Stillwaters AI

ARTIFICIAL INTELLIGENCE BASED CHIP DESIGN

TRANSFORMERS ON CHIP

Hello My Family, Friends, and Colleagues,

We at Stillwaters have some great news to share but, a little reflection first.
The year gone by has put our whole life in perspective. The lockdown came as a shock and just like all of us going through the pandemic, I too felt the frustration building up. Focussing on work and more importantly, trying to give back in whatever way I can was the only way I saw to salvage my sanity.

We immediately put together a small team at Stillwaters to arrange for Blood/Platelets/Beds/Medicine/ICU for people in need. Initially, it was a losing battle, but slowly we started turning it around. I must thank everyone who we worked with for this to be possible. I am personally deeply indebted to all at Stillwaters and LIONs club.We would love to join hands with all of you in the future to make our efforts more effective.

Back to the great news. Our tech team did what it does best. We have cracked a very difficult technical problem for our clients. To put it simply, it reduces the latency of speech-based transformers. Also, The way we have done it is quite novel.

There are 2 parts to it

- **Mapping the compute to hardware resources of the FPGA chip**
- **Designing the Chip and its layout specifically for the load**

For non-techies, the closest analogy I can think of is,
imagine having a vehicle designed to wrap around(shrink wrap) for the load(passengers, luggage, travel time, etc.) it carries to save gas and other resources.
Did I mention this is our greatest work so far?

Warm Regards,

VEENA MOHIT

CEO & Founder | STILLWATERS AI

Stillwaters AI

We present the use of Artificial Intelligence to design FPGA chips for Ultra-Low-Latency based speech transformers.
Our brief was **to reduce latency from TTS(Text-to-Speech) and STT(Speech-To-Text) systems.** This started as an experiment to lower latency for a seq-to-seq LSTM based speech synthesis system.

Post a thorough analysis, we broke it down into:
1. Mapping the matrix multiplication of deep learning transformers into an FPGA chip(Transformers On Chip(part-1))

2. Co-designing the FPGA chip for the appropriate workload(ChipDesign(part-2)).

In **part-1 (Transformers On Chip(part-1))** we look specifically at resources available on an FPGA chip and a few different strategies of hardware mapping of the workload(matrix multiplication) on the above chip.

We have taken a traditional **(TTS)Text-to-speech** and **(STT)Speech-to-text** system and spun it on a **microchip**.

In the pipeline of a traditional TTS system there is a Seq-to-Seq LSTM layer rearing its “**ugly latency head**”. It takes approximately **tens of thousands** iterations of this network to generate a second of audio. Sometimes this latency is not acceptable.

LSTMs have played their part in the Deep Learning history and it is time for them to pass on the baton to the new King, **The Transformers.**

STT (SPEECH-TO-TEXT)

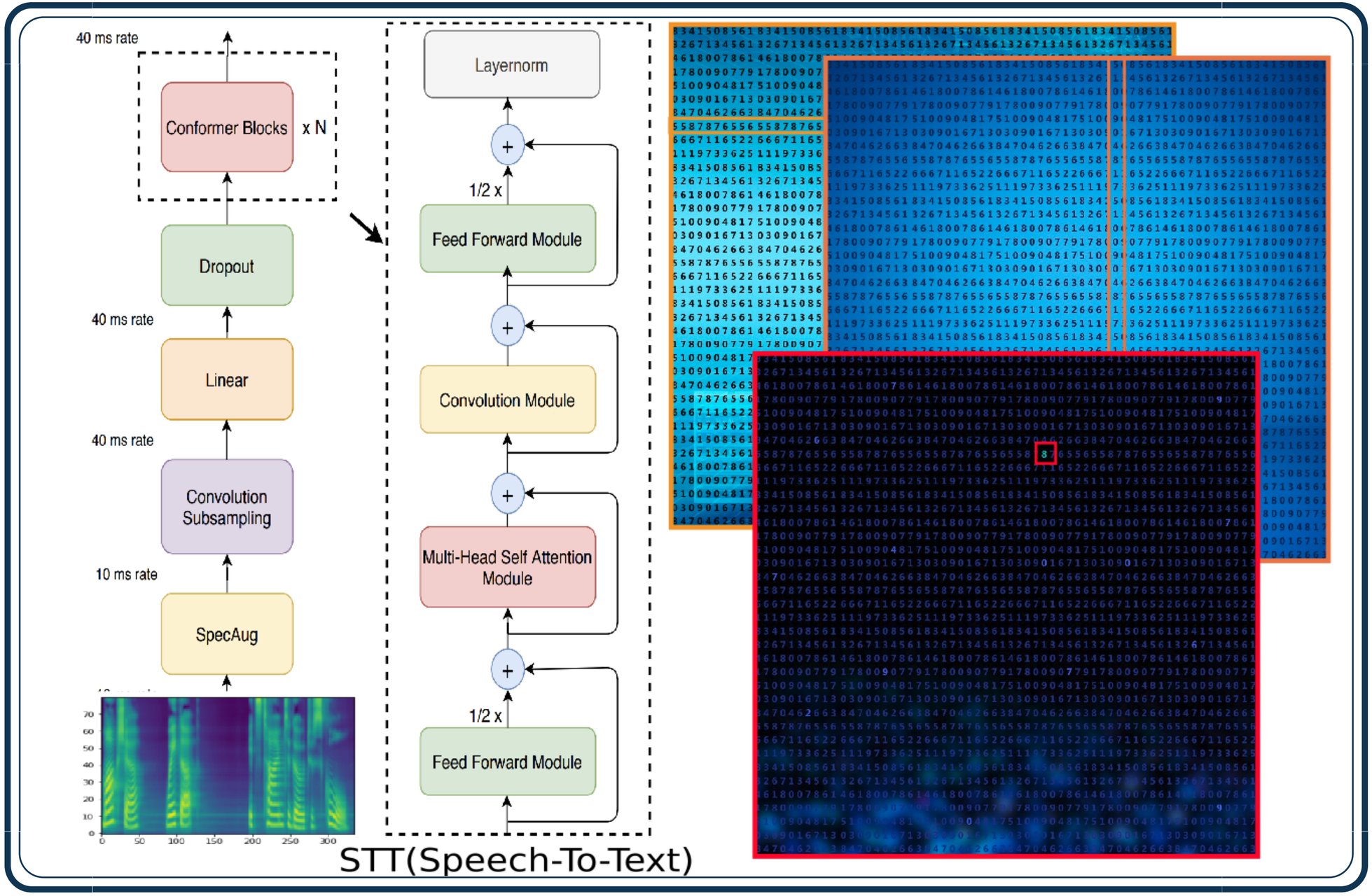
Firstly, we took the STT based on **Conformer** (a **transformer** variant) and rewrote it using **Trax**.

