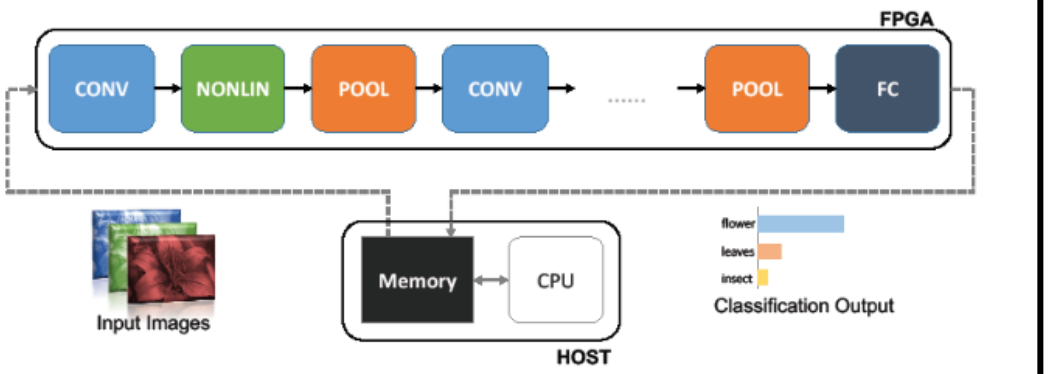
- **Streaming Architecture**
- **Modeled with HLS**

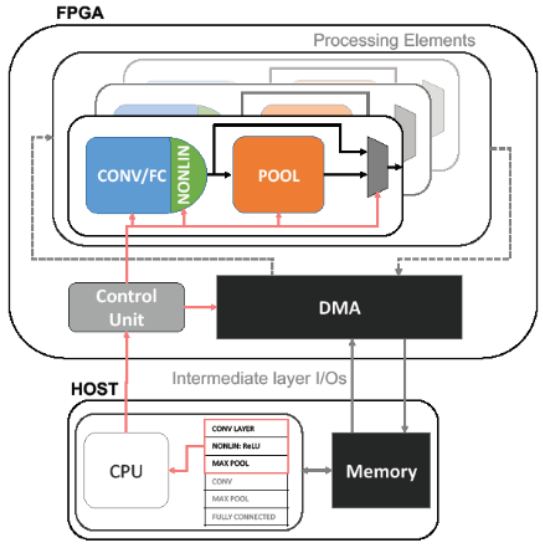
FINN Ref: https://xilinx.github.io/finn/about.html

DNNWeaver is an open-source framework that employs groups universal processing elements and a state machine to assign data to one layer of the CNN at a time

- **Single Computation Engine**
- **Modeled with Caffe**

DNNWeaver Reg: http://dnnweaver.org/





Ref: Christos Bouganis, "Tool-flows for mapping CNNs into FPGAs: Trends and Challenges"

An Energy Adaptive CNN using DPR

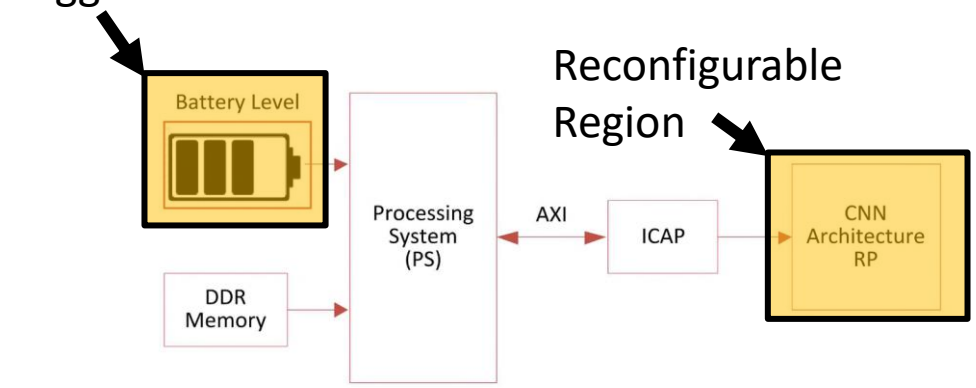


Fig. 5. DPR system block diagram.

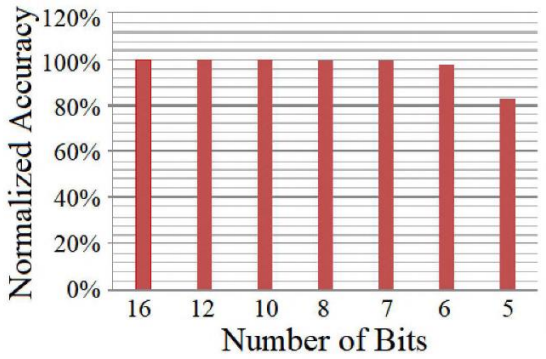


Fig. 6. Normalized accuracy of EEPS-CNN-1.

