Received: 16 April 2020 | Revised: 15 June 2020 | Accepted: 2 July 2020

DOI: 10.4218/etrij.2020-0134

SPECIAL ISSUE

ETRI Journal WILEY

AB9: A neural processor for inference acceleration

Yong Cheol Peter Cho | **Jaehoon Chung** | **Jeongmin Yang** | **Chun-Gi Lyuh** | **HyunMi Kim** | **Chan Kim** | **Je-seok Ham** | **Minseok Choi** | **Kyoungseon Shin** | **Jinho Han** | **Youngsu Kwon**

AI Processor Research Team, AI SoC Research Department, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea

Correspondence
Yong Cheol Peter Cho, AI Processor Research Team, AI SoC Research Department, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea.
Email: cho@etri.re.kr

Funding information
This research was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2018-0-00195, Artificial Intelligence Processor Research Laboratory).

We present AB9, a neural processor for inference acceleration. AB9 consists of a systolic tensor core (STC) neural network accelerator designed to accelerate artificial intelligence applications by exploiting the data reuse and parallelism characteristics inherent in neural networks while providing fast access to large on-chip memory. Complementing the hardware is an intuitive and user-friendly development environment that includes a simulator and an implementation flow that provides a high degree of programmability with a short development time. Along with a 40-TFLOP STC that includes 32k arithmetic units and over 36 MB of on-chip SRAM, our baseline implementation of AB9 consists of a 1-GHz quad-core setup with other various industry-standard peripheral intellectual properties. The acceleration performance and power efficiency were evaluated using YOLOv2, and the results show that AB9 has superior performance and power efficiency to that of a general-purpose graphics processing unit implementation. AB9 has been taped out in the TSMC 28-nm process with a chip size of 17×23 mm^2. Delivery is expected later this year.

KEYWORDS
AI SoC, inference, neural network accelerator

1 | INTRODUCTION

Sustained technological advances have progressively yielded more powerful and efficient computing capabilities facilitating the recent realization of artificial intelligence (AI) for a wide range of real-world applications. This has sparked substantial interest and research in the field with improved neural networks (NNs) achieving breakthrough results developed in rapid succession.

New networks—each with their own features and benefits—are introduced, yet they all must be executed by a hardware computing platform. It is has become evident that von Neumann-based general-purpose CPUs are inherently ill-suited for the efficient execution of the massively parallel and data hungry computational tasks required in NNs. The parallelization capabilities of general-purpose graphics processing units (GPGPUs) have resulted in their emergence as the default medium for AI computing. However, GPGPUs were originally designed and optimized for graphics processing, which possesses a different computing profile than NNs. GPGPUs are limited by a memory bandwidth bottleneck, where more time is spent moving the large quantity of parameters required for NN computation instead of the computations itself. Furthermore, the use of power-hungry GPGPUs has been shown to be costly in server systems and infeasible for embedded applications

on edges. This has led to the emergence of a new market of semiconductors that are custom designed and optimized for the acceleration of NN computations. Hardware engineers are tasked with making design decisions on trade-offs that balance hardware performance, cost, power efficiency, complexity, and so on with flexibility (a variety of NNs that can be accelerated).

In this paper, we introduce AB9, a quad-core system-on-chip (SoC) that includes a systolic tensor core (STC), an accelerator for NN inference computations. The STC employs a systolic array (SA) populated with sophisticated and highly configurable processing elements capable of almost all computations that take place within a normal neural net layer [multiply-accumulate (MAC) operations, linear activations, rectified linear units (ReLUs), max pool, bias, normalization, and so on], allowing for all computations for a layer to take place within the SA in one iteration. Furthermore, address generation hardware provides synchronized addresses cycle-by-cycle for each data access, removing the need for data manipulation routines such as im2col that are commonly employed to prepare prearranged data to an SA. Complementing AB9 and the STC is a development environment that provides users with an STC compiler (STC-C) and STC simulation engine (STC-SE), both integrated in an intuitive and concise tool flow to expedite application implementation.

The STC accelerator was carefully designed to meet the following goals:

1. Accelerator generality/robustness for a variety of neural nets
2. Concise, intuitive, and effective development environment
3. Accelerator performance

The remainder of this paper is organized as follows. Section 2 describes the hardware architecture of AB9, mostly focusing on the STC accelerator. Section 3 describes the development environment. Section 4 presents an implementation case study using YOLOv2. Section 5 discusses the results obtained from AB9 with an emphasis on the STC accelerator.

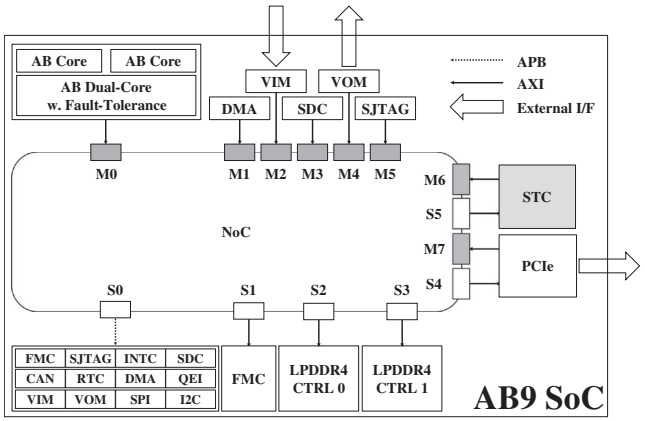

FIGURE 1 AB9 SoC block diagram

2 | AB9 HARDWARE ARCHITECTURE

The AB9 SoC includes four Aldebaran cores [1] that run the 32-bit SPARC instruction set architecture (ISA) at 1 GHz. Two of the four cores in the quad-core setup can be configured to run in the lock-step mode for safety critical applications [2]. The AB9 SoC was designed for both embedded and server environments. The SoC is equipped with two LPDDR4 memory controllers, each with a bandwidth of up to 170 Gbps as well as a PCIe interface. The AB9 SoC is implemented using TSMC 28-nm process technology with a chip area of 17×23 mm^2. Figure 1 shows a block diagram of the major components in AB9. The notable industry-standard peripheral intellectual properties (IPs) included in AB9 are listed in Table 1. The remainder of this section describes the STC accelerator and its components in detail.

