# System-Level Design and Integration of a Prototype AR/VR Hardware Featuring a Custom Low-Power DNN Accelerator Chip in 7nm Technology for Codec Avatars

H. Ekin Sumbul, Tony F. Wu, Yuecheng Li, Syed Shakib Sarwar, William Koven, Eli Murphy-Trotzky, Xingxing Cai, Elnaz Ansari, Daniel H. Morris, Huichu Liu, Doyun Kim, Edith Beigne

Reality Labs, Meta

**Abstract**

Augmented Reality / Virtual Reality (AR/VR) devices aim to connect people in the Metaverse with photorealistic virtual avatars, referred to as "Codec Avatars". Delivering a high visual performance for Codec Avatar workloads, however, is a challenging task for mobile SoCs as AR/VR devices have limited power and form factor constraints. On-device, local, near-sensor processing provides the best system-level energy-efficiency and enables strong security and privacy features in the long run. In this work, we present a custom-built, prototype small-scale mobile SoC that achieves energy-efficient performance for running eye gaze extraction of the Codec Avatar model. The test-chip, fabricated in 7nm technology node, features a Neural Network (NN) accelerator consisting of a 1024 Multiply-Accumulate (MAC) array, 2MB on-chip SRAM, and a 32bit RISC-V CPU. The featured test-chip is integrated on a prototype mobile VR headset to run the Codec Avatar application. This work aims to show the full stack design considerations of system-level integration, hardware-aware model customization, and circuit-level acceleration to meet the challenging mobile AR/VR SoC specifications for a Codec Avatar demonstration. By re-architecting the Convolutional NN (CNN) based eye gaze extraction model and tailoring it for the hardware, the entire model fits on the chip to mitigate system-level energy and latency cost of off-chip memory accesses. By efficiently accelerating the convolution operation at the circuit-level, the presented prototype SoC achieves 30 frames per second performance with low-power consumption at low form factors. With the full-stack design considerations presented in this work, the featured test-chip consumes 22.7mW power to run inference on the entire CNN model in 16.5ms from input to output for a single sensor image. As a result, the test-chip achieves 375 µJ/frame/eye energy-efficiency within a 2.56 mm$^2$ silicon area.

## 1. Introduction

Mobile telepresence aims for photorealistic social interactions in the Metaverse [1]. Enabled by mobile Augmented Reality / Virtual Reality (AR/VR) devices, users will be represented with their lifelike virtual avatars, referred to as "Codec Avatars" [2], to feel socially present and share experiences together even when they are physically distant. The main working model of the Codec Avatars application is provided in Figure.1. The model is implemented with two main building blocks: Encoding and Decoding [2][3]. To virtually represent faces in a realistic fashion, facial expressions of one user are captured and encoded on their headset. The encoded facial expression data is continuously transmitted to the other user(s), and then decoded and rendered to a life-like avatar on the other user's headset. The encoding and decoding blocks simultaneously work on both headsets for a realistic conversation. The shared medium can be a virtual environment for VR headsets today, and could be physical locations of the users for AR glasses in the future.

Codec Avatars application demands high visual performance with low latency for realistic interactions. For a smooth rendering and mobile telepresence quality, encoding and decoding blocks have to achieve high frame-rates on the captured video images coming from the sensors on the device. As a result, the mobile AR/VR devices have to provide high visual processing capability reaching a target frames per second (fps). Mobile AR/VR devices, however, have very limited physical space and tight form-factors to fit a variety of wearable device requirements. Therefore both battery and the available SoC area are very limited. To enable the Codec Avatars workload on mobile AR/VR devices, the headset system has to then achieve high visual performance and fps on a very power and area limited hardware platform. A custom silicon solution would provide the design flexibility to achieve such low power and high performance processing capability at the strict low form factors.

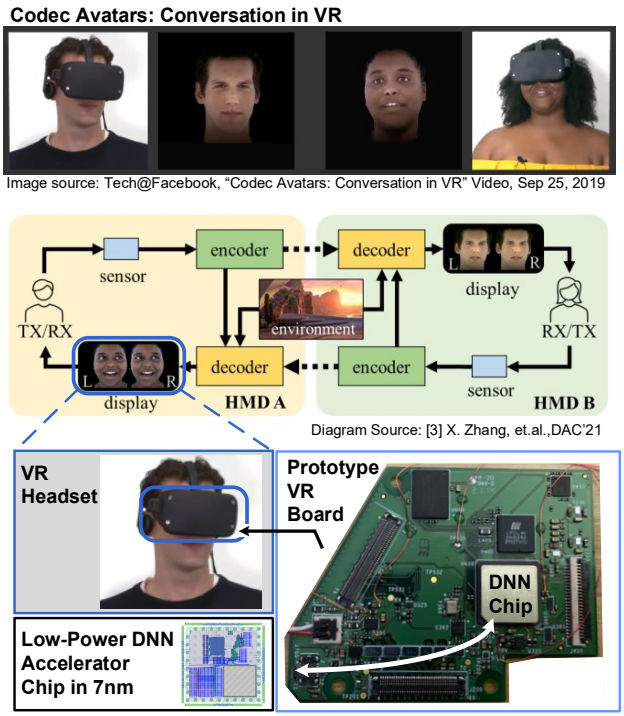


Fig. 1. Photo-realistic mobile conversation with Codec Avatars, application-level diagram of the model with two users, and a prototype hardware featuring custom DNN accelerator test-chip targeting Codec Avatars demonstration on a VR headset.

Local, near-the-sensor computation minimizes data transmission to save energy and enables stronger privacy solutions in the long run. To meet the combined challenges of performance, power, and area of running Codec Avatars on mobile AR/VR devices, we built a VR prototype system where a part of the workload is off-loaded to a custom-built low-power Deep Neural Network (DNN) accelerator test-chip (Fig.1). The encoding and decoding blocks are implemented as distributed architecture models, and thus are distributed to different SoC components on the presented prototype hardware. The prototype printed circuit board (PCB) consists of the custom-built test-chip running a part of the encoding block of the Codec Avatar workload, and an FPGA communicating with the DNN accelerator chip and the VR headset sensors for pre/post-processing image data. An off-the-shelf system SoC on the prototype VR headset runs the decoding workloads and performs rendering of the virtual avatars on its compute DSP and mobile GPU components.

The custom-built DNN accelerator test-chip is implemented in 7nm technology node. The prototype chip features a NN accelerator consisting of a 1024 Multiply-Accumulate (MAC) array, 2MB embedded SRAM capacity, and a 32bit RISC-V CPU to control the on-chip components and execute the program code. Eye gaze feature extraction of the encoding block is implemented as a Convolutional NN (CNN) architecture and is deployed on the test-chip. The eye gaze extraction NN is designed in a hardware-friendly fashion to fit the model on the custom-built chip's total embedded SRAM capacity. This way, the test-chip runs inference on the NN model without loading data from off-chip memory during runtime to save system-level energy and increase performance as only final results are sent to the system SoC. Compared to a traditional central SoC based processing, this approach also enables a higher level of user data security as all sensor data is consumed immediately without being buffered on the system memory.

This work aims to show the implementation details of the whole design stack to realize a Machine Learning workload on AR/VR devices. The presented work focuses on custom SoC implementation, NN architecture customization and deployment on chip, and finally system-level integration of the chip on the prototype PCB for running inference on the NN model. We further provide layer by layer performance breakdown and silicon power measurements.