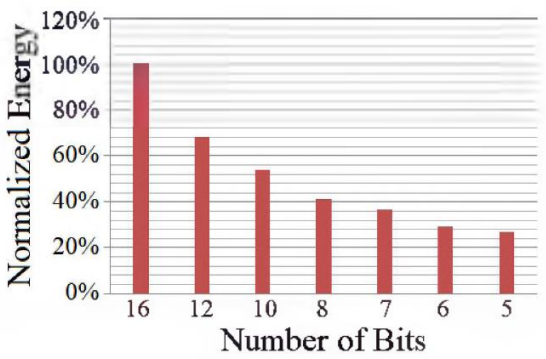


Fig. 7. Reduction in energy.

Ref: E. Youssef, H. A. Elsemary, M. A. El-Moursy, A. Khattab and H. Mostafa, "Energy Adaptive Convolution Neural Network Using Dynamic Partial Reconfiguration," (2020).

Youssef et. al developed an MNIST CNN architecture supporting DPR for **reducing processing bit precision**
This reconfiguration **decreased dynamic power consumption along with model accuracy**

The energy consumption of this model at 16-bit processing was shown to be 11.25µJ per image
When configured to 5-bit processing, 3.02 µJ per image while retaining 81.74% accuracy

This model utilizes DPR for configuring the **full CNN pipeline rather than by layer/PE**