2.1 | Systolic tensor core

The compute profiles of NNs indicate that the computations in the convolutional and fully connected layers are the dominant contributors to poor performance [3]. Computations in both the convolutional and fully connected layers can be expressed in matrix form as General Matrix Multiplication (GEMM) operations.

The STC NN accelerator (Figure 2) employs a 128×128 SA of neural processing (NP) elements to perform the GEMM operations required for fully connected and convolutional layers. Computations are performed in 16-bit half-precision floating-point congruent with research that indicates half precision to be sufficient in the range for maintaining negligible accuracy loss during inference [4].

TABLE 1 AB9 IPs

Symbol	Description	Symbol	Description
DMA	Direct memory access	FMC	Flash memory controller
VIM	Video input module	I2C	I2C interface
VOM	Video output module	INTC	Interrupt controller
SPI	Serial peripheral interface	QEI	Quadrature encoder interface
SDC	SD card interface	CAN	CAN bus interface
SJTAG	SJTAG interface	RTC	Real-time clock

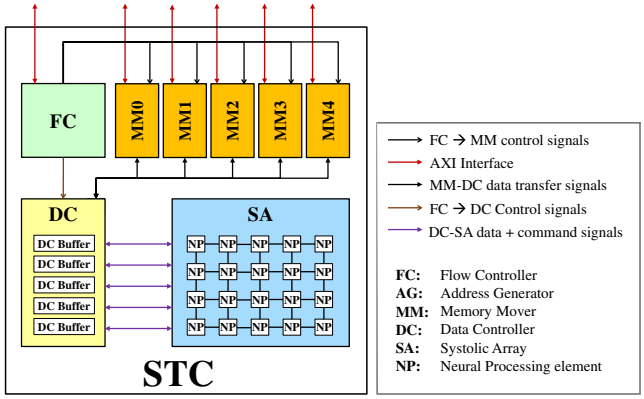

FIGURE 2 STC block diagram [Colour figure can be viewed at wileyonlinelibrary.com]

The systolic approach allows for maximum data reuse and parallelization of computations. Data reuse mitigates the well-documented memory bandwidth performance bottleneck that plagues NN computation. Section 2.5 describes the SA in detail, while Section 2.6 discusses the design of the NPs, which are the processing element nodes that populate the SA.

The resulting data output from the SA and the input operands are stored in over 36 MB of on-chip SRAM within the data controller (DC) module. The DC module is responsible for storing and managing data, delivering commands and operand data to the SA, and satisfying data transaction requests from memory mover (MM) modules. The MM modules provide direct memory access (DMA) functionality, transferring data between memory in the DC module and the external Advanced eXtensible Interface (AXI) bus. Sections 2.3 and 2.4 detail the function and design of the MM and DC modules.

The STC accelerator is centrally controlled by a flow controller (FC) module. The FC instructions dispatched by the host determine its behavior. These include initiating MM data transfer operations, initiating the operand and NP instruction sequence input to the SA, and managing the addresses of where data should preside. Section 2.2 describes the FC in detail.

2.2 | Flow controller

The FC module is the central controller of the STC. The FC module consists of the following submodules shown in Figure 3: FC control (FC_Control), FC command queue (FC_CMD_Q), NP sequence table (FC_NPSEQ), and address generator (FC_AG).

Operation is initiated by the host notifying FC control of the address and quantity of FC instructions to read from DDR4 memory. FC control issues and receives FC instructions via the AXI bus to fill the FC_CMD_Q first-in first-out (FIFO) buffer. The FC instructions are 32-bit long, and new instructions are introduced to the FIFO as older instructions are consumed. FC_ Control decodes the 32-bit instructions that are populated by the instruction types listed in Table 2.

The consecutive-write-command (CWC) instruction is encoded with the number of instructions to transfer (n) and the destination subblock (FC_AG, FC_NPSEQ, MM). Execution of the CWC instruction results in n subsequent entries in the FC_CMD_Q FIFO buffer being sent to the specified subblock. The CWC instruction is used to transfer task parameters to the MM and FC_AG and to populate the NP sequence table (FC_NPSEQ). The NP sequence table is stored in 1024 × 32-bit memory containing 32-bit NP commands.

The NPSEQ instruction transfers the NP commands stored in FC_NPSEQ to the SA via the DC module. Encoded in the instruction are the start address, the end address, and the number of iterations. Entries within the range of the start and end addresses in FC_NPSEQ are sent to the DC for the number of iterations encoded in the instruction. NPSEQ transfers can be interrupted by stalls from events such as DC buffer memory bank conflict, in which case transfers are paused until the stall is resolved.

The PROBE instruction enables the host to check the current status of other modules in the STC accelerator. Status information is used to control the STC execution flow and is conveyed via flags that signify operation start, operation end, and operation in progress.

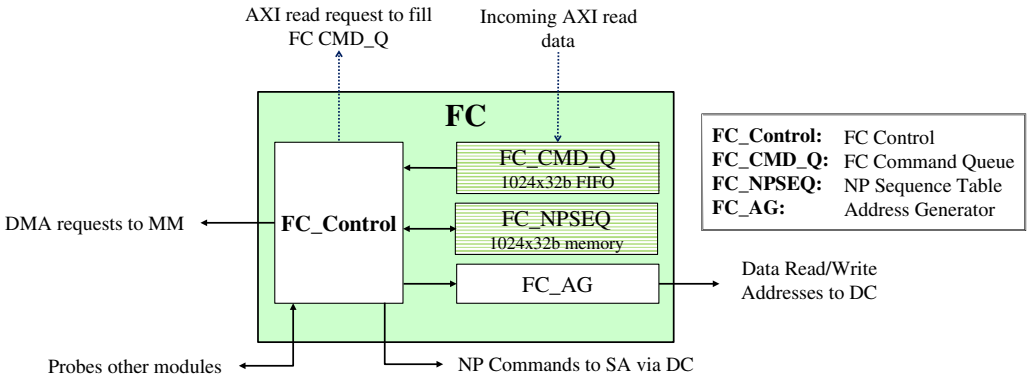

FIGURE 3 STC flow controller block diagram [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 2 FC instructions

Type	Description
CWC (Consecutive-Write-Command)	Transfer multiple units of control data to FC subblock or MM
NPSEQ	Send NP commands to SA
PROBE	Probe status of subblock
WAIT_N	Wait N clock cycles
TERM	Terminate into idle state

2.2.1 | Address generator

The address generator (AG) is a submodule of the FC that generates the sequential DC buffer memory addresses of the input, output, and weight data for the dataflow between the SA and the DC.