978-1-6654-0756-4/22/$31.00 ©2022 IEEE

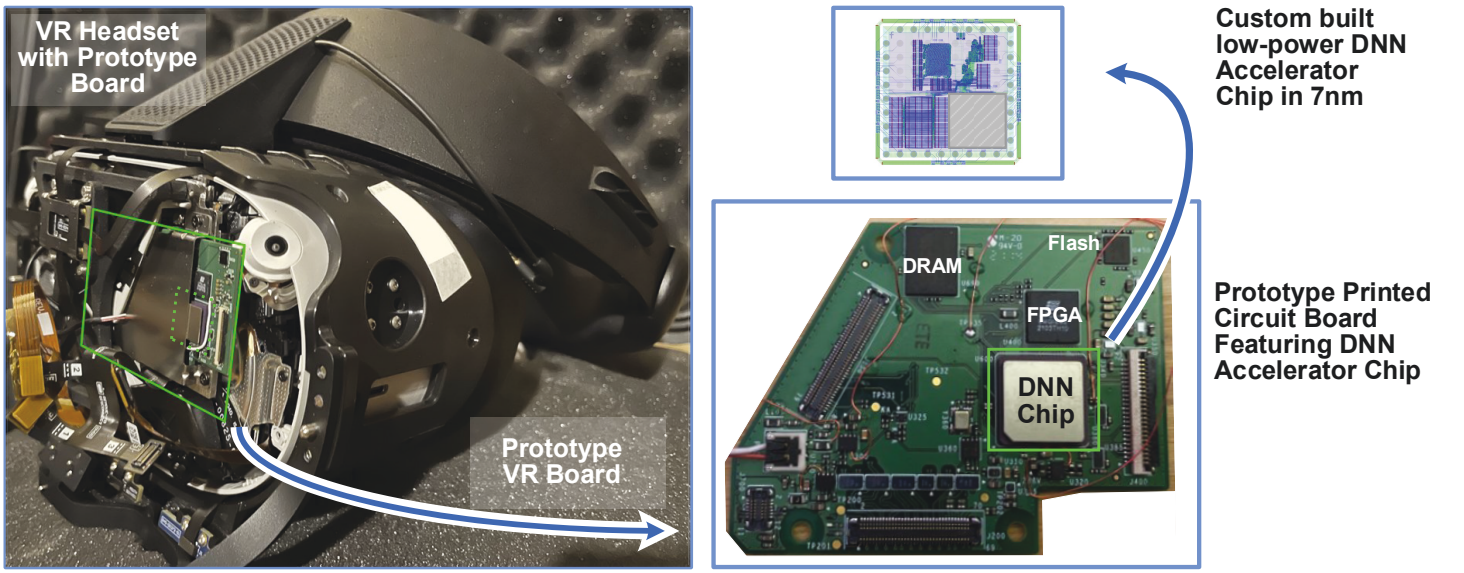


Fig. 2. System-level details of prototype VR hardware: a modified Oculus Quest 2 Mobile VR Headset, prototype printed circuit board (PCB) with its integrated components, and the featured custom built low-power DNN accelerator test-chip.

By combining circuit-level acceleration and hardware-aware CNN model customization, the prototype SoC achieves the system-level 30fps performance target for two eyes per single frame. Our silicon measurements show that the presented test-chip achieves an average low power consumption of 22.7mW (at 500MHz, 0.75V) for processing the entire CNN model, while effectively accelerating convolution operation by trading off high power consumption for very short durations of time for gaining high performance.

The rest of the paper is organized as follows. System-level details of the prototype hardware are provided in Section 2. Section 3 highlights NN model customization details and a custom built compiler to map the NN based workload on the chip. Section 4 provides circuit level details of the DNN test-chip. Finally, the power and performance results of the featured chip are provided in Section 5. The paper is concluded in Section 6.

## 2. Prototype VR Hardware and System-level Integration Details of the Featured DNN Test-Chip

The mission of Codec Avatars is enabling authentic social interaction where natural eye contact plays a very critical role for providing extra interactive information between people. To achieve the best experience for social telepresence in AR/VR, a novel encoder model for sensing joint eye and face appearance was proposed in [4]. In this work, we present a prototype hardware for running the encoding block of the Codec Avatar application, targeting mobile telepresence demonstration on a VR headset. Lab images of the prototype PCB board and a modified Oculus Quest 2 Mobile VR Headset augmented with sensors and lighting placement to demonstrate the Codec Avatars workload are provided in Figure.2.

The prototype PCB consists of the featured DNN accelerator chip, an FPGA, a DRAM chip, and a non-volatile flash type memory chip to store various program and model parameters. The featured custom-built DNN accelerator chip runs inference on the eye gaze extraction CNN model of the Codec Avatars encoding block. The FPGA communicates with the DNN chip for pre/post-processing data. An off-the-shelf, high-performance mobile SoC on the VR headset runs the decoding block workloads.

The FPGA functions as the main system-level interface on the PCB and communicates with the test-chip (on the PCB) and the system SoC residing in the VR headset (off the PCB). The DRAM chip is provided for the FPGA to increase its memory capacity. The non-volatile flash memory is also controlled by the FPGA, and it stores the model parameters and the application programs. The FPGA handles pre-/post-processing the input/output to/from the DNN accelerator chip, and works in parallel to process several encoding block related workloads. The FPGA uses both SPI interface and DNN chip's parallel bus (pbus) interface to communicate with and load data into the accelerator chip. Interface details for the DNN chip are provided in Section.4. Clock and configuration signals (e.g. reset, etc.) for the DNN chip are generated by the FPGA and provided over the GPIO pins.

To run inference on the eye gaze CNN model on the custom chip, model parameters are first loaded into the on-chip SRAM using the pbus interface before program execution. The on-chip configuration registers are also set with the same interface to load the program code binary into the RISC-V core. Since model parameters stay in the on-chip SRAM persistently, no further instructions or additional parameters are loaded during inference runtime. We implemented a custom compiler framework to translate the CNN model to an executable code and to deploy the model on the chip. Detailed information on model customization and deployment are provided in Section.3.

For the presented work, the eye gaze extraction model is trained for right and left eyes separately, meaning that the same CNN architecture has two sets of algorithm-level model parameters for left and right eyes. The application performance target is achieving 30 fps at the system-level. This translates to performing inference on two sensor images of the two eyes within 33.33ms latency at the chip-level. Therefore, the DNN chip integrated on the prototype VR hardware has to achieve less than 16.67ms latency for one eye to run inference on the entire CNN model from a single input sensor image to its final CNN output.

During runtime, the on-chip CPU executes the program of the CNN model per single input image captured from the VR headset sensor(s). VR prototype sensor data is first pre-processed on the FPGA before sending it to the DNN chip, to form a single batch, 16x16 sensor image of 8bit pixels with 64 channels, for one eye. This input activation tensor of dimension (1x16x16x64) and size 16KB is sent to the DNN chip as the input sensor image. Once the input is received, the featured chip performs inference to generate an output eye gaze extraction vector of size 1x1x1x3 output tensor, for one eye (left/right). The CPU moves from one NN layer to the next NN layer by executing the offline compiled hardware operations in order, which readily provide the correct addresses for model parameters (e.g. layer weights, biases, etc.) and all intermediate work data residing in the global SRAM and the CPU cache. After completion, output vector is sent back to the FPGA, where several post-processing and merging steps are performed. The FPGA works on other parts of the encoding block in parallel with the DNN chip, and sends the final output to the main SoC in the VR headset. The chip performs and completes inference on both eyes in a sequential manner, completing two inference runs on two eyes in a single frame by ping-ponging between the CNN model parameters.