Convolution Processor Using Parallel Pipeline

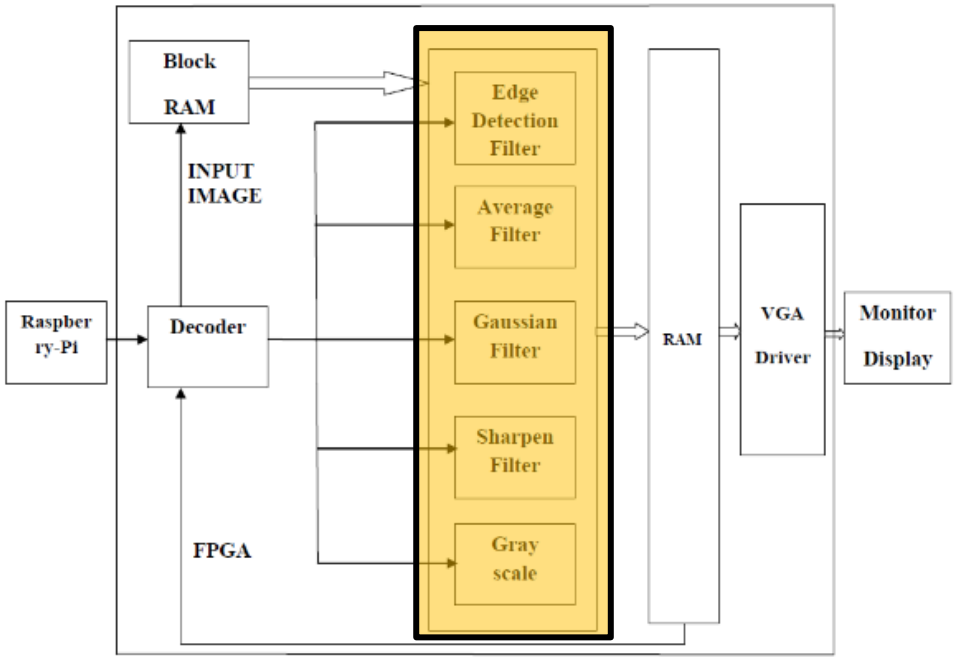


Figure 1: Proposed System Architecture

Figure 3: Input Image

Figure 4: Gaussian Filter Output

Figure 5: Sharpen Filter Output

Figure 6: Grayscale Output

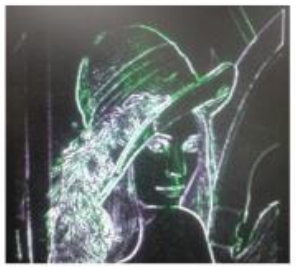


Figure 7: Edge Detection Output

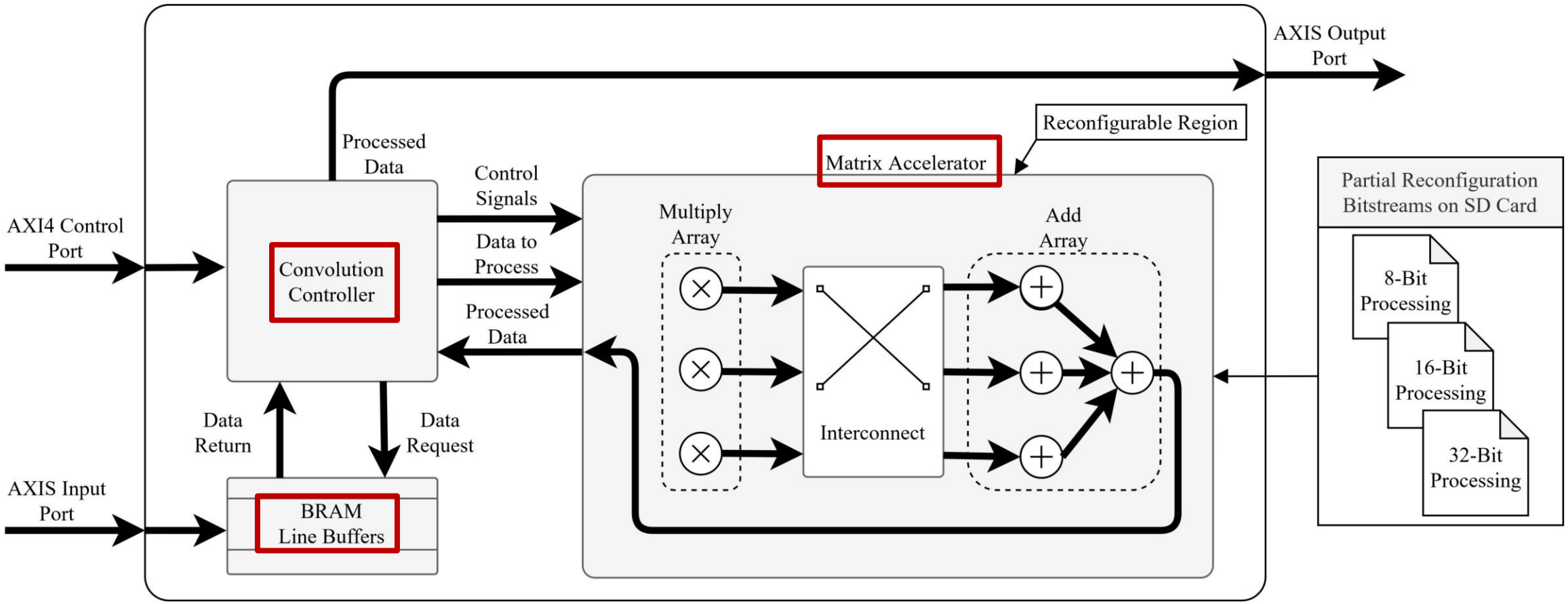
Figure 8: Average Filter Output

Rupani et. al present a convolution processing model which utilizes parallel data paths for handling flexible requirements of CNN

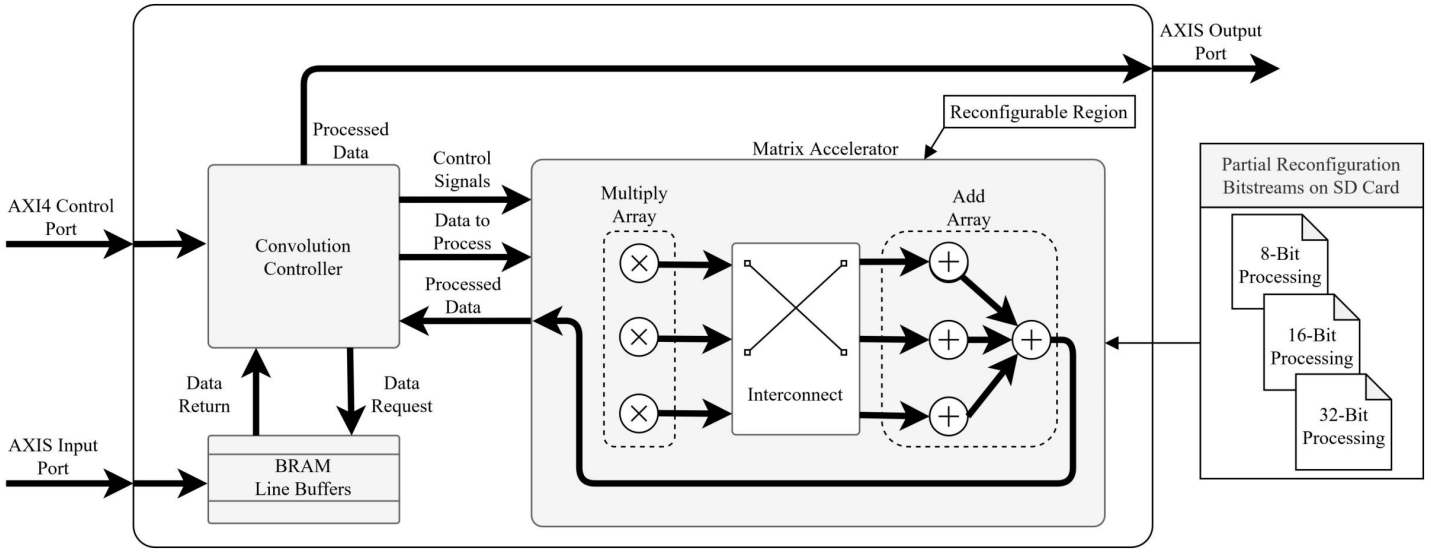
Using this method means when one filter is selected, remaining hardware is idle and unavailable for use