Memory address information is sent to the DC synchronized with the issuing of the corresponding NP commands (NPSEQ). Addresses are generated with logic circuitry based on a programmable *n*-dimension nested loop, with parameters to match the input, output, and weight data. For the example convolutional layer in Figure 4, the pseudocode for the seven-dimensional nested loop for input dataflow control is shown in Figure 5. Each loop represents the iterations of the following:

- I loop: weight/input data channel direction
- J loop: weight data width direction
- K loop: weight data height direction
- L loop: input data pooling width direction
- M loop: input data pooling height direction
- N loop: input data sliding window width direction
- O loop: input data sliding window height direction

Among other parameters, the initial values for the number of iterations for the seven loops (`*_LOOP`), seven offset values (`*_INC`), and start address (`*ADDR`) are configured by FC control. In this example, `I_LOOP` and `I_INC` in Figure 5 correspond to "C_i" and "$W_i \times H_i$" in Figure 4, respectively.

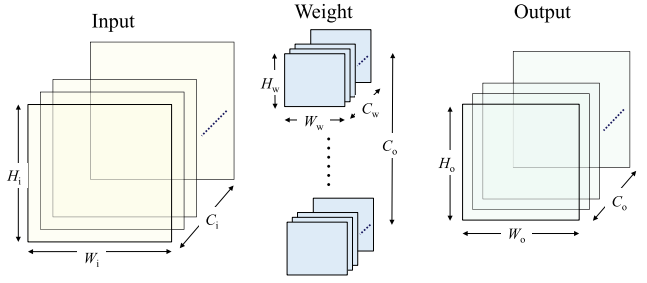

FIGURE 4 Input, weight, output data loop parameters of a convolutional layer [Colour figure can be viewed at wileyonlinelibrary.com]

After the initial address, subsequent addresses are sequentially calculated during run-time.

The programmability of the loop dimensions of the AG design enables its functionality to be applicable to various NNs, while performance is maximized with the optimal loop parameters that are determined during compile time.

Compared to methods that require data manipulation (reordering or redundant data in memory) for "im2col"-like functionality, this AG design conserves memory usage, bandwidth, and manipulation overhead by generating synchronized addresses based on the parameters to retrieve and store data in the DC buffers when necessary.

2.3 | Memory mover

The MM modules are DMA blocks that interface data transfer between the STC internal memory (DC buffer) and the external bus. The STC consists of five MMs. In the standalone mode, MM0 and MM1 are used to transfer the input image data and kernel data into DC buffers, while MM3 is used to output the resulting data from the STC to the external interfaces. In the server mode, MM4 is used as an AXI slave port for the PCIe interface.

2.4 | Data controller

The DC module has the following three main functions:

1. Manage input operand, weight values, and output data in the DC buffers.

```
OADDR = SADDR;
for(O = 0; O < O_LOOP; O++)
    NADDR = OADDR;
    OADDR += O_INC;
    for(N = 0; N < N_LOOP; N++)
        MADDR = NADDR;
        NADDR += N_INC;
        for(M = 0; M < M_LOOP; M++)
            LADDR = MADDR;
            MADDR += M_INC;
            for(L = 0; L < L_LOOP; L++)
                KADDR = LADDR;
                LADDR += L_INC;
                for(K = 0; K < K_LOOP; K++)
                    JADDR = KADDR;
                    KADDR += K_INC;
                    for(J = 0; J < J_LOOP; J++)
                        IADDR = JADDR;
                        JADDR += J_INC;
                        for(I = 0; I < I_LOOP; I++)
                                ADDRESS = IADDR;
                            IADDR += I_INC;
```

FIGURE 5 Pseudocode of the address generator with the seven-dimensional nested loop. (Pooling and sliding window parameters not shown.)

2. Satisfy MM data requests to read/write to/from the DC buffer.
3. Manage the data flow to and from the SA according to the parameters and control signals from the FC.

Figure 6 shows the DC and SA for the first five rows. Each row of the SA is allotted with a DC buffer of 256 kB divided into eight banks for a total of 32 MB for all 128 rows. Different DC buffer banks can be simultaneously accessed. Three systolic register chains (eight stages, 16 rows per stage) transfer requests and data between the DC buffers and the MM modules. Parameters and control signals from the FC are propagated row-by-row in systolic fashion. Information received from the FC includes control and configuration information for the DC, feature map/weight addresses for input to the SA, output data addresses to save data from the SA, NP commands to be issued to the SA, and so on. Data can be rerouted to/from the DC buffers of the neighboring rows when necessary.

The DC module also includes an activation unit (AU) in each row for the computation of nonlinear activation functions using a 2 kB configurable look-up table (LUT). Output data from the SA can either bypass or be fed into the AU.

2.5 | Systolic array

The SA consists of 128 × 128 (NP) elements, each of which is connected to NP elements to propagate feature map/weight data, output data, and carry out NP commands in a systolic fashion. Feature map data and NP commands are propagated from left to right, weight data are propagated from top to bottom, and output data are propagated from right to left. The feature map data are fed into the array via the left-most column, and the weight kernels are fed fed into the NP elements of the first row from the top. The output data exit via the NP elements in the left-most column and are transmitted to the DC.

2.6 | Neural processing element

Most SA architectures in other systems use processing elements that are only capable of MAC computations. The NP in AB9 is more sophisticated and highly configurable, allowing much more than just MAC operations within the processing element of the SA (additional registers for intermediate value storage, maximum value detection, pooling, comparison, ReLUs, and so on). This allows for almost all computations (except nonlinear activations that occur in the AU) that take place in a layer to be executed within the NP without additional iterations.