The featured chip operates at 500MHz clock frequency and 0.75V VDD. By customizing the CNN architecture for the featured DNN chip, using the in-house compiler framework to deploy and execute the model, and accelerating the convolution operations on the chip, we were able to achieve 16.51ms latency to perform inference on a single eye, thereby meeting the strict system-level 30fps target of the application. Details on how the CNN model is customized and deployed on chip are provided in Section.3, and circuit-level details of the DNN chip are further provided in Section.4.

## 3. Customizing and Mapping Eye Gaze Extraction Model on Chip

The eye gaze extraction model has to provide good accuracy, but at the same time fit well in the on-chip SRAM storage capacity of the featured hardware accelerator to eliminate off-chip memory accesses during the inference runtime, saving latency and system-level energy. For this purpose, a mobile-friendly CNN architecture is implemented to extract the eye gaze of a user from video images with sensors capturing images inside the VR headset. High-level CNN architecture details targeting the featured mobile SoC are provided in Figure.3. Hardware-aware implementation details for the eye gaze CNN model to fit on chip and to facilitate meeting the 30fps performance target are provided in this section.

### 3.1. CNN Architecture and Hardware-Specific Customizations

The eye gaze NN model is comprised of convolutional (CONV) 1x1 and 3x3 layers, and element-wise operations such as Rectified Linear Unit (ReLU) and Pooling. Batch-normalization is implemented to increase model accuracy during training. The custom implemented eye gaze NN model graph and its layer dimensions are provided in Figure.3. The DNN chip is implemented with an integer arithmetic based NN acceleration block. To fit the model parameters on chip in a persistent fashion to avoid off-chip memory accesses during inference runtime, as well as to meet the strict application-level latency requirements using the on-chip acceleration circuit, the NN model is trained for the hardware using quantization aware training (QAT) with signed 8bit integer input activations and weights.

We implemented batch-normalization folding during inference [5] to minimize the amount of required hardware operations. In the folded batch-normalization, multiplicative and additive normalization

| # | Layer | Activation | Filter | Bias | Stride | Output |
|---|---|---|---|---|---|---|
| 0 | conv3x3 | 1, 16, 16, 64 | 128, 3, 3, 64 | 128 | 2 | 1, 8, 8, 128 |
| 1 | conv1x1 | 1, 8, 8, 128 | 256, 1, 1, 128 | 256 | 1 | 1, 8, 8, 256 |
| 2 | conv3x3 | 1, 8, 8, 256 | 128, 3, 3, 256 | 128 | 2 | 1, 4, 4, 128 |
| 3 | conv1x1 | 1, 4, 4, 128 | 256, 1, 1, 128 | 256 | 1 | 1, 4, 4, 256 |
| 4 | conv3x3 | 1, 4, 4, 256 | 32, 3, 3, 256 | 32 | 2 | 1, 2, 2, 32 |
| 5 | conv1x1 | 1, 2, 2, 32 | 64, 1, 1, 32 | 64 | 1 | 1, 2, 2, 64 |
| *p* | pool Avg | 1, 2, 2, 64 | | | 2 | 1, 1, 1, 64 |
| 6 | conv1x1 | 1, 1, 1, 64 | 3, 1, 1, 64 | 3 | 1 | 1, 1, 1, 3 |

Input activation format: [batch, Row, Col, in_channel]
Filter format: [out_channel, Row, Col, in_channel]

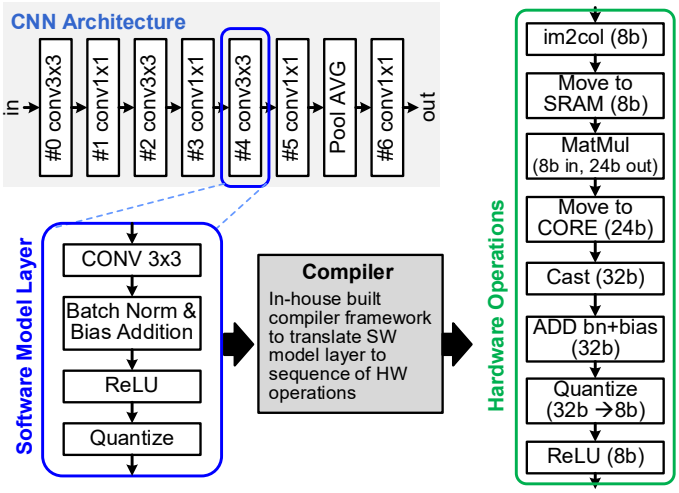


Fig. 3. Eye gaze extraction CNN model dimensions and model graph, with the in-house built compiler framework translating software model layers to executable hardware operations.

parameters are fused with the preceding convolution weights and biases, respectively. This way, batch-normalization is inherently performed at inference during convolutional operations of matrix-multiplication and bias addition. The batch-normalization folding step is performed offline to enable efficient inference. As a result, the NN model only requires trained weights, biases, and quantization parameters stored on the chip for running inference.

To achieve an adequate model accuracy, arithmetic operations following the MAC operation on 8b inputs and weights require higher bit precisions. For convolution operation, multiplication of 8bit inputs and weights generate 16bit results, which are then accumulated over 24bits. However, the 24bit output from MAC operation needs to be re-quantized to an 8bit integer so that it can be used as the input for next layer computation. This re-quantization step requires normalization and rescaling operation with a scale factor generated offline. To meet the strict frame-rate latency considerations, we implemented the integer-based quantization method from [5] to deploy on the accelerator chip. In this method, the integer replacement of the single precision floating point (float32) scaling factor $S_{float32}$ is generated off-line using the following equations:

$$\frac{(Scale_{input} \times Scale_{weight})}{Scale_{output}} = S_{float32} = (2^{-N_{int8}}) \times S_{0_{int32}} \quad (1)$$

$$N_{int8} = round(-\log_2(2 \times S_{float32})) \quad (2)$$

$$S_{0_{int32}} = round(2^{(31+N_{int8})} \times S_{float32}) \quad (3)$$

where $N_{int8}$ in (2) and $S_{0\_int32}$ in (3) are 8bit and 32bit integers, respectively. Using these equations, the quantization layer can be implemented on hardware with integer arithmetic based on multiplication and bitwise shift operations, with scaling factors calculated offline. $N_{int8}$ is used for the shift operation and $S_{0\_int32}$ is used for the multiply operation during quantization. To match the model level accuracy on hardware, the 24bit signed integer (INT24) accumulation outputs of the systolic-array are cast to 32bit signed integers in the RISC-V core before the quantization operations.

We performed behavioral simulations to quantify the cost of carrying out all the normalization operations in the float32 domain by casting the INT24 to float32 on the chip. Our system-level RTL simulation results showed that casting back-and-forth between integer and floating-point domains in the RISC-V core is extremely costly in terms of chip-level performance. In a case-study, our simulations showed that float32 based quantization method in the core takes ~0.9µs per element, at 500MHz clock frequency. Applying the above integer-based quantization method, we were able to achieve >30× faster quantization per element. Therefore, we implemented the integer-based quantization method to eliminate incurring high latency penalty of casting back-and-forth integer and floating-point domains for every NN layer.

Initially, ResNet network architecture [6] was considered to achieve a better model accuracy. However, residual connections introduce additional complexity in hardware implementation due to rescaling of regular path and shortcut path outputs for quantization purposes. Therefore, shortcut connections are dropped for a more hardware friendly implementation of quantization steps while trading off some model level accuracy.