Ref: Rupani, Ajay & Whig, Pawan & Sujediya, Gajendra & Vyas, Piyush. "Hardware Implementation of IoT-Based Image Processing Filters," (2018).

Proposed Convolution Processing Element



CPE features a **convolution controller**, **line buffers**, and **matrix accelerator**

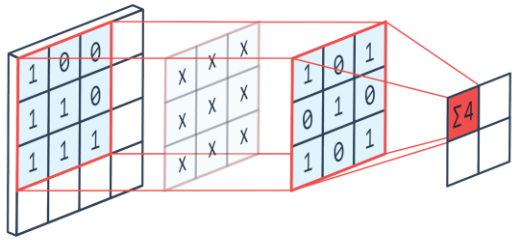


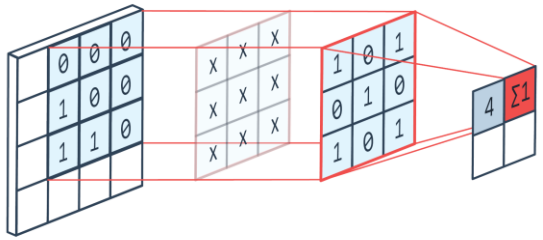
Partially Reconfigurable Parameters
Data Precision
Hard Synthesis Parameters
Process Channel Count
Kernel Size
Max BRAM Bank Width

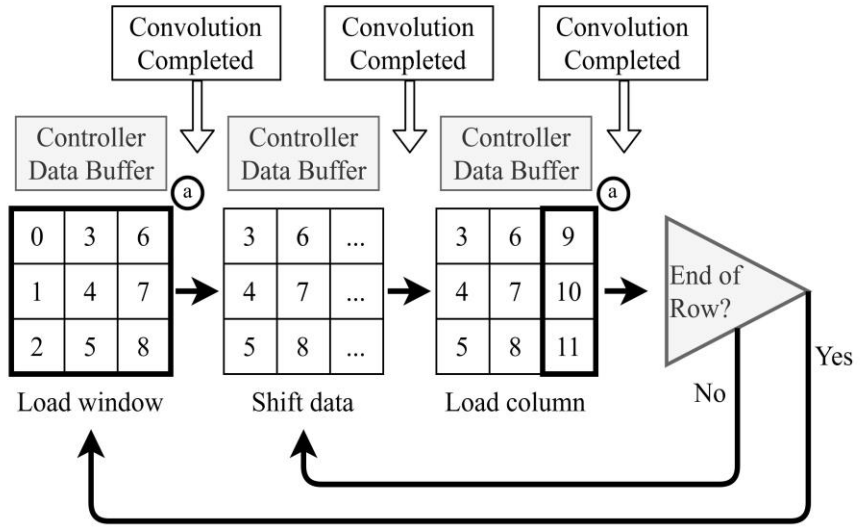
The proposed Convolution Processing Element utilizes **DPR at a processing element level**

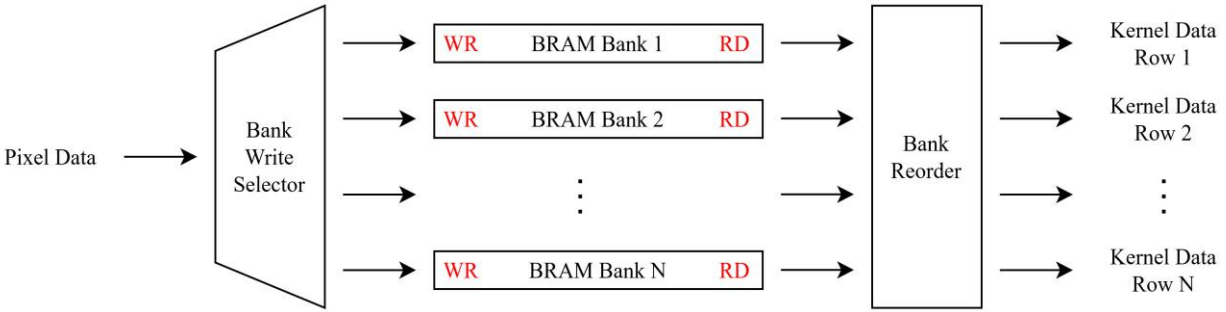
Traditional FPGA methods implement parallel data paths, fully reconfigure, or use **DPR at a full pipeline level**