The NP elements that populate the SA are highly configurable cores that can be programmed to perform a variety of operations. NP behavior is dictated by the NP commands that are fed into the SA from the DC and propagated left to right to a neighboring NP element, as shown in Figure 7. Input feature operands travel through NP elements left to right (*A* operand in Figure 7), while

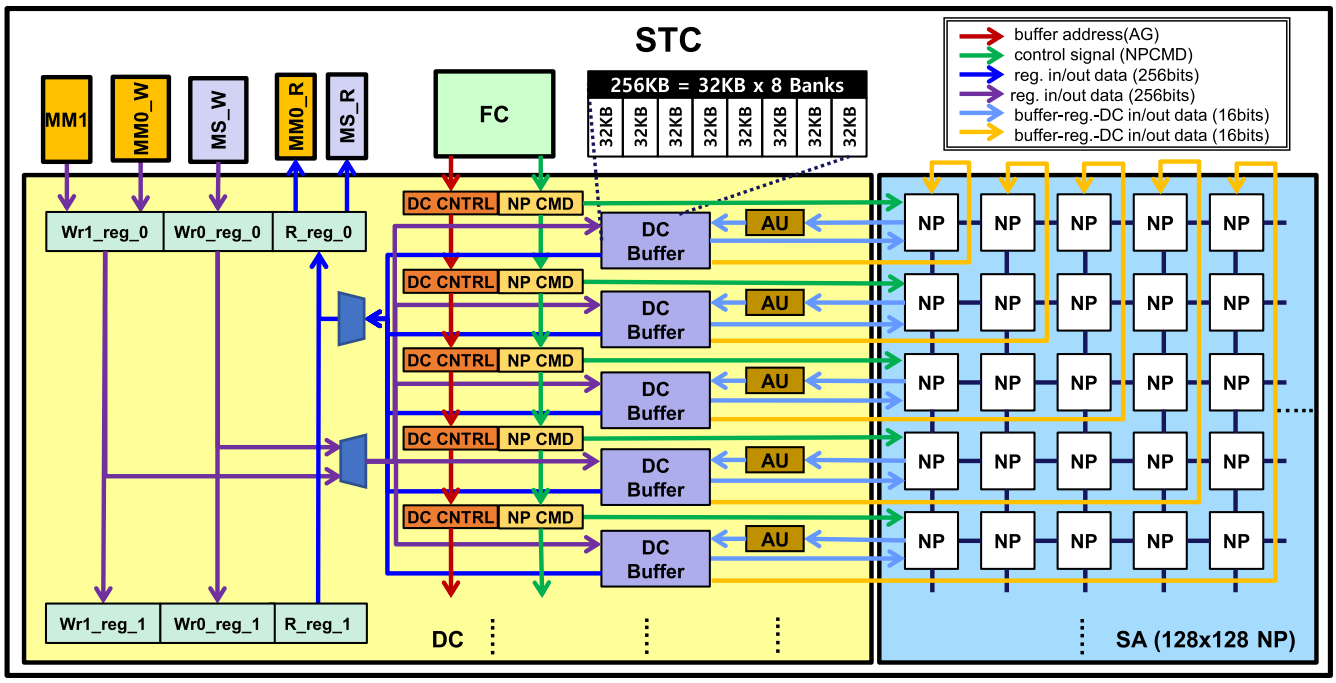

FIGURE 6 STC block diagram showing the DC and NPs for the first five rows [Colour figure can be viewed at wileyonlinelibrary.com]

weight operands travel top to bottom (*B* operands). Output data are relayed to the DC module from right to left (*Z* signal in Figure 7). Operands and the corresponding control signals are synchronized. Each NP includes one half-precision floating-point adder and one multiplier in two pipelined stages. Signals propagate to neighboring NP elements in one cycle. Feedback routes enable the use of data in the *Z* registers as the operand inputs of the multiplier and adder. Structural hazards that may occur when using the feedback routes are taken into account when generating NP commands during compile time. The NP design is highly programmable, where the control information encoded in the NP commands for the nine NP multiplexors enables flexible control of the datapath to execute computations for a variety of NN layer types.

Figure 8 shows how the NP elements can be configured to operate for a MAC operation. Active signals are color-coded by the pipeline stage at which they are active, while inactive signals are faded gray. The control settings encoded in the NP command in *cntrl_r0* determine the active signals shown in red by setting the red multiplexor select signals (*mul_a_sel*, *mul_b_sel*, *P_reg_in_sel*). In this case, input operands A and B from neighboring NP elements (as opposed to data saved in the *Z* registers *Z0_reg*, *Z1_reg*) are fed into the multiplier, and the resulting product is stored in the *P* register (*P_reg*). Similarly, the NP command in *cntrl_r1* configures the green multiplexor select signals to accumulate the product in *P_reg* to the value in *Z0_reg* and store the results back in *Z0_reg*. Output values (*Z_out*) can be configured to be from registers *Z0_reg* and *Z1_reg* or from *Z_in_reg*, which holds the values relayed from the neighboring NP element to the right.

3 | DEVELOPMENT ENVIRONMENT

The application development environment is designed to be intuitive and efficient for nonexpert users. The development environment includes STC-C and STC-SE, as shown in the tool flow presented in Figure 9.