We used QAT with 8bit precision which can perform on par with float32 models [5][7] for state-of-the-art DNN architectures. From analyzing the training loss logs, the model accuracy is not impacted by quantization. In fact, model simplification for designing a hardware friendly architecture actually improved the training loss slightly compared to the full-scale baseline model in this work (loss term 1.10 for simplified model and 1.26 for baseline model). This can be attributed to low resolution input used to train the simplified model and the optimized training flow. We further tested the trained encoder model by rendering the avatar animation and confirmed that there is no visible degradation in the animated avatar quality when low resolution input, quantization method, and hardware-aware model customizations are all employed together.

As a result of implemented model customizations and integer-based quantization method, the final model parameter size that is needed to be stored on chip persistently adds up to less than 550KB. Eye gaze NN model is trained for each eye separately, and the chip can

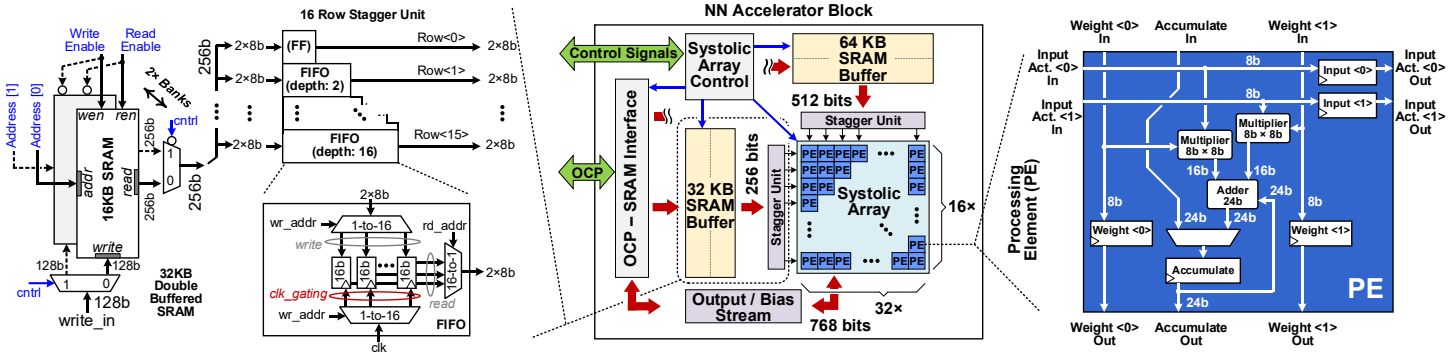


Fig. 4. NN accelerator block details: (Left) Double-buffering and stagger unit circuit details. (Middle) Block-level circuit diagram of the NN accelerator featuring systolic-array based 1K Multiply-Accumulate (MAC) block. (Right) Processing-element circuit details: output-stationary, reduction-of-2 architecture, with two multipliers with signed 8bit integer inputs and a shared 24bit adder for accumulation.

use 2x 1MB partitions to store 2x NN model parameters for the two eyes. Therefore, we allocated approximately half of each 1MB on-chip SRAM partitions to be used as model parameter storage and the remaining halves to be used as a scratchpad memory to store working data (i.e. layer-to-layer activation and intermediate data). Model parameters are stored contiguously in the 2x 1MB SRAM partitions, within the address space of 0 to 550KB for both. To switch back and forth for right and left eyes during runtime, all the hardware operation sequences remain the same while the on-chip CPU core points to two sets of addresses for accessing model parameters. Therefore while one 1MB address space of the memory is active, the other 1MB global partition stays in standby when processing a single sensor image for one eye (i.e. left/right).

### 3.2. Custom Compiler Framework for Mapping the CNN on Chip

We developed a custom built compiler framework to map the NN model onto the accelerator chip (Figure.3). The in-house built compiler first translates NN layers to executable hardware operations in C programming language. The program code in C is then compiled to RISC-V instructions using GNU Compiler Collection (GCC) tool. All of the code compilation is performed offline before runtime. During runtime, the on-chip CPU controls the overall execution of the program by orchestrating the correct sequence of operations and dataflow in between the core, the global SRAM, and the DNN accelerator block.

Each NN layer is first compiled to a sequence of hardware operations specific to our custom chip. The implemented set of hardware operations can be grouped into three main subsets: (i) moving data between the global SRAM, the core, and the DNN accelerator block, (ii) matrix-multiplication performed on the DNN accelerator block, (iii) element-wise operations such as ReLU, Pool, im2col, quantization, casting, etc. performed on the core.

The compiler first parses the NN model and generates a comprehensive list of NN layers and their tensor dimensions. The parsing is performed on model summary data, typically available in PyTorch or Tensorflow models, or by loading an ONNX model. Then each NN layer in the list is translated into a set of hardware operations. For instance a convolutional layer of filter size 3x3 (i.e. CONV3x3) is mapped to equivalent hardware operations as shown in Figure.3. While translating each layer, the compiler calculates correct tensor dimensions and keeps track of SRAM/core address pointers for model parameters and for intermediate layer data (i.e. input activations and MAC outputs). The RISC-V CPU has a total of 128KB cache size. Therefore, to fit both program instructions and working data into the available 128KB core cache, the compiler also keeps track of working data size. Whenever the data size is more than the allocated data cache size, the compiler tiles the hardware operations into manageable data chunks, to be executed within a for-loop. As a result, NN model layers are individually mapped to one or multiple sets of hardware operations in the correct sequence that the custom chip can execute.

We implemented a function library in C language corresponding to the hardware operations that the RISCV core can execute. All the intended hardware operations are then implemented as a corresponding C function that is based on the C library functions. As a result, once the list of hardware operations is ready, the custom compiler translates the operations to their corresponding C functions and automatically prints the C code snippets. These code snippets in the correct sequence with correct address pointers are then automatically stitched together to generate the program code. The final code is passed to GCC to be compiled into RISC-V instructions in binary format. Model parameters are also printed out in binary format to be loaded into the chip. Instruction cache and data cache address space is automatically allocated by GCC during program compilation. As a result, the compiler framework translates the eye gaze CNN model to an executable code for the presented chip.

For a hardware friendly inference, all the weight, input activation, and output tensors are stored in “channel-last” format, meaning that channels are stored in contiguous addresses in hardware. This way, im2col operation for pointwise convolution (i.e. conv1x1) becomes “free”, as the input tensor is inherently stored in the preferred format. To eliminate any matrix reorganization operation on the weights, 4D weight tensor parameters are lowered to 2D matrices offline, and stored in the global on-chip memory in the correct format ready to be loaded into the NN accelerator block without on-chip operations.

After compilation, the program binary file takes 13.5KB of instruction space. As a result 14KB is allocated for instructions in the 128KB core cache. This means that the compiler framework has to fit the working data in the remaining 114KB cache space. With these limits, the compiler automatically fits the layer-to-layer work data within a 78KB address space by tiling large size operations into smaller chunks. The custom-built compiler framework successfully fits the CNN model instructions and all the intermediate data within less than 100KB cache memory space in the RISC-V core.

As a result of the hardware-aware customizations, the eye gaze extraction CNN model fits on the chip for running inference. Utilizing QAT enables on-chip circuit acceleration of convolutional layers with integer-based arithmetic and the custom compiler framework further enables deploying the customized model on the chip.