Only convolution layer of CNN is implemented/evaluated









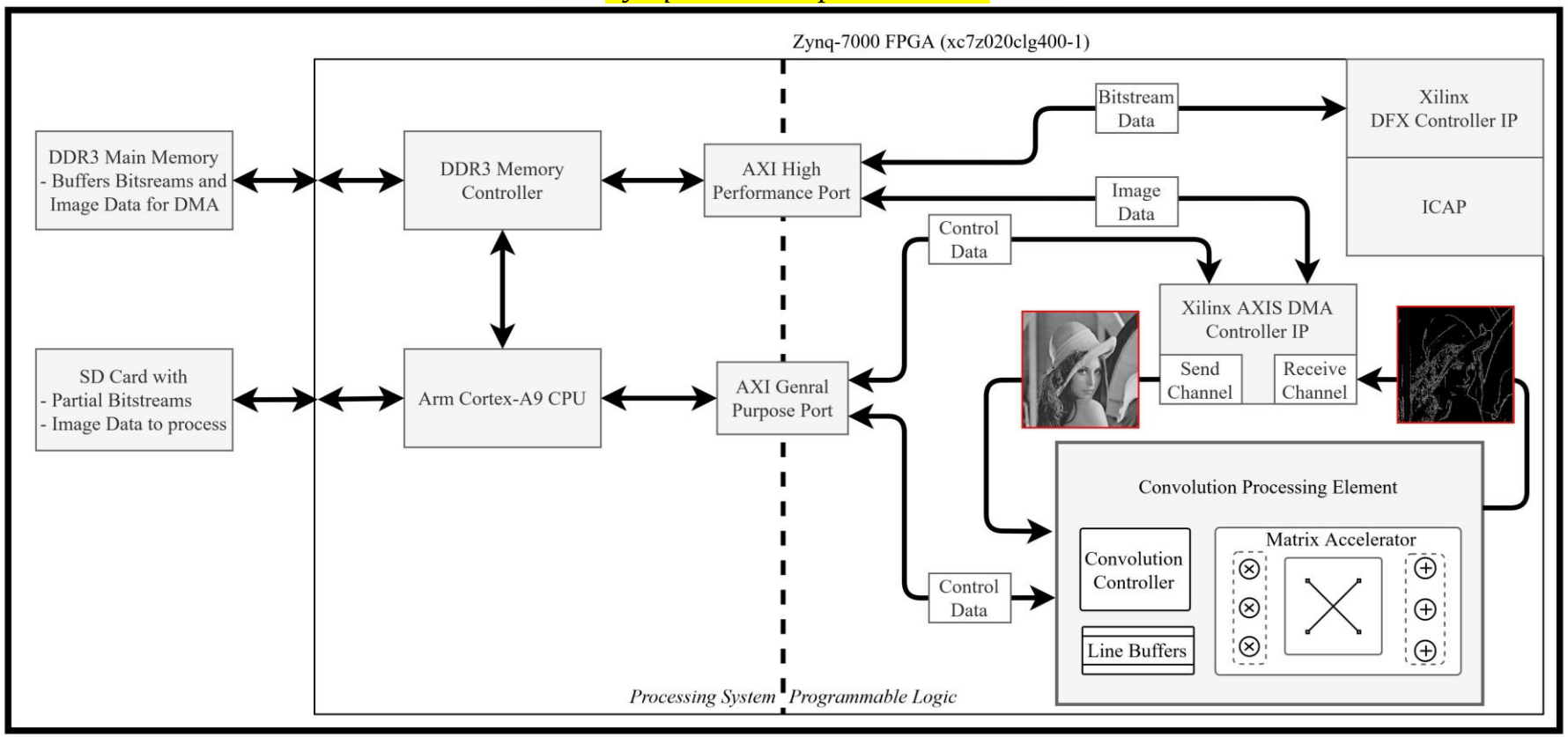
The convolution controller handles the data stream with an internal state machine

For a kernel of size N, the line buffers temporarily store N rows of input map as each convolution requires NxN data points from N rows and N columns