Why Trax? Well it has an optimizing **Just-In-Time compiler** for **Machine learning**. We have extended that for FPGA. But more on that later.

It was a huge improvement on the LSTM variant. While there was a marked improvement, it wasn't enough. And this was running on a 2020 GPU.

Transformers are essentially **(MHSA)Multi-headed-Self-Attention** and a **(FFM)feed forward Module**. Both these are **matrix multiplications of grand and varying scale**. The Conformer has an additional **CM(Convolutional Module)** which is a matrix filter operation.

STT (SPEECH-TO-TEXT)



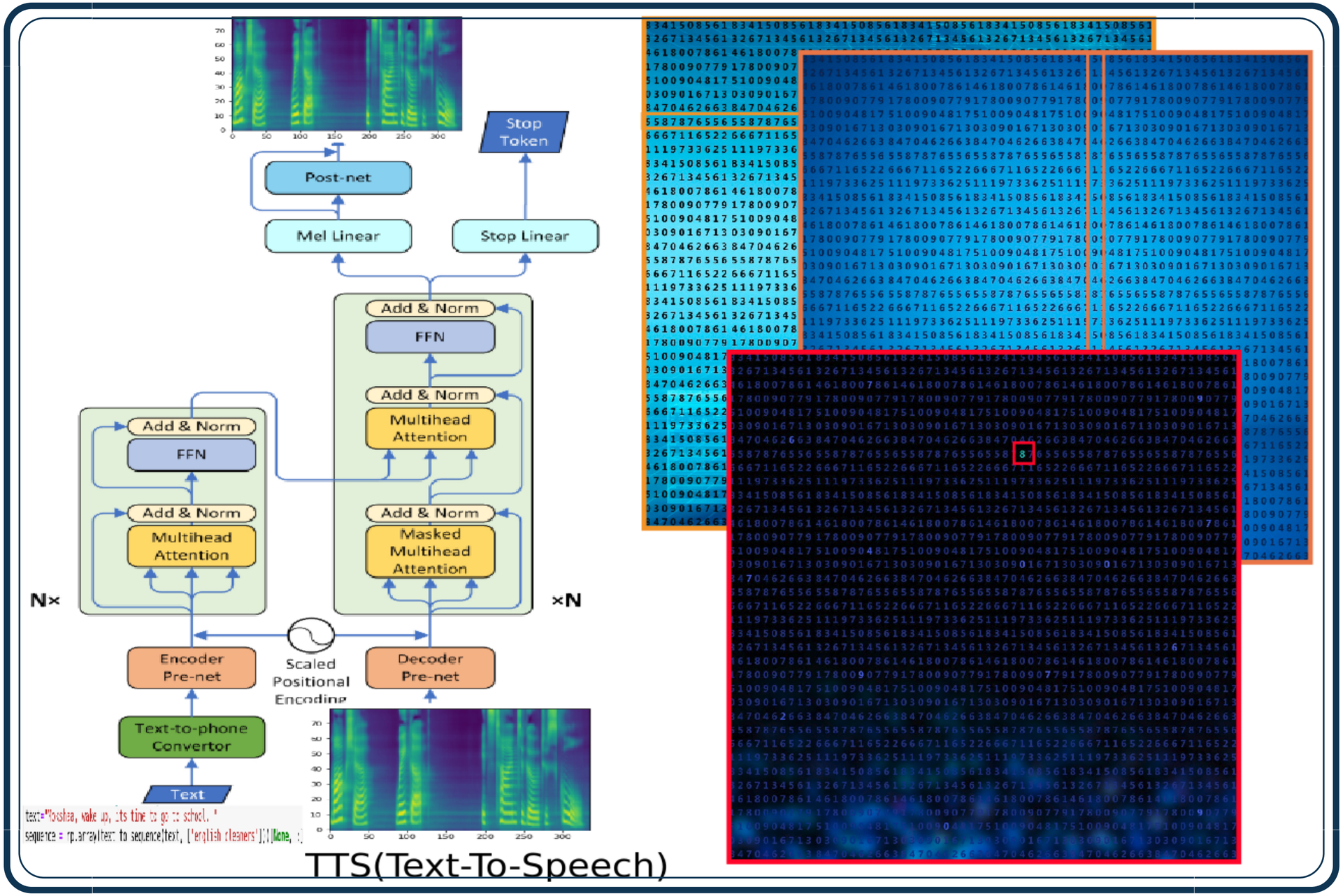
TTS (TEXT-TO-SPEECH)

Secondly, we took the TTS based on the transformer and did exactly the same.
Again it was a decent improvement on the LSTM version. But, not enough. Again, this was running on a 2020 GPU.