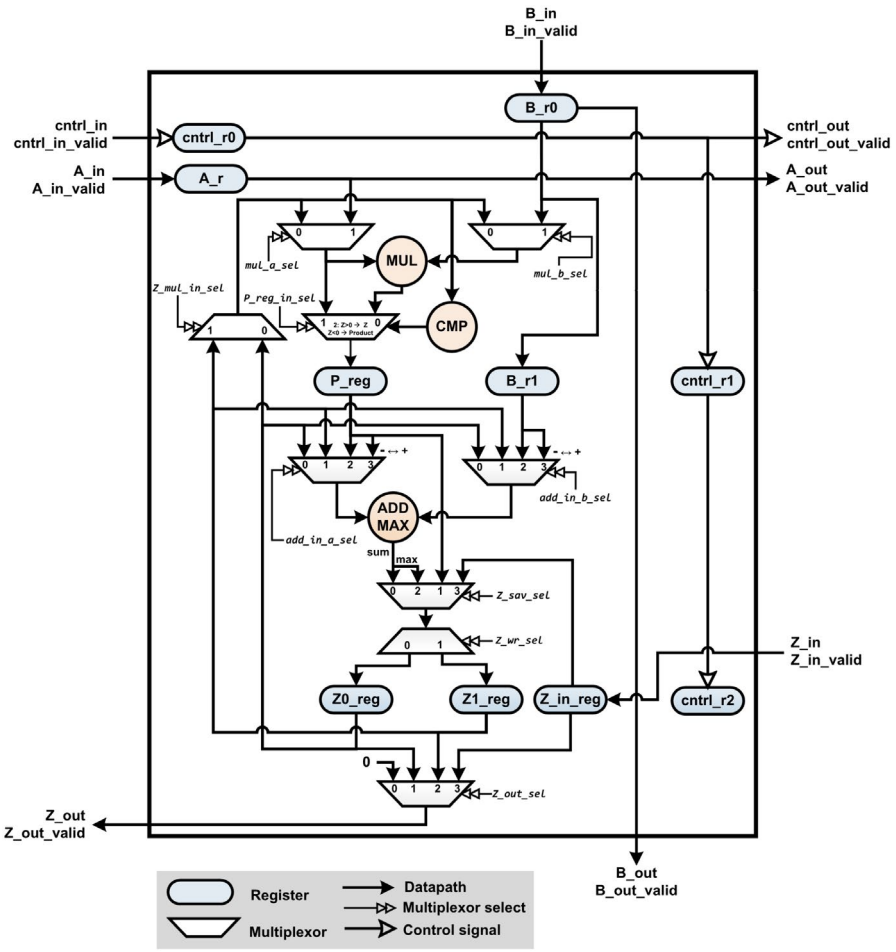

FIGURE 7 NP block diagram [Colour figure can be viewed at wileyonlinelibrary.com]

3.1 | STC compiler

The main function of STC-C is to generate the commands that drive the STC accelerator in order to provide maximum performance for the target application. The three main functional steps of STC-C are shown in color in Figure 10 and consist of (a) parsing configurations (blue), (b) a scheme for the selection of optimal parameters and scheduling (yellow), and (c) command generation (green).

In the first step, STC-C parses three configuration files, as shown in Figure 9: a neural net description file (.stcnn,.prototxt, ONNX, and so on), a file with hardware structure information (.stchw), and a compile option file (.stccc). The second step involves the selection of the optimal scheduling schemes and parameters based on the parsed configuration by comparing the STC performance under configurations with variations in the operational parameters that include tiling, memory allocation, batch size, and hardware operation scheduling.

Tiling of the weight data is based on the number of weights processed in parallel in the SA columns. The input and output data tiling decisions are based on the required and available DC buffer memory for the storage of the input, output, and weight data and the batch size. The memory allocation algorithm assigns the input, output, and weight data to the specific banks in the DC buffer on the basis of the data size and bandwidth parameters. The tiling and memory allocation algorithms ultimately produce parameters that optimize the bandwidth and memory utilization. The hardware operation scheduling algorithm determines the optimal scheme by maximizing the number of parallel computations and concurrently accessible DC buffer banks. The algorithms are optimized on the basis of the three operation parameters but are cohesively streamlined into STC-C because each parameter can influence the other two parameters. Further optimizations such as subsequent layer prefetching during the SA operation of a current layer using a double buffering scheme, multichip execution, and data reuse between layers to minimize memory access are performed by STC-C.

STC command sequences by the layer are generated in the final step on the basis of the optimal parameters determined in the second step, producing three output files: a command sequence file (cmd.h), user-defined file (user.h), and main function file (reference_top.c). The command sequence file includes STC commands by layer in the one-dimensional (1D) array format. The user-defined file lists variables defined by the application, such as the external memory address of data. The main function file guides the main code to call the command sequence file for the application user and will be described in Section 4.

The order and period of data fetching and computations are produced by STC-C during compilation and hence are deterministic. STC-C is therefore able to produce

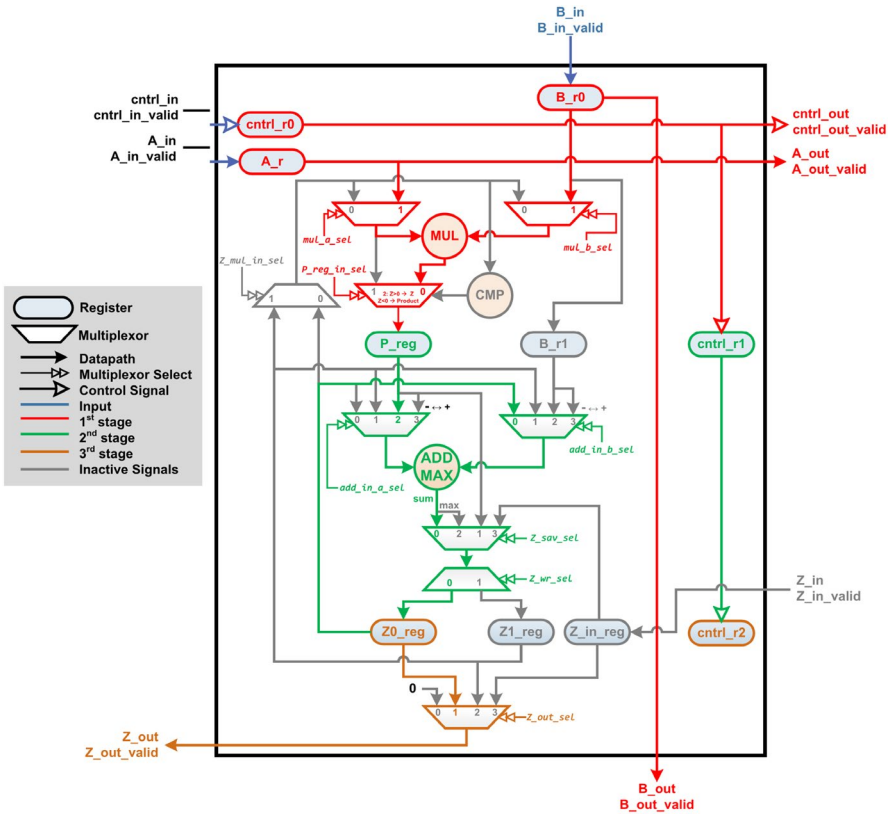

FIGURE 8 NP operation example [Colour figure can be viewed at wileyonlinelibrary.com]