Line buffers are implemented to infer into BRAM as to not waste available device logic

Hardware Implementation & Comparisons

Pynq-Z2 Development Board



	Hard Config 1	Hard Config 2
Partially Reconfigurable Parameters		
Data Precision	8-,16-,32-Bit	8-,16-,32-Bit
Not Partially Reconfigurable Parameters		
Process Channel Count	1 Channels	3 Channels
Kernel Size	3x3	3x3
Max BRAM Bank Width	1800 px	1800 px

*Hard configured for **3 channels** in comparison*

TABLE II
THREE CHANNEL HARDWARE UTILIZATION BREAKDOWN

	Component	LUTs	DSPs
	Controller	5503	0
32-bit	Matrix Accelerator	271	27
	CPE	6316 (11.87%)	81 (36.82%)
16-bit	Matrix Accelerator	365	9
	CPE	6610 (12.42%)	27 (12.27%)
8-bit	Matrix Accelerator	721	0
	CPE	7741 (14.55%)	0 (0.00%)

TABLE III
HARDWARE UTILIZATION COMPARED TO OTHER DESIGNS

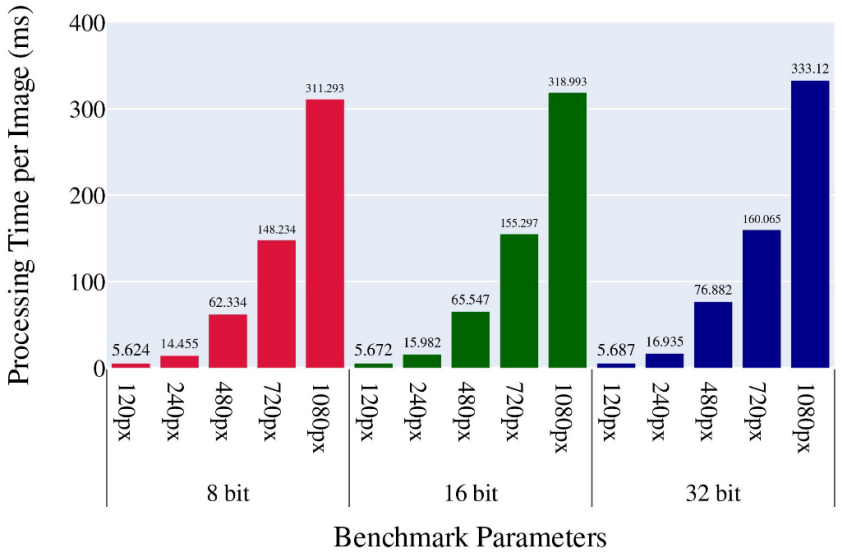
	[14] (Data reuse)	[16]	CPE	CPE	CPE
Data Type	N/A	N/A	Integer	Integer	Integer
Precision	16-bit	N/A	32-bit	16-bit	8-bit
Kernel Size	3×3	3×3	3×3	3×3	3×3
LUTs	684	38069	5291	5398	5785
FFs	672	18557	3762	3597	3661
BRAMs	N/A	72	6	6	6
DSPs	18	3	27	9	0

At 32-bit configuration, **DSP utilization reaches 36%** of total available on device

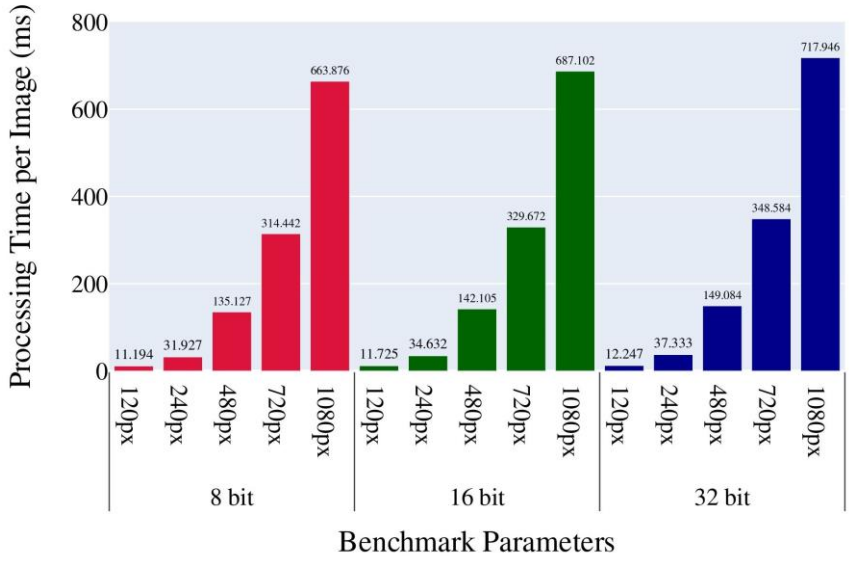
- This is a limited resource for larger systems so lower precisions should be considered