GPUs are great at handling the “grand” part despite some transformers having billions to even trillions of parameters.

- The “varying” part is proving extremely tricky for GPUs. Both MHSA and FFM have varying parallelism requirements(from one model to another).
- Also, the standard precision leads to lots of wastage energy. GPU’s SIMD/SIMT **lock-step style execution model** makes matters worse. Lock step style execution is prohibitive in terms of energy wastage.
- Last but not least, the extra control structure on the GPU is not useful(wasteful) for matrix multiplication.
- GPU is a general purpose device and for something as specific as tuning a massive matrix multiplication it is found wanting. It is not a natural fit would be the right phrase.
- All of the above contribute to tons of electricity, produce a lot of heat, and use fans for cooling.
  - And a large number of environments where we apply deep learning like speech devices, self-driving cars, production lines etc, are not agreeable to it.
  - This also makes maintenance and life expectancy of a GPU an issue.

TTS (TEXT-TO-SPEECH)

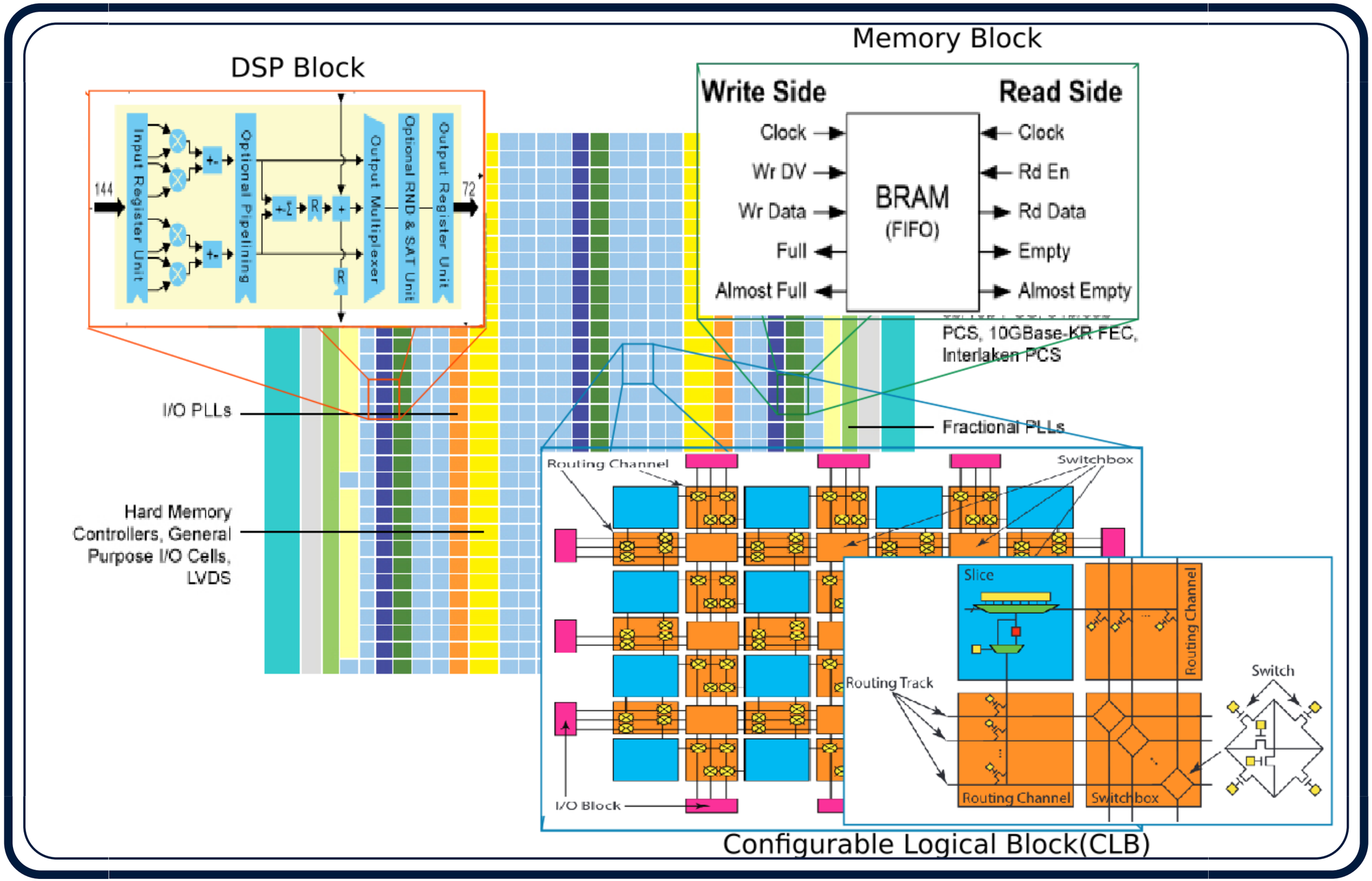


FPGAS AS MODERN, DIGITAL SHAPE-SHIFTERS

The CPUs and especially GPUs are nearing their transistor limit.
The world is moving toward specialized hardware to meet the exponentially growing demand for computers:.