## 4. Low-Power DNN Accelerator Chip Implementation Details

The presented chip integrated on the VR hardware prototype targets running inference on the CNN model, enabled by hardware-aware customizations and the in-house compiler framework highlighted in the previous section. The chip consists of a 1K MAC array block to accelerate matrix-matrix multiplication based convolution operations, 2MB on-chip SRAM that stores both workload model parameters and acts as a scratchpad memory for holding intermediate data, and a 32bit RISC-V CPU to run the program code and control the communication in between the on-chip components over Direct Memory Access (DMA) interconnect block.

### 4.1. 1024 Multiply-Accumulate Array Accelerator Circuit Details

Array-type convolution acceleration on SoC is demonstrated to enable energy efficiency for mobile platforms [8]. Systolic arrays are

further demonstrated to be efficient hardware accelerators for convolution [9][10]. In convolution, weight tensors consist of *output channel, filter height, filter width, input channel* and input activation tensors consist of *batch, input height, input width, input channel*. These four-dimensional weight and input activation tensors are first lowered to two-dimensional matrices (e.g. via im2col operation), and then multiplied via Generalized Matrix-Matrix Multiplication (GEMM) operation [11]. GEMM operation can be parallelized (or tiled) and mapped to a systolic-array in a resource efficient manner to achieve high throughput and energy-efficient operation. Biases are added to the multiplication output in an element-wise fashion.

Top-level diagram of the NN accelerator block is provided in Figure.4 (middle). To accelerate performing the matrix multiplication based convolution operation, we implemented a systolic-array consisting of 1024 MAC elements. MAC block arithmetic is designed as 8bit signed integer (INT8) inputs for multiplicands that accumulate over a 24bit signed integer (INT24). Each processing element (PE) is implemented as a reduction-of-2 arithmetic block, such that 2× sets of matrix operand inputs are multiplied in parallel in two multiplier circuits and are then accumulated on a shared adder block. This way, the PE executes 2× sets of MAC operations in a single clock cycle (Figure.4(right)). The chosen MAC architecture is output-stationary, such that the accumulation is stored in a 24bit flip-flop per PE until outputs are ready to be collected. Each PE unit in the array is pipelined in systolic fashion with flip-flops storing and passing the inputs and weights to the next PE in the row and the column, while the output flip-flop holding the accumulation result. The PEs are arrayed in a 16x32 configuration in Rows x Columns format, with each PE consisting of 2× MACs accumulating on a single output, therefore totaling to 1024 MAC units. The array configuration of 16x32 was chosen to accommodate for the average layer size of our workloads. The reduction-of-2 architecture was chosen to minimize the overall MAC energy consumption for the specific technology node. To reduce overall dynamic energy, each column of the array is automatically clock-gated when the streamed input is zero during the beginning of each matrix multiplication. For a case study of an example convolutional layer that has 9.4% compute utilization ratio due to a small output channel dimension size of 3 (i.e. *output channel = 3*), the implemented column-wise clock-gating technique reduces the overall dynamic power of the PE array by 36%.

The input matrices to the MAC block are stored in two SRAM buffers of sizes 64KB and 32KB. Both buffers can be chosen as either the input activation buffer or the weight buffer in a mutually exclusive fashion, depending on the convolutional layer parameter sizes. Buffer sizes were chosen to accommodate the average layer size of our workloads. Both buffers are partitioned and implemented by tiling 8KB foundry SRAM macros. SRAM macros are clock gated with read/write enable signals to reduce standby energy. The internal banking mechanism for both memories is implemented with double-buffering architecture as shown in Figure.4(left) to increase the overall NN accelerator throughput. For large data-reuse workloads, double-buffering increases the accelerator throughput by up to 2x. This way, while one memory bank feeds the MAC array with data, the other bank receives the next data tile to parallelize data fetching and minimize overall latency. The 64KB and the 32KB buffers feed the 32× PE columns and the 16× PE rows with 2× elements of 8bits, respectively. As a result, the 64KB and 32KB buffers provide 512bits (i.e. 32x2x8bits) and 256bits (i.e. 16x2x8bits) of data per clock cycle, respectively, to feed the PEs when the systolic-array is fully-utilized. This internal bandwidth is sustained by accessing 2x and 4x 8KB SRAM sub-partitions per bank with 128bit words each for the 32KB and the 64KB buffers, respectively.

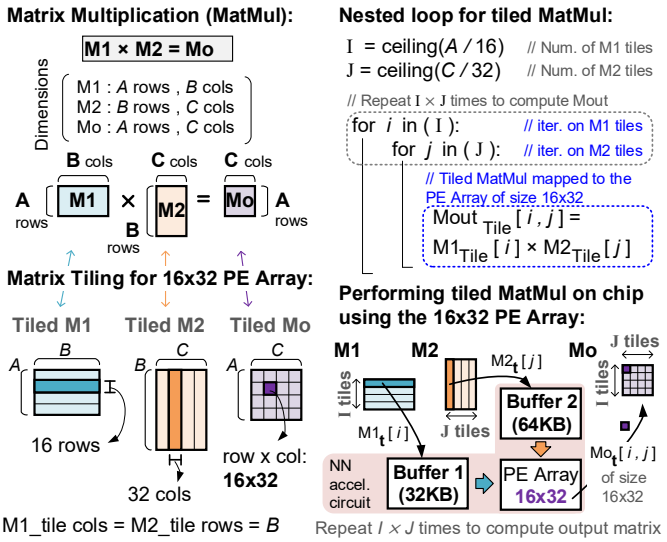


Fig. 5. Matrix multiplication tiling: input and output matrices are tiled into sub-blocks to fit on the PE array circuit. Each output tile of size 16x32 (Row x Cols) is computed one at a time by feeding input matrix tiles to the PE array in a nested loop fashion.

The output-stationary systolic array requires matrix data to be streamed into the array in a staggered manner. We implemented multiple FIFOs (implemented with flip flops) to act as shift registers between the SRAM buffers and the MAC array to provide a fixed delay for each row/column of the array. The staggering unit and a 16-deep FIFO feeding data into the 16th row is shown in Figure.4(left), denoted by *row<15>*. Each FIFO entry is 16bits to hold 2× elements of 8bits. The staggering unit avoids re-organizing the matrices in a diagonal, staggered fashion with initial zeros in the SRAM buffers, which is an energy expensive operation compared to using flip-flops for the chosen size of FIFOs. Flops in the FIFOs are clock-gated such that only the flop that is written a new entry receives a clock pulse to minimize the energy consumption of the staggering unit. FIFO read and write address generation is simplified by using a shared local counter of increment by 1, as the systolic array only requires sequential accesses for the rows and columns.

Majority of the matrix sizes the workload incorporates are larger than the physical 16x32 PE array size. Therefore, larger size matrix operands are divided into smaller matrix tiles to perform the matrix multiplication in a tiled fashion within a loop, as shown in Figure.5. Consider the matrix multiplication (i.e. MatMul) example given in Figure.5, *Moutput = M1 × M2*, with M1 and M2 having *(A x B)* and *(B x C)* matrix dimensions in *Row x Column* format, respectively. Consider also that M1 and M2 are stored in the 32KB and 64KB buffers, respectively. Since the on-chip PE array is implemented as a 16x32 size, M1 and M2 are segmented into multiple tiles of sizes *(16 rows x B cols)* and *(B rows x 32 cols)*, respectively. The MAC block then goes through the M1 and M2 tiles in a nested loop fashion and computes the 16x32 output tiles one by one in an iterative way. Note that if either M1 or M2 tiles have less than 16 rows or 32 columns, respectively, then the 16x32 PE array is not fully filled which creates compute under-utilization. The under-utilization typically occurs at the last M1 or M2 tile, and therefore the number of tiles to iterate in the nested loop are calculated by a ceiling function (i.e. round up). Compute under-utilization also occurs at later CNN layers in the architecture where the convolution layer dimensions are smaller. For M1 and M2 tiles, double buffering mechanism continuously writes the next tile(s) into the local SRAMs to continue feeding the PE array with operands.