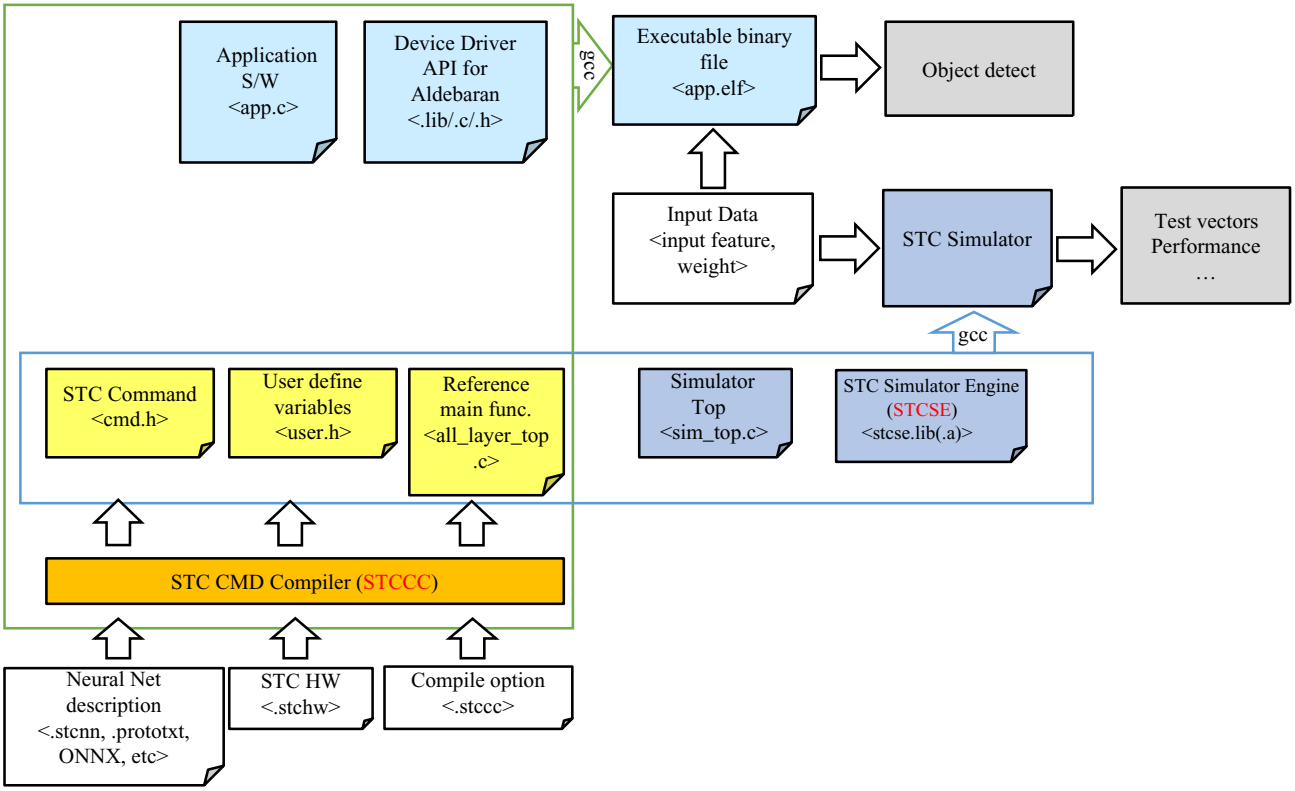

FIGURE 9 AB9 SDK tool flow [Colour figure can be viewed at wileyonlinelibrary.com]

commands that control the power gating circuitry, reducing power consumption by powering off sections of the SA when idle. The time required for power gating toggling to actually take effect is considered (owing to the cell/wire delay of the header cells). Power gating is switched off in advance to eliminate any delay resulting from the power gating scheme. If the SA idle time between two active periods is less than the time for the power gating circuit to