Rupani et. Al [16] use a design which buffers the full image which causes high BRAM utilization

System Operating Frequency: 100 MHz



Single Channel Config (Jupyter Notebooks)



Three Channel Config (Jupyter Notebooks)

TABLE IV
CPE RUNTIME ANALYSIS ON 64X64 INPUT MAP

CPE Channels	Input Channels	Conv. per Frame	Cycles per Frame
1	1	3969	31190
1	3	11907	93570
3	3	11907	47070
CPE Channels	Input Channels	Process FPS	Process Latency
1	1	3206	N/A
1	3	1068	N/A
3	3	2124	N/A

Three Channel Config (Vitis)

When testing with **Jupyter Notebooks** (Left & Middle), **reconfiguration was completed using JTAG**

Benchmarking was moved to **Vitis** (Right) which enabled use of the **ICAP for reconfiguration**

Changing software environments does not affect hard processing capabilities.

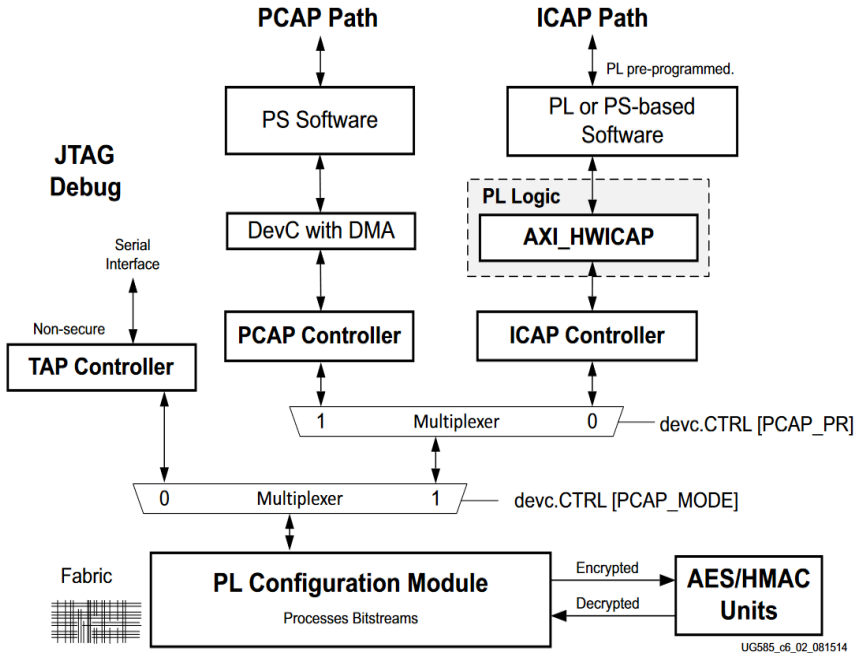


Figure 6-2: **PL Configuration Paths**

Ref: Xilinx UG585 Zynq-7000 SoC Technical Reference Manual

Table 8-4: **Maximum Bandwidths for Configuration Ports in 7 Series Architectures**

Configuration Mode	Max Clock Rate	Data Width	Maximum Bandwidth
ICAP	100 MHz	32 bit	3.2 Gb/s
SelectMAP	100 MHz	32 bit	3.2 Gb/s
Serial Mode	100 MHz	1 bit	100 Mb/s
JTAG	66 MHz	1 bit	66 Mb/s

Ref: Xilinx UG909 (v2018.1) Partial Reconfiguration

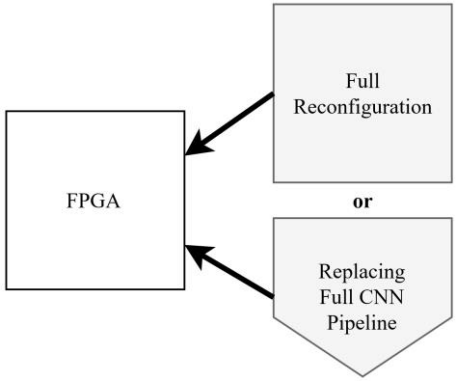
FPGA has several ways to programmed on boot that vary with device family

The Zynq device used in implementation allows programming over Serial, JTAG, ICAP, and PCAP