A state-machine is implemented to control the data flow of the NN accelerator block; (i) receiving data from global SRAM and storing it in the double buffered input/weight SRAM buffers, (ii) feeding data into the PE array from the local buffers through the staggering units, (iii) start/stop of the systolic array, and (iv) output collection. Matrix tiling and number of tiles are decided offline before runtime. During runtime, the control unit keeps track of operand sizes to orchestrate start/stop for the systolic array per matrix tile and the address generation for the input/weight buffers. The set of two consecutive elements going into reduction-of-2 are grouped together and stored contiguously to be fed into the PE arrays. The matrix re-organization is performed at runtime at the RISC-V core, during im2col operation.

Once the multiplication on an output tile is completed, the outputs are physically collected from the opposite side of the 64KB buffer, such that 32x24bit elements are written to the global SRAM as 128bit data in 6 clock cycles. Output flops are shifted downwards in a daisy-chain structure, and the output collection is repeated for the 16x rows to collect the 16x32x24bit elements. In similar fashion, biases can be

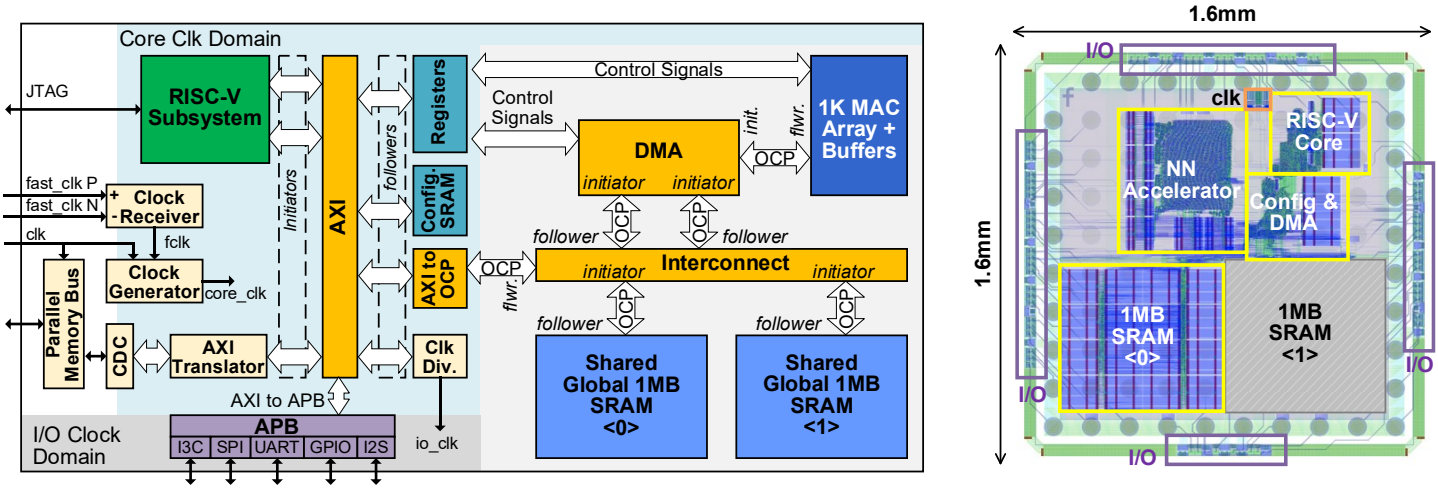


Fig. 6. Custom-built low-power DNN accelerator test-chip details: System-level diagram and chip layout.

first loaded into the output flops to start the accumulation from an initial added value.

The NN accelerator block occupies 0.24 mm$^2$ silicon area on chip. The 64KB and 32KB SRAM buffers are physically placed on the left and bottom sides of the 1K MAC array, respectively, as close to the PEs as possible to minimize the wire energy cost. We measured the power consumption of the test-chip to estimate the energy efficiency of the standalone NN accelerator block. Details on how the power measurements are carried out and more comprehensive power and performance results at the chip-level are provided in Section.5. As a test-case, multiplication of matrices with sizes 128x576 and 576x64 is performed at 500MHz clock frequency and 0.75V VDD. The test-case dimensions are chosen to exercise realistic output tiling and MAC array utilization in the measurements. Cost of writing data into the buffers, systolic array compute operation, and final output collection are all included in the measurement. Global SRAM data read/write energy cost is not included to get the standalone NN accelerator block energy. Our test-chip measurements indicate that the standalone accelerator circuit consumes 29.43mW average power at 500MHz, 0.75V for this test. An additional post-layout RC extracted netlist simulation is performed (500MHz, 0.75V, TT, 25°C) to estimate the power breakdown and the cycle count at the circuit-level. Our analysis shows that the PE array, the local SRAM buffers (input and weight), and the staggering unit consumes 81%, 16%, and 3% of the NN accelerator circuit power, respectively. The standalone NN accelerator achieves an estimated 0.20 pJ/op of energy-efficiency at 500MHz (where a single MAC is two OP's). When including all the circuit components of 1K MACs, 96KB buffer SRAM, and dataflow control, the standalone NN accelerator circuit achieves 4.98 TOPs/W (Tera Operations/sec/Watt) of performance. Different matrix sizes, cost of matrix tiling, and under-utilization due to non-ideal matrix dimensions would change the estimated energy-efficiency accordingly.

### 4.2. Top-level Chip Implementation Details

Top-level block diagram showing the chip organization and the chip layout are provided in Figure.6. The DNN accelerator chip features a RISC-V core to orchestrate memory accesses and data transfers between the systolic array and the global SRAM memory. We implemented a small set of instructions, where each instruction either directs a direct memory access (DMA) controller to transfer data from the 2x 1MB memory partitions to/from the systolic array or instructs the systolic array to compute matrix multiplication. Any remaining control or element-wise operations are performed in the core.

The 32bit RISC-V CPU core is implemented with a generator and includes both integer and floating point arithmetic capability. We implemented a 128KB SRAM based cache in the core, where the address space is allocated for and shared by both program instructions and the data. Address space allocation for instruction and data portions of the cache is predetermined offline for a given program. The RISC-V core includes an integer arithmetic block for addition/subtraction, multiplication, and bitwise operations. It further includes a floating-point unit for float32 based arithmetic operations. Division operation is possible by utilizing either the integer or the floating-point arithmetic blocks for the chosen precision domain, however it is very costly in latency and results in accuracy loss.

The main communication interface chosen for the global SRAM is the Open Core Protocol (OCP). OCP interface allows for customizing the interconnect signals, and therefore was chosen to minimize the amount of control signals and maximize the data bandwidth. Since the majority of DNN applications work on contiguous data (e.g. matrix operations and element-wise loops), we implemented a small set of instructions using the customized OCP based data communication to include burst type of memory read and write accesses. The global SRAM is implemented by arraying multiple partitions of 64KB SRAM macros. We implemented the global SRAM word size to be 128bits. To communicate over the OCP based interconnect, a custom OCP wrapper is implemented around the arrayed memory macros. Global SRAM macros are clock gated with enable signal (read/write) to reduce energy during inactive cycles.