- Field-programmable gate arrays (FPGAs) are reconfigurable computer chips that can be programmed to implement any digital hardware circuit.
- FPGAs consist of an array of different types of programmable blocks (logic, IO, and others) that can be flexibly interconnected using prefabricated rout-ing tracks with programmable switches between them.
- The bit-level reconfigurability of FPGAs enables implementation of the exact hardware needed for each application (e.g. datapath bit-width, pipeline stages, number of parallel compute units, memory subsystem, etc.) instead of the fixed one-size-fits-all architecture of CPUs or GPUs.
    - Consequently, they can achieve higher efficiency than CPUs or GPUs by im-plementing instruction-free streaming hardware

FPGAS AS MODERN, DIGITAL SHAPE-SHIFTERS



FPGAS AS MODERN, DIGITAL SHAPE-SHIFTERS

FPGA RAM

- An FPGA BRAM consists of an SRAM-based memory core, with additional peripheral circuitry to make them more configurable for multiple purposes and to connect them to the programmable routing
- FPGA vendors can add circuitry that allows designers to repurpose the LUTs that form the logic fabric into additional RAM blocks.

DSP Blocks

- With the prevalence of multipliers in FPGA designs from key application domains, and their lower area/delay/power efficiency when implemented in soft logic, they quickly became a candidate for hardening as dedicated circuits in FPGA architectures.

OPTIMIZING ON HARDWARE ACCELERATOR

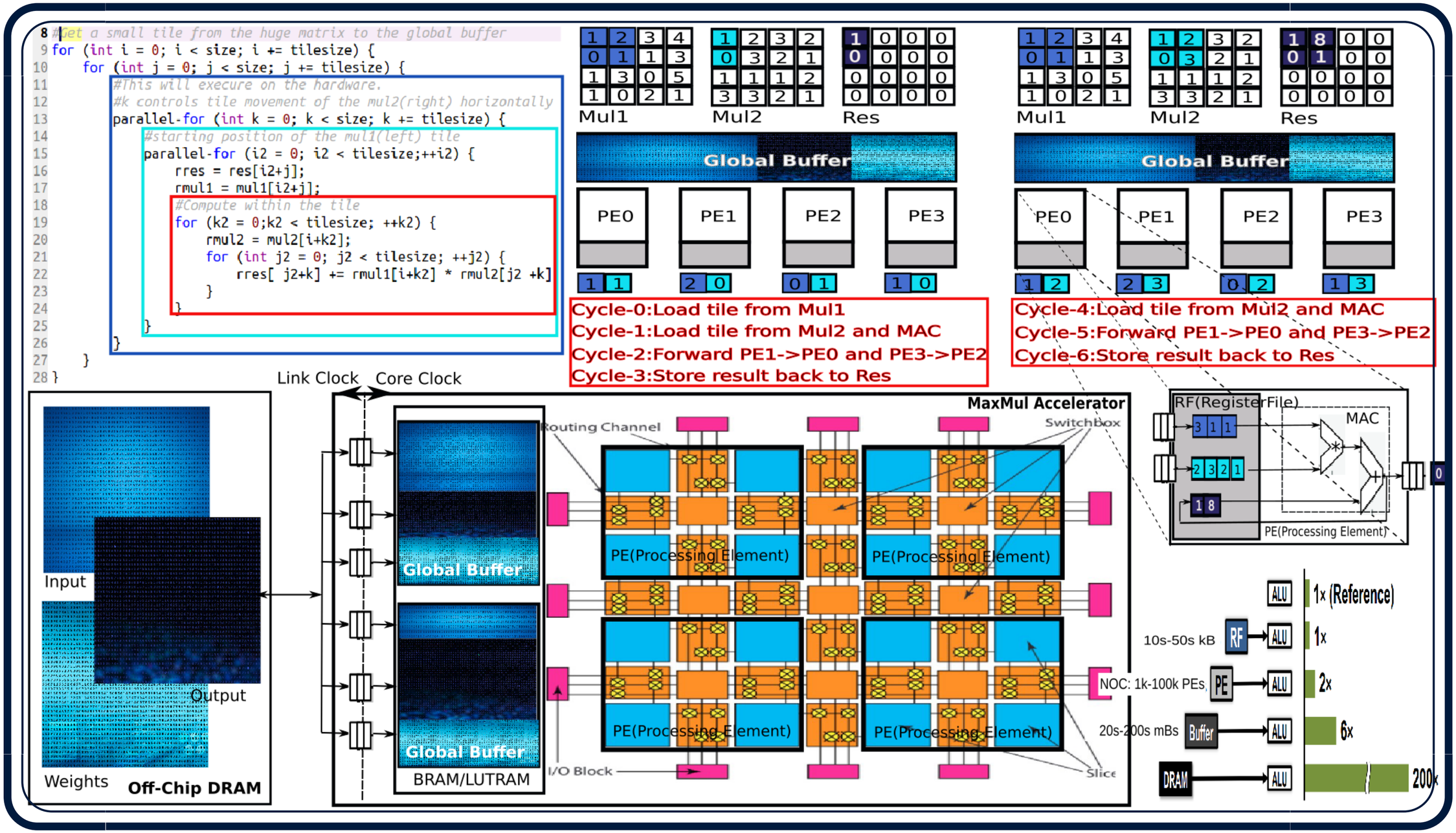
The idea here is to hardware accelerate the huge matrix multiplication which is the core of Transformers. Given the size of the matrices in transformers (a few billion parameters in 2020), there is practically no chance of them fitting inside a top end FPGA board, not for at least the next decade.