The port bandwidth **varies drastically**

$$Reconfiguration\ Time = \frac{Bitstream\ Size}{Port\ Bandwidth}$$

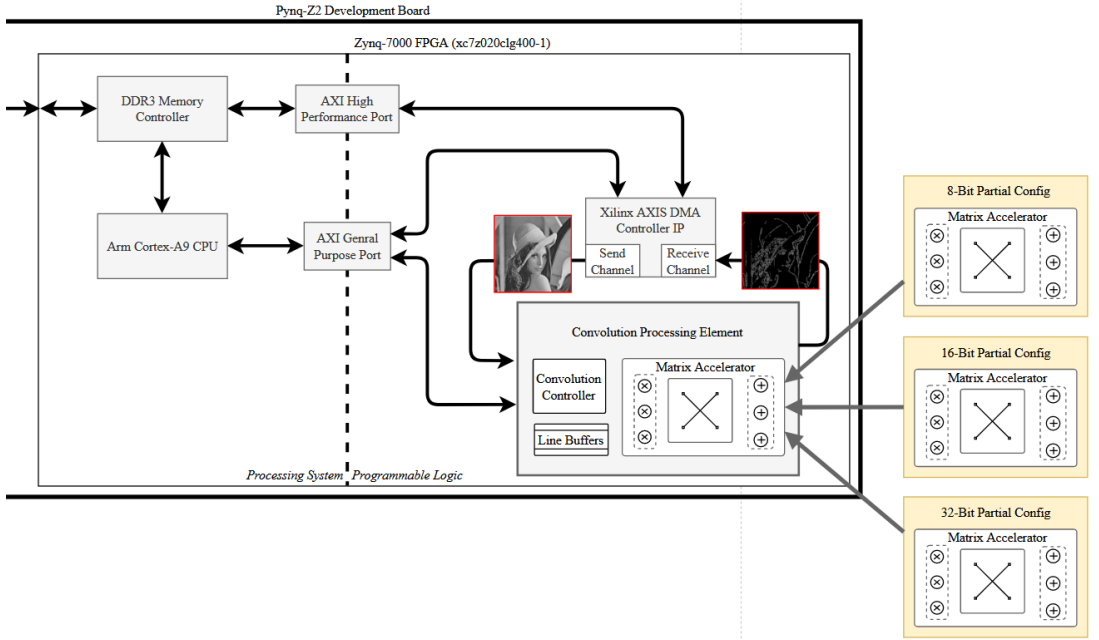
(Traditional)
Full Reconfiguration
Using JTAG



Bitstream Size:
32,364,512 b
Port bandwidth:
66 Mb/s

Reconfiguration Time:
490.371 ms

(Proposed Implementation)
Partial Reconfiguration at
processing element level
Using ICAP



Bitstream Size:
1,766,976 b
Port bandwidth:
3.2 Gb/s

Reconfiguration Time:
0.552 ms

As traditional methods would reconfigure the full device via JTAG,
using DPR at a processing element level reduces configuration time by 800x

Contributions

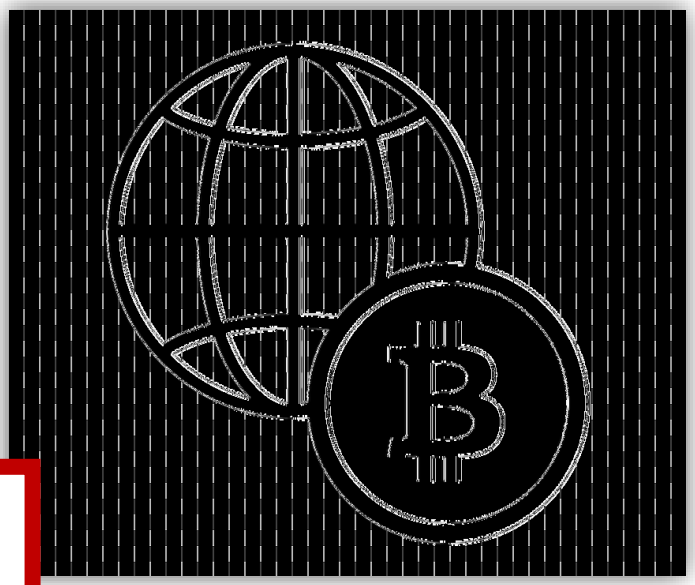
A design architecture which **enables system downtime to be minimized** if pipelines are strategically queued and reconfigured

A design architecture can be elaborated further to **reuse hardware** for algorithm changes or process parallelization at lower precision

Conclusions

FPGAs enables **resource efficient processors to be developed for application specific needs**

FPGAs also offers **Dynamic Partial Reconfiguration** which greatly suits the **computationally constrained systems** found in IoT embedded devices



Thank You

(Images processed using CPE)