The DMA controller is used to load data from the global SRAM to the NN accelerator buffers and from the systolic array back to the shared global memories. The DMA controller includes 32bit registers to control the data access pattern (e.g., starting address, stride, ending address) for both the read and write ports. These registers can be written to by the RISC-V core through the main AXI interconnect. The data paths of the DMA controller, however, operate over an OCP interconnect which exclusively includes the two shared global memories, NN accelerator buffers, systolic array, and the DMA controller. All data ports in this interconnect are 128 bits wide.

To communicate with off-chip sensors, the RISC-V core uses the APB bus for low speed interconnects (e.g., SPI, I3C, I2S, UART, GPIO). To provide higher bandwidth access to on-chip memories via off-chip (e.g., from the FPGA), the chip includes a 16-bit parallel memory bus (pbus) running at 50MHz, which acts as an initiator on the main AXI interconnect. A dedicated 128KB SRAM is implemented as a configuration buffer to facilitate temporarily storing input data coming from off-chip, connected to the AXI bus. The chip includes several unswitched power domains, operating at the same voltage, with power supplied externally in order to be able to measure on-chip current for various parts of the chip. The main chip clock is supplied externally (e.g., by the FPGA) as an LVDS clock and transformed into a single-ended clock on-chip (i.e. core_clk domain). The 7nm die occupies 2.56 mm$^2$ silicon area.

## 5. DNN Chip Silicon Results for Performance and Power

This section provides detailed performance and power analysis of the DNN chip. Test-chips are verified successfully in a testing lab after fabrication. Power is measured on silicon by running the entire eye gaze CNN model on the chip, from input to output, for a single

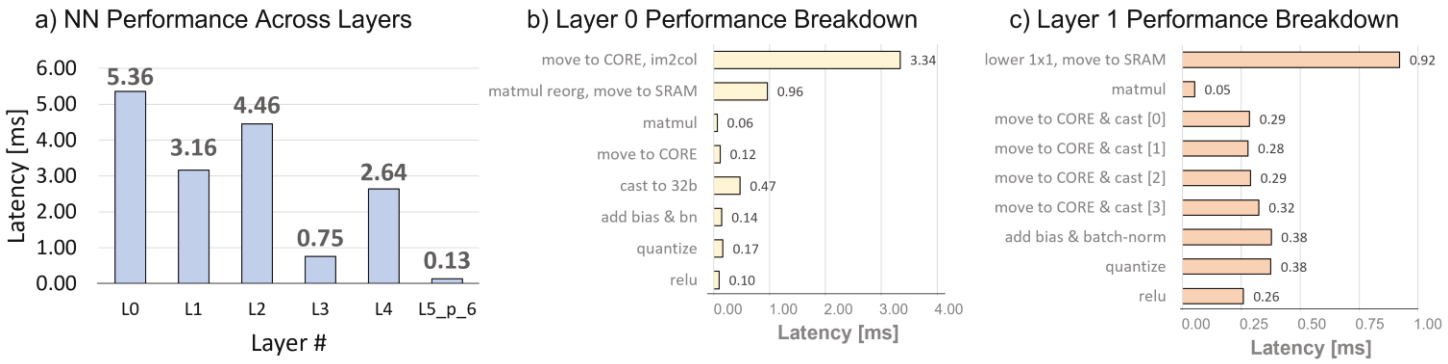


Fig. 7. (a) Layer-by-layer performance results. (b) Latency breakdown for conv3x3 layer L0. (c) Latency breakdown for conv1x1 layer L1.

sensor input. Performance is extracted by using an in-house built profiler on chip-level post-layout RTL simulations.

Boot up sequences for the on-chip core and initial sanity checks were first performed to verify correct test-chip functionality and its on-chip components. These initial tests include global SRAM read/write, example matrix multiplication, and passing messages over the DMA. After the functional verification, we deployed the workload model on the chip. The DNN accelerator chip runs inference on the pre-processed sensor data at 500MHz, 0.75V, processing the entirety of the eye gaze CNN model from input to output while finishing within the target latency per single eye. Chip results after processing the CNN model are read-out and verified with respect to the software implementation of the model to match one-to-one. One time data loading of model parameters and program instructions before the runtime are omitted from the power measurements.

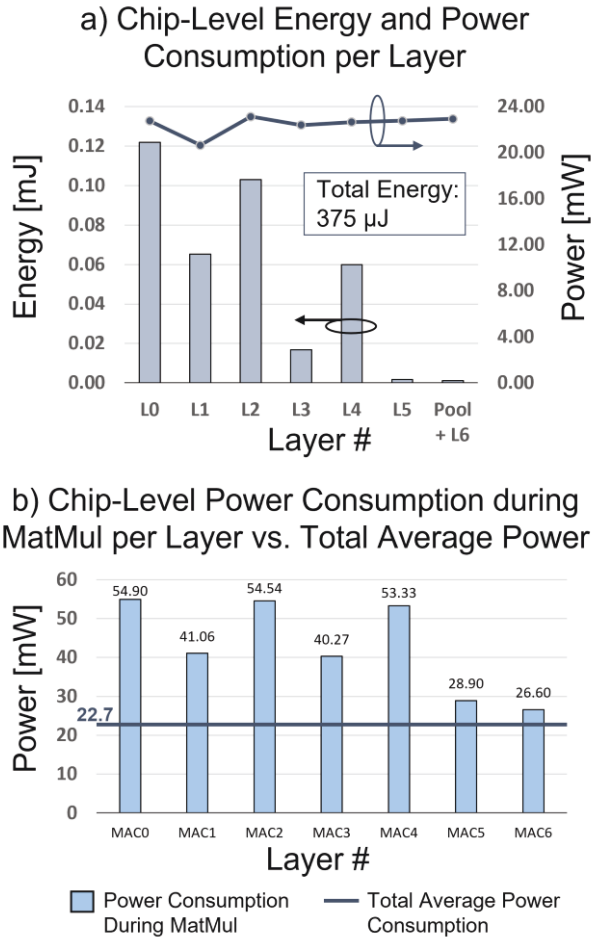


Fig. 8. (a) Chip-level silicon power measurements and energy consumption for all layers. (b) Chip-level power consumption during matrix-multiply acceleration per layer compared to chip-level total average power consumption.

Performance Evaluation

We built a profiler tool to extract the execution time of various NN layers deployed on the chip. The profiler tool reads a decompiled version of the NN code and maps memory address to the entrance and exit of all functions in the program. The tool then scans a simulation dump file (e.g. in .vcd, .vpd, or .fsdb format) and marks the simulation time when the core program counter (e.g. pc) fetches instructions from memory at the entrance/return to/from a function. The time stamps for each function call and return are then processed to calculate how much time the process spends executing each function in the program.

Using the in-house profiler, we extracted the performance of running the eye gaze CNN model on post-layout gate-netlist. The post-layout netlist is used to make sure the results are clock cycle accurate with respect to the test-chip. Performance results for running the eye gaze CNN per single sensor image are shown in Figure.7. The CNN layer dimensions were previously provided in Figure.3. The model consists of seven main layers with various channel and stride characteristics. Layers are grouped into seven for ease of reading the results, where each main layer changes the dimension from input to output. The overall model translates to 51 hardware operations stitched in order, as discussed in Section.3.

The chip runs the model from its 16KB input to the final three element output vector in 16.51ms per sensor image, meeting the workload frame-rate target. Layer-by-layer model execution times are provided in Figure.7.(a). The performance results show that the chip spends most of the execution time in the first three convolutional layers, where L0, L1, and L2 takes approximately 32%, 19%, and 27% of the overall latency, respectively. The remaining layers are executed relatively faster, with L3, L4, L5, pool, and L6 layers taking a combined 21% of the overall execution time. This expected behavior is due to CNN layer dimensions; while the sequence of hardware operations remains the same for the CONV3x3 and CONV1x1 layers, subsequent CNN layers get smaller dimension-wise.