So the trick is to chop these huge matrices being multiplied into smaller tiles and reuse these tiles as much as possible before moving on to the next tile. One of the oldest tricks in a new Avatar.

There are 4 key points of FPGA Hardware Optimization:

- Play to FPGAs strength, no extra control logic
- Minimize access to OFF-chip RAM by having multi level on chip buffer made of BRAM and LUTRAM
- Maximize the compute parallelism (PE)ProcessingElements
- And the most vital, balance the above 2.
  - PEs must be well fed and not starving
  - Balance cached data with streaming(weights(Mul1) may be cached more and the input will be streamed more (Mul2))

OPTIMIZING ON HARDWARE ACCELERATOR (MAC STYLE)



OPTIMIZING ON HARDWARE ACCELERATOR (MAC STYLE)

- The hardware mapping is the most fabulous part carried out by our mapper. It takes the code below and unrolls the MaxMul loop into the Hardware.
- There are 2 Styles
  - MAC Style(Multiply Accumulator)
    - For a 4X4 matrix a 2X2 PE array as shown will take 24 cycles give or take.
    - For a 4X4 matrix a 4X4 PE array as shown will take 6 cycles give or take.
  - SA Style (Systolic Array)
    - For a 4X4 matrix a 2X2 PE array as shown will take 24 cycles give or take.
    - For a 4X4 matrix a 4X4 PE array as shown will take 6 cycles give or take.
- We retrofit this mapper to trax.

OPTIMIZING ON HARDWARE ACCELERATOR (SYSTOLIC ARRAY STYLE)

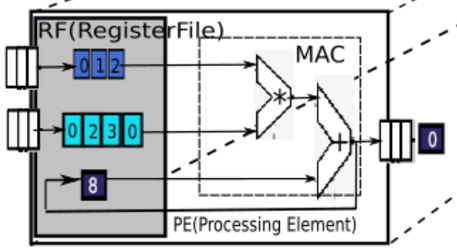
Same as above except the microarchitecture uses systolic array design with The PEs being pipelined.

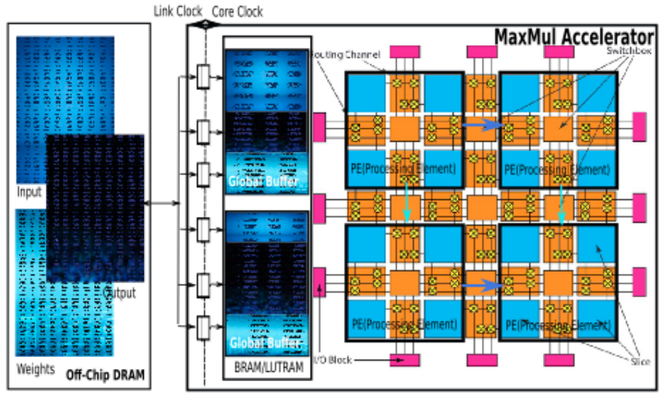
Follow the color coding especially the data movement(light and dark blue boxes and arrows) into the **PEs** and data forwarding among **PEs**(Processing Elements).