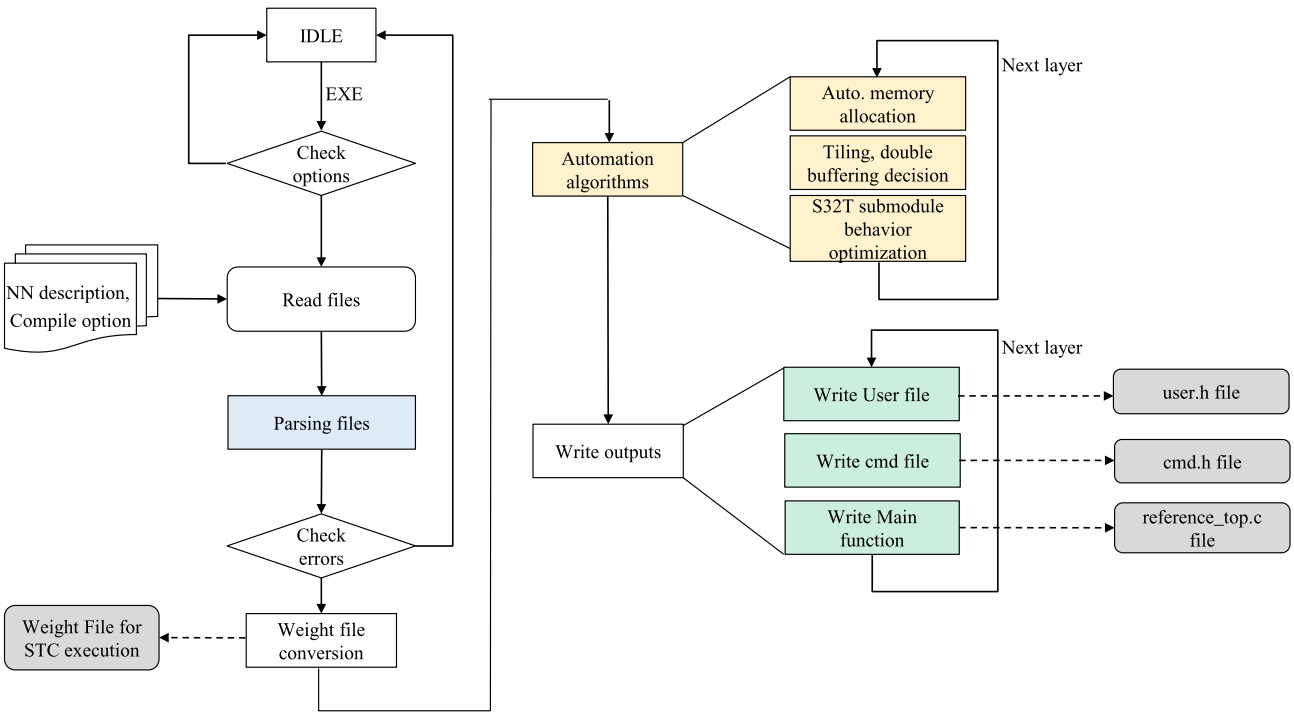

FIGURE 10 STC-C functional steps [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 3 STC-C results for YOLOv2

Layer #	Layer Type	Data Tiling	Weight Tiling	Memory allocation (subblock #)				Input load	Output store
				Weight 0	Weight 1	Input	Output		
0,1	Conv + Pool	2	1	0		2–3	4–7	√	√
2,3	Conv + Pool	2	1	1		4–7	2–3	√	√
4	Conv	2	1	0		2–3	4–7	√	√
5	Conv	2	1	1		4–7	2–3	√	√
6, 7	Conv + Pool	1	1	0		2–5	6–7	√	
8	Conv	1	2	1		6–7	2–5		
9	Conv	1	1	1	0	2–5	6–7		
10, 11	Conv + Pool	1	2	0		6–7	2–5		
12	Conv	1	4	0	1	2	6–7		
13	Conv	1	2	0	1	6–7	2		
14	Conv	1	4	0	1	2	6–7		
15	Conv	1	2	0	1	6–7	2		
16	Conv	1	4	0	1	2	6–7	√	√
17	Pool	1				6–7	2		√
18	Conv	1	8	0	1	2	6–7	√	
19	Conv	1	4	0	1	6–7	2		
20	Conv	1	8	0	1	2	6–7		
21	Conv	1	4	0	1	6–7	2		
22	Conv	1	8	0	1	2	6–7		
23	Conv	1	8	0	1	6–7	2–3		
24	Conv	1	8	0	1	2–3	6–7		√
26	Conv	1	1	0		2–5	7	√	√
29	Conv	1	8	1	0	2–4	6–7	√	
30	Conv	1	4	1	0	6–7	2		√

[a]Color shading shows when input load and output store are skipped because output data of one layer is used as the input of a subsequent layer.

switch on and off, power is maintained to avoid performance loss.

3.2 | STC-SE

STC-SE is used to analyze the performance of the STC accelerator for a specific NN and to verify the command files generated by STC-C. All of the key components are modeled, including the DDR4 memory, Aldebaran core, PCIe interface, and the network-on-chip (NoC).

STC-SE was developed to enable accurate cycle-level simulation for the STC and transaction-level verification for the data exchange between DDR memory and on-chip memory. For verification, the weight and input data are loaded into memory, the STC model is run for each layer, and the generated result is compared to the reference output data.

STC-SE is implemented in the form of a library and can be executed by compiling with command files generated by STC-C and C code for model generation to generate and run an executable file. It was designed to maximize the utilization of the processor resources of the system running STC-SE by running the host central processing unit (CPU), AB9, and STC models simultaneously via multithreading functions.

4 | CASE STUDY: YOLOv2

This section describes the development of an application using YOLOv2 [5] as an example.

4.1 | AB9 bare-metal application implementation: YOLOv2

YOLOv2 for a 416 × 416 image was implemented in a bare-metal environment. The original YOLOv2 program was converted into a bare-metal version. The network configuration

TABLE 4 AB9 SoC specifications

Performance	40 TFLOPS,
Power	15 W–40 W
Die area	17 mm × 23 mm, 391 mm^2
Transistor count	~1 billion
Process technology	TSMC 28 nm
Frequency	1 GHz
Arithmetic units	32 768
On-chip memory	>36 MB
Number representation	16-bit floating-point

TABLE 5 VGG-19 comparison

	fps	Image Size	Type	Source
AB9 STC	74	256 × 256	SoC	
Hisilicon Kirin 990 5G	32.5	256 × 256	SoC	[7,8]
Snapdragon 855 Plus	5.5	256 × 256	SoC	[7,8]
NVIDIA GTX 1080Ti	20.8	224 × 224	GPU	[9]
Intel Dual Xeon E5-2630	0.277	224 × 224	CPU	[9]
NVIDIA Titan RTX	12.2	256 × 256	GPU	[7,8]
NVIDIA Tesla V100	16.4	256 × 256	GPU	[7,8]
NVIDIA Titan X Pascal	9.26	256 × 256	GPU	[7,8]
Edge TPU	3.25	256 × 256	SoC	[10]

files were converted to static linked lists to enable the use of the original parsing code. The operand data were converted to 16-bit half-precision floating-point.

The convolutional and pooling layers are handled by the STC hardware while the routing layers are executed by setting the output of a previous layer as the input to the consuming layer (eg, layers 27 and 24 to layer 28). The reorg layer is executed by software running on the Aldebaran core. Aldebaran core 0 issues the address of the layer commands to the FC and waits until layer processing is carried out before proceeding to the next layer.

4.2 | STC acceleration of YOLOv2

STC-C generates commands to control the MM, FC, AG, and NP elements. The data header file for the commands should be defined in the actual application software using a #define statement. The commands are in stccc_cmd.h. User-configured #define statements must be provided in the stccc_user.h file and included in the stccc_cmd.h file. The weight and layer output buffer locations are set by the user, while the network input buffers locations are determined by the compiler and linker.

The network_run() function invokes forward processing and contains a for loop, where the layer processing function "all_layer_top" is called layer-by-layer. Some software layers are separately called in this loop. The all_layer_top function receives the layer number, which is used to pass a pointer to the command data and the length of the commands for the layer to the STC hardware. Prior to calling the layer processing to the hardware, the software resets a flag indicating layer completion, and the interrupt service routine later sets the flag so that the software can proceed to process the next layer.

4.3 | STC-C decision result for YOLO2

Table 3 presents the tiling, memory allocation, and data reuse results from STC-C for YOLOv2 with a batch size of 8 and a batch height of 16. DC bank numbers (eight banks available per row) are separately allocated for the input, output, and weight data and are listed in the memory allocation column. Weight has two columns because the weight data are prefetched by a double buffering scheme expediting data fetching and reducing the SA idle time. The input/output data for layers 0–5 are tiled owing to their large size. Interlayer data are reused as much as possible to minimize the run-time by reducing the movement of the data in and out of external memory. For the last two columns, the cells are shaded in blue when input data loading is skipped because the output data from the previous layer are reused as input data for the current layer, while the cells are shaded in yellow when output data storing is skipped as the output data are reused as input data in the next layer.

5 | RESULTS

To evaluate the performance of the AB9 SoC, we compare its performance with that of other hardware acceleration platforms found in the literature. Two well-known networks were chosen for measurement. Table 4 lists the specifications of AB9. Table 5 shows the performance when running VGG-19 [6], while Table 6 shows the performance for YOLOv2 [5] with the standard 416 × 416 image size.

TABLE 6 YOLOv2 comparison

	fps	Type	Source
AB9 STC	100	SoC	
Movidius	3	SoC	[11]
NVIDIA Pascal Titan X	67	GPU	[5]
NVIDIA Jetson TX2	7	GPU	[5]
NVIDIA Xavier AGX	30	GPU	[12]
NVIDIA GTX1080	28	GPU	[12]
Xilinx FPGA with DPU	25	FPGA	[12]

TABLE 7 AB9 execution time distribution for YOLOv2

Layer	% Time	SA Active	Filters	Size	Input	Output	Type
0, 1	4.31%	23.48%	32	3 × 3	416 × 416 × 3	208 × 208 × 32	Conv + maxpool
2, 3	4.59%	35.10%	64	3 × 3	208 × 208 × 32	104 × 104 × 64	Conv + maxpool
4	6.70%	26.04%	128	3 × 3	104 × 104 × 64	104 × 104 × 128	Conv
5	3.71%	8.82%	64	1 × 1	104 × 104 × 128	104 × 104 × 64	Conv
6, 7	3.13%	55.11%	128	3 × 3	104 × 104 × 64	52 × 52 × 128	Conv + maxpool
8	4.12%	43.76%	256	3 × 3	52 × 52 × 128	52 × 52 × 256	Conv
9	1.86%	17.65%	128	1 × 1	52 × 52 × 256	52 × 52 × 128	Conv
10, 11	2.69%	61.91%	256	3 × 3	52 × 52 × 128	26 × 26 × 256	Conv + maxpool
12	3.40%	49.74%	512	3 × 3	26 × 26 × 256	26 × 26 × 512	Conv
13	1.80%	35.48%	256	1 × 1	26 × 26 × 512	26 × 26 × 256	Conv
14	3.40%	49.80%	512	3 × 3	26 × 26 × 256	26 × 26 × 512	Conv
15	1.82%	35.10%	256	1 × 1	26 × 26 × 512	26 × 26 × 256	Conv
16	3.39%	49.83%	512	3 × 3	26 × 26 × 256	26 × 26 × 512	Conv
17	1.21%	13.49%			26 × 26 × 512	13 × 13 × 512	Maxpool
18	5.90%	55.66%	1024	3 × 3	13 × 13 × 512	13 × 13 × 1024	Conv
19	1.78%	71.89%	512	1 × 1	13 × 13 × 1024	13 × 13 × 512	Conv
20	5.90%	55.70%	1024	3 × 3	13 × 13 × 512	13 × 13 × 1024	Conv
21	1.78%	71.89%	512	1 × 1	13 × 13 × 1024	13 × 13 × 512	Conv
22	5.89%	55.69%	1024	3 × 3	13 × 13 × 512	13 × 13 × 1024	Conv
23	9.45%	69.01%	1024	3 × 3	13 × 13 × 1024	13 × 13 × 1024	Conv
24	9.44%	69.11%	1024	3 × 3	13 × 13 × 1024	13 × 13 × 1024	Conv
25	-	-				26 × 26 × 512	Route
26	0.87%	18.87%	64	1 × 1	26 × 26 × 512	13 × 13 × 2048	Reorg
27	-	-				13 × 13 × 3072	Route
28	11.23%	72.51%	1024	3 × 3	13 × 13 × 3072	13 × 13 × 1024	Conv
29	1.63%	69.12%	425	1 × 1	13 × 13 × 1024	13 × 13 × 425	Conv

From the results, we observe that the AB9 SoC exhibits a higher fps than the other surveyed hardware. The advantage in performance can be attributed to the hardware optimizations customized for NN computations. This starts from the robust flexibility and functionality of the custom processing elements (NP) to the systolic architecture for data reuse with address generation hardware that eliminates data manipulation overhead, as well as the NN compilation environment that provides the optimal configurations based on NN parameters.

Table 7 lists the distribution of the STC execution time for each YOLOv2 layer as well as the relative time that the SA was active for that layer. We can observe that the SA is utilized more often with batch parallelism during the later layers where feature map size is reduced and the filter quantity is large. The SA is active for about 52% of the total execution time.

A comparison of the power efficiency is rather difficult, as comparable measurements are difficult to find and several variables undermine evaluation on equal grounds. For the sake of argument, public data on GPGPUs allow us to make assumptions and estimates of power efficiency. The GPU power can reach upward of 200 W depending on the workload. The NVIDIA Pascal Titan X GPU is documented to consume 75 W when idle [13]. Using this as an optimistic assumption with YOLOv2 at 67 fps, it consumes 1.117 J/frame, while the AB9 SoC consumes 0.15–0.4 J/frame. The power efficiency of the AB9 SoC can be attributed to the custom design for NN acceleration, as opposed to the generality that the GPGPU maintains for other applications.

Figure 11 shows the layout of the AB9 SoC, while Figure 12 shows pictures of the actual SoC.

6 | CONCLUSION

We have introduced the AB9 neural processor SoC that features an STC NN accelerator for inference. Complementing the hardware is a concise and effective development