We further analyzed the latency of sub-layer operations. Hardware operations for the layers L0 (CONV3x3) and L1 (CONV1x1) are shown in Figure.7.(b) and Figure.7.(c), respectively. For both layers, our analysis shows that the NN accelerator block completes matrix multiplication operations very quickly, while the core is relatively slower since it needs to handle multiple tasks simultaneously, i.e. program execution, issuing memory accesses over the DMA, and performing all the element-wise arithmetic operations.

For layer 0, our analysis shows that combined initial move to core and performing im2col takes approximately 60% of the layer's execution time, as shown in Figure.7.(b). This cost is because of moving the initial 16KB data into core and working on the im2col operation that prepares the 2D input activation matrix for a 3x3 filter. The systolic-array MAC block accelerates the matrix-multiplication efficiently and completes the overall operation within less than 65µs. For the subsequent L1 layer, the execution time gets faster compared to L0, since there is no im2col cost for 1x1 filter convolution (as discussed in Section.3.2) and the input data size is

reduced to half compared to the previous layer. L1 layer's CONV1x1 operation results in an output matrix size and subsequent casting data size that do not fit into the core's available cache capacity. Therefore the "move to core & cast" operation is performed in four tiles, as shown in Figure.7.(c). Since the output of the L1 layer is larger in size than its input, post-MatMul operations take the majority of the execution time for the L1 layer.

Power Measurements

Test-chip power measurements are performed on standalone chip(s) integrated to a custom testing pcb setup to measure isolated drawn current. Communication to the test-chip to load program code, model data, and input pre-processed data is performed over GPIO using a tester FPGA connected to a lab desktop computer. After program completion, the chip dumps its output over the GPIO back to the FPGA to verify functional correctness in software. For silicon measurements, the model is run from input to output per single eye, and the average drawn current per voltage rail is measured via power supplies. Individual CNN layers are run separately on valid intermediate data for a finer grained silicon measurement. Average power consumption for the entire CNN model or a chosen specific layer is measured by running the associated section of the code repeatedly.

Layer-by-layer silicon measurements for running the model on chip at 500MHz clock frequency, at 0.75V supplied to logic and memory power rails, on a typical corner test-chip are provided in Figure.8. Average power consumption per layer is provided in Figure.8.(a). From the performance analysis, MatMul is effectively accelerated and therefore core related operations occupy the majority of the execution time. This is also evident when we analyze the power consumption. Power consumption of the chip ranges from 20.63mW to 23.13mW for all the layers from L0 to L6, indicating an active core dominated power. At chip-level, the average power consumption for running inference on the entire eye gaze extraction model from input to output is measured to be 22.7mW. This measurement corresponds to processing the entire CNN model in 16.5ms for a single sensor image (i.e. for single eye).

Layer-by-layer and total energy consumption are also provided in Figure.8.(a). The chip consumes 375µJ energy per eye gaze extraction (i.e. per single eye sensor input) over 16.5ms. Similar to the latency analysis, the energy consumption gradually decreases towards the final layers as data dimensions get smaller and the execution gets faster. Our analysis also shows that a given conv3x3 layer consumes more energy when compared to its previous or next conv1x1 layers, due to 2D lowering and re-organization for 3x3 filters generating larger data sizes and thus longer execution times.

Chip-level power consumption during NN acceleration compared to the overall chip-level average power consumption for the entire model execution are provided in Figure.8.(b). Our analysis shows that when the NN accelerator block is actively performing matrix multiplication, the chip-level power consumption goes up to 40mW to 55mW ranges when the compute utilization of the MAC array is high (L0 to L4). NN acceleration power is lower for layers 5 and 6, at the range of 26mW to 29mW as the MAC array utilizes clock gating for the small matrix dimensions in layers L5 and L6.

Combining the power and latency analysis, our results show that the NN accelerator circuit block efficiently trades-off high power consumption coming from its compute resources for speed. The accelerator circuit enables completing the convolution operation consistently within less than 65µs for this workload to achieve overall energy-efficiency at the chip-level. The silicon measurements show that the chip effectively goes into a higher power consumption mode for convolution acceleration for a short burst of time, and remains in a lower-power mode to process the model. As a result, the presented mobile SoC provides efficient high-performance for CNN inference using its acceleration circuit while consuming lower-power on average. Hardware-friendly customizations further enable fitting the model on chip to avoid costly off-chip memory accesses during runtime.

## 6. Conclusion

System-level design and integration considerations for a custom-built, low-power DNN accelerator test-chip is presented in this work. The featured test-chip is integrated on a mobile, prototype AR/VR hardware targeting Codec Avatars demonstration. Eye gaze extraction CNN model of the encoding block is offloaded to the presented test-chip to run inference. The test-chip consumes 22.7mW power at 500MHz clock frequency, at 0.75V, to process the entire CNN architecture from input to output within 16.5ms for a single sensor image, thereby meeting the 30fps target performance for two eyes. In this work, we show that by combining efficient on-chip compute acceleration and hardware-centric NN model rearchitecting, the presented prototype mobile SoC achieves 375 µJ/frame/sensor image energy-efficiency within a 2.56mm$^2$ silicon area.

## References:

[1] Shugao Ma, Tomas Simon, Jason Saragih, Dawei Wang, Yuecheng Li, Fernando De La Torre, Yaser Sheikh, "Pixel Codec Avatars", 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), April 2021, 64-73

[2] Stephen Lombardi, Jason Saragih, Tomas Simon, Yaser Sheikh, "Deep Appearance Models for Face Rendering", ACM Transactions on Graphics (TOG) Vol. 37, Issue 4, 2018, pp 1-13

[3] X. Zhang, D. Wang, P. Chuang, S .Ma, D. Chen, Y. Li, "F-CAD: A Framework to Explore Hardware Accelerators for Codec Avatar Decoding", ACM/IEEE Design Automation Conference 2021 (DAC'21), December 2021

[4] Gabriel Schwartz, Shih-En Wei, Te-Li Wang, Stephen Lombardi, Tomas Simon, Jason Saragih, Yaser Sheikh, "The Eyes Have It: An Integrated Eye And Face Model For Photorealistic Facial Animation", ACM Transactions on Graphics (TOG), Vol. 39, Issue 4, July 2020, pp 91:1 - 91:15

[5] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference", Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pg 2074-2713, Jun 2018

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition", Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun 2016

[7] TensorFlow Model Optimization Team, "Quantization Aware Training with TensorFlow Model Optimization Toolkit - Performance with Accuracy", TensorFlow Blog, April 08 2020, https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html.

[8] Bert Moons, Daniel Bankman, Lita Yang, Boris Murmann, Marian Verhelst, "Binareye: An Always-On Energy-Accuracy-Scalable Binary CNN Processor With All Memory On Chip In 28nm CMOS", IEEE Custom Integr. Circuits Conf. (CICC), Apr. 2018, pp. 1-4.

[9] Y.-H. Chen, T. Krishna, J. Emer, V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE International Conference on Solid-State Circuits (ISSCC), pp. 262-264, February 2016.

[10] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey", Proceedings of the IEEE 105 (2017), 2295-2329

[11] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, Evan Shelhamer, "cuDNN: Efficient Primitives for Deep Learning", ArXiv abs/1410.0759 (2014)