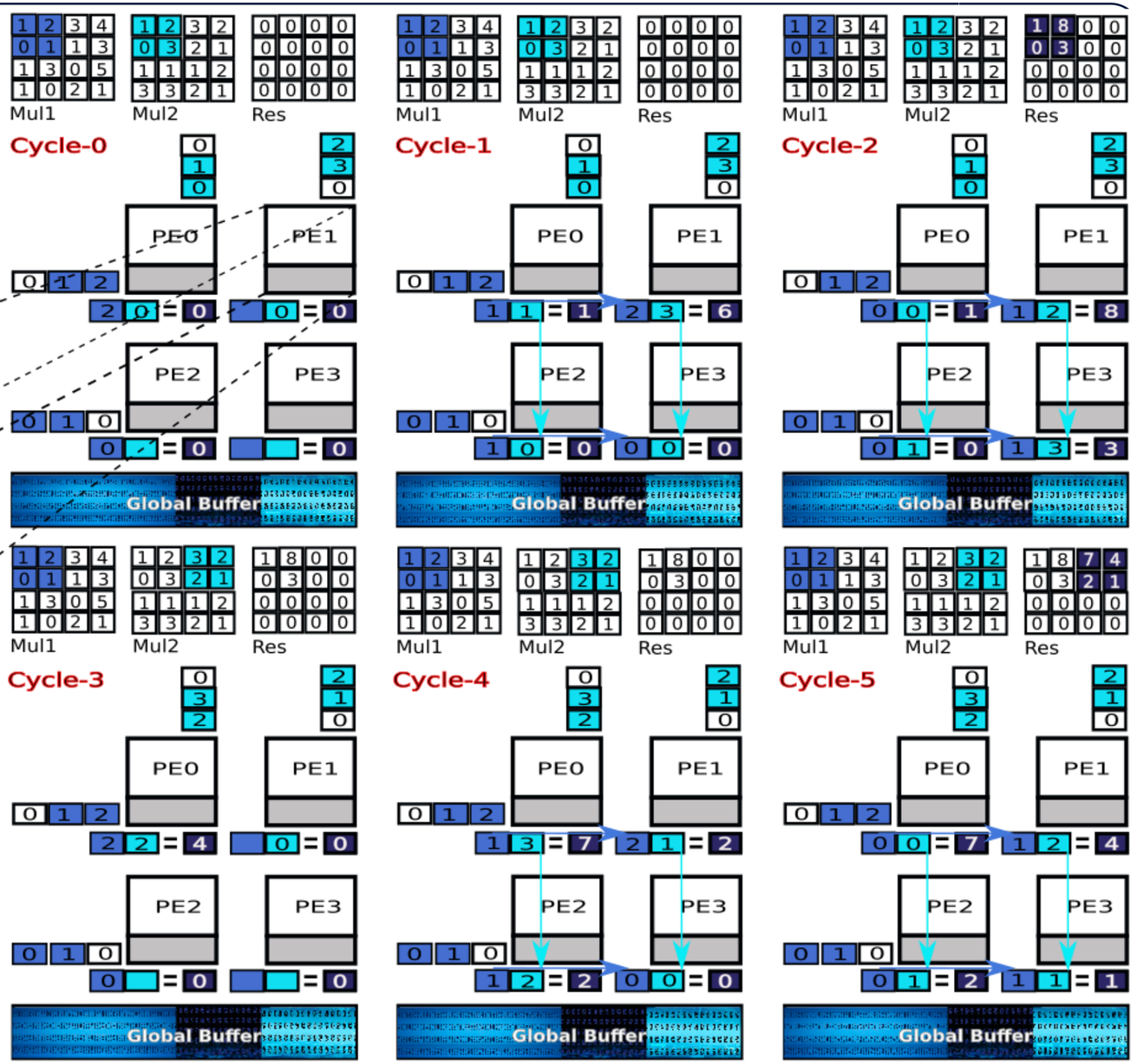
The **NOC**(Network On Chip) architecture which is not shown, is slightly different for both the **MAC Style**(Multiply Accumulator) and the **SA Style** (Systolic Array).

OPTIMIZING ON HARDWARE ACCELERATOR (SYSTOLIC ARRAY STYLE)

```
#Get a small tile from the huge matrix to the global buffer
for (int i = 0; i < size; i += tilesize) {
    for (int j = 0; j < size; j += tilesize) {
        #This will execure on the hardware.
        #k controls tile movement of the mul2(right) horizontally
        parallel-for (int k = 0; k < size; k += tilesize) {
            #starting position of the mul1(left) tile
            parallel-for (i2 = 0; i2 < tilesize;++i2) {
                rres = res[i2+j];
                rmul1 = mul1[i2+j];
                #Compute within the tile
                for (k2 = 0;k2 < tilesize; ++k2) {
                    rmul2 = mul2[i+k2];
                    for (int j2 = 0; j2 < tilesize; ++j2) {
                        rres[ j2+k] += rmul1[i+k2] * rmul2[j2 +k]
                    }
                }
            }
        }
    }
}
```







VARIABLES IN THE DESIGN

The SA Style (Systolic Array) works better for bigger tiles when it's pipelines are fully fed for longer duration. But not too big as the NOC feeding the buffers has its own limitations. The **size of the tile** and indeed the **style of the microarchitecture(MAC or SA)** are one of the many design variables that need to be configured for best results. These in turn depend on quite a few more design variables, some of them are due to **software constraints** (Matrix size etc.) and others are **hardware constraints** (BRAM etc). Here are a few important design variables

- The **SGB**(size of Global Buffer) based on available BRAM and LUTRAM.
  - More LUTRAM would mean bigger buffers, better reuse, but fewer PEs and lesser parallelism.
- The **LGB**(Layout(Locality) of GLobal Buffer)
- The size of the **MT** (Matrix Tile) and **PET**(PE Tile)
- The size of **RF**(RegisterFile)
  - As a general rule, bigger/multi-level caching(SGB/NOC/RF are like L3/L2/L1) would better lookup performance but increase data replication and eat into logic space(PEs)
- The layout of **NOC**(Network on Chip)
- **With 100s of MBs to 1000 GBs of matrix sizes to choose from in modern transformers, myriad different Accelerator boards, Shape shifting configuration options on these boards, all the above variables to choose from, the configuration space is a serious problem of plenty.**

A general configuration done by a domain expert(on Transformers and FPGA board) will give us **decent improvement** over a software only implementation but, **nowhere near what the hardware is capable of** unless the configurations are done by either Ramanujam or Tesla(Only these 2, others were normal).

Stillwaters AI

ARTIFICIAL INTELLIGENCE BASED

CHIP DESIGN

Stillwaters AI

In part-2(ChipDesign(part-2)) we treat the resources on FPGA chips, Workload, and other design parameters as design variables. Finally, we feed the design variables into the Deep Reinforcement Learning agent to learn.

Post learning the expectation from Deep Reinforcement Learning agent being the **optimal placement** of blocks to maximize certain goals like latency, etc. The Deep Reinforcement Learning Algorithm is supposed to figure out a balance that **speeds up computation for tolerable accuracy losses(if at all)**. To use a boxing term, **pound for pound, the same hardware micro-designed by a Deep Reinforcement Learning Agent for carrying a specific load.**

CHIP DESIGN: LAYOUT, LOCALITY AND RESOURCE ALLOCATION

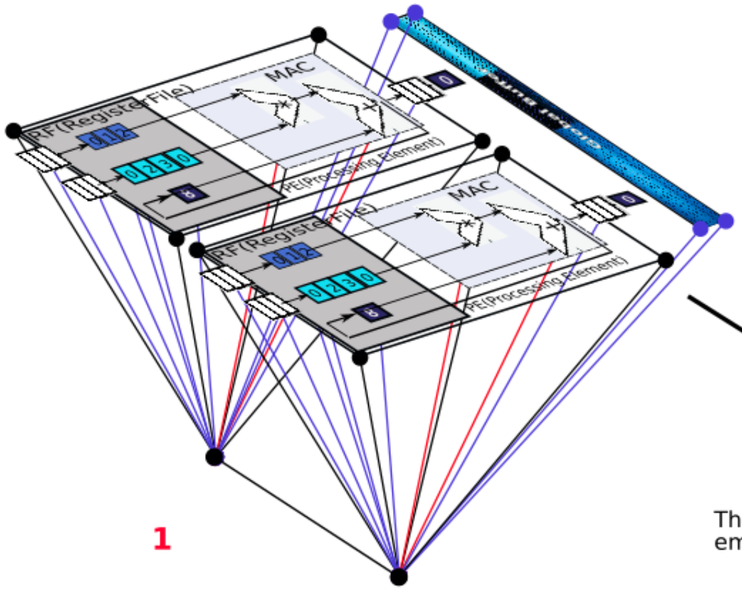
Although current FPGA accelerators have demonstrated better performance over generic processors for specific workloads, the **accelerator design space** has not been **well exploited.** One critical problem is that the **computation throughput** may not well match the memory bandwidth provided by FPGA platforms.
But the bigger issue is Hardware design or even certain elements in it are usually not a variable in application design. Hardware design is a bubble, and so is software design and these don't meet. There is a case to be made for **hardware and software** co-design. But a few more hardware specific design variables first.

- **Matrix Sparsity(MS)** is known to cost a lot. Checks can be built into the PE to reduce the overall cache requirement, but it comes at the cost of chip fabric space and end-to-end latency. It's a classic trade off.
- **Reducing Bit Width(BW) costs accuracy**
  - Recent developments point toward next-generation Deep Learning algorithms that exploit **extremely compact data types (e.g., 1bit, 2bit, 4bit, 8bit, 12bit etc.).**
- This is FPGAs' backyard, but was never done dynamically and as a whole with software in the loop.

OPTIMIZING ON HARDWARE ACCELERATOR (SYSTOLIC ARRAY STYLE)

With so many design variables identified above, this essentially is a deep search problem. Many of these design variables can assume millions to billions of values and the combinatorial space is practically impossible to go through with an exhaustive search policy. Last but not least, the entire argument for Artificial Intelligence is generalization, which is the whole point of the Design Variables.

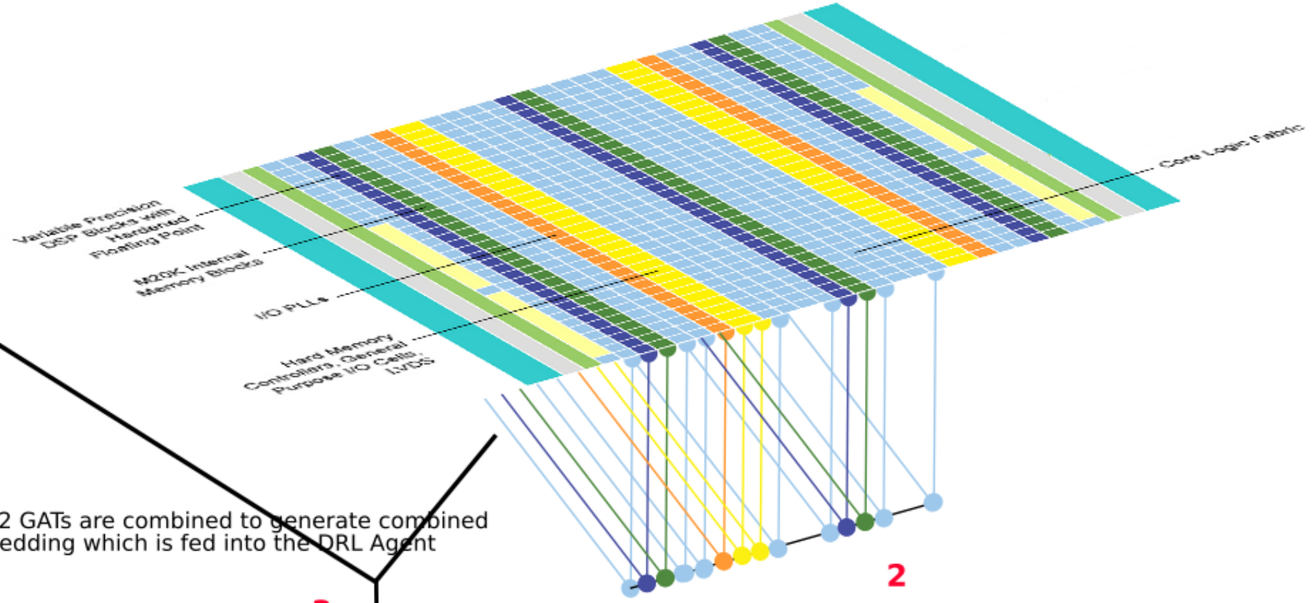
OPTIMIZING ON HARDWARE ACCELERATOR (SYSTOLIC ARRAY STYLE)



1

Express the **constraints** and **composition** of the microarchitecture as a **Graph Attention Network(GAT)**

Such complex arrangements that takes locality and other constraints into account is easier to express using **GATs.**



2

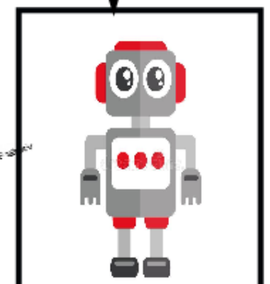
Express the Resources and Constraints of the chip as the **"Rules of Game"**

As an example the dark **blue(DSP)** and the **bottle green (BRAM)** are pre-arranged in columns right beside eachother on the FPGA chip to aid lower latency and easier integration.

As a general rule, the **design variables** identified earlier and the **Resources** and **Constraints** of the FPGA Chip are expressed as the **"Rules of Game".**

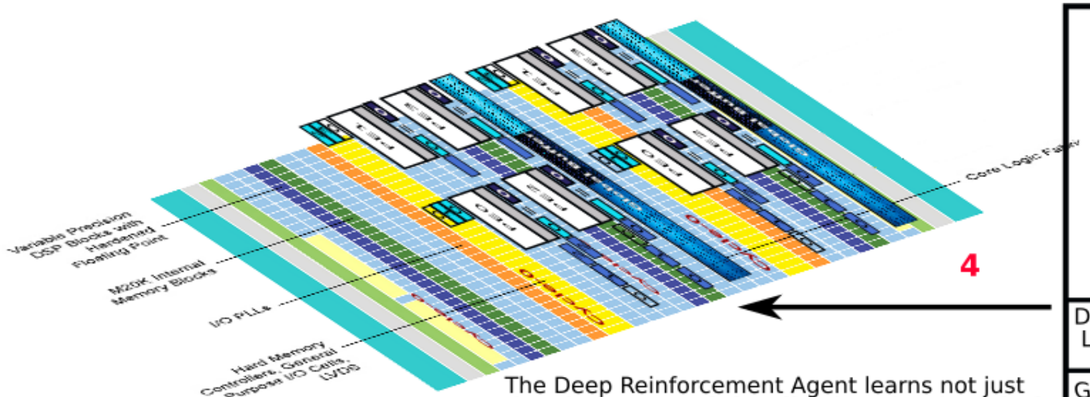
The 2 GATs are combined to generate combined embedding which is fed into the DRL Agent

3



Deep Reinforcement Learning Agent

Goal-1: Latency
Goal-2: Energy



4

The Deep Reinforcement Agent learns not just the **placement of the tiles** but also the **relative sizing of caches.**

RULES OF THE GAME

Essentially these are the **boundary conditions** asserted by the design and/or inherent in the chip. Add to that the **design variables** and we have the complete rules of the game.

1. Express the constraints and composition of the microarchitecture as a GAT(Graph Attention Network)
    a. Such complex arrangements that take locality, composition, arrangement and other constraints into account are easier to express using GATs. Our current implementation is custom but for the next iteration we'd like to use the **graph neural network.**
2. Express the **Resources and Constraints** of the **FPGA Chip** as the "Rules of Game(Alphago Zero learnt the game of go from scratch, with the rules of GO as boundary constraints)"
    a. As an example the **dark blue (DSP)** and the **bottle green (BRAM)** are pre-arranged in columns right beside each other on the FPGA chip to aid lower latency and easier integration.
    b. The **logic fabric**(FPGA fabric) can be used in **compute** or as **RAM**(LUTRAM). Both Altera/Intel and Xilinx have elected to make only **half of their logic blocks LUT-RAM capable** in their recent architectures. This is another "rule of the game". Similar to **chinese chess,** where a piece is captured and reprogrammed rather than removed from the board. A little like saying if you are not logic you are ram. And the **Deep Reinforcement Learning Agent is expected to learn it over multiple training iterations.**
    c. As a general rule, the **design variables identified** earlier and the Resources and Constraints of the FPGA Chip are expressed as the "Rules of Game".
3. We generate combined embeddings that are fed into the **Deep Reinforcement Learning Agent for Training.**
4. The **Deep Reinforcement Agent** with multiple training iterations learns not just the placement of the tiles but also the relative sizing of caches and indeed the values of the **design variables.**
    a. It goes without saying that the prediction for most design variables more than satisfies the **primary goal(latency)** and shatters the **secondary goal(energy).**
    b. But the result is fascinating for a completely different reason. This is the beginning of a recipe for the future of chip design.

CONCLUSION

What started as an experiment led to something as fascinating as this was something beyond expectation at the beginning. Now that it has, it can be extended to all kinds of FPGA chips and beyond. The goals may change but the general recipe will remain essentially the same.

WE CAN HELP YOU BUILD ARTIFICIAL INTELLIGENCE BASED SYSTEMS.

For any enquiries or to book a free seminar on any of our modules, please get in touch:

Abhishek Edachali

Marketing Head

relations@stillwaters.ai

+91 93729 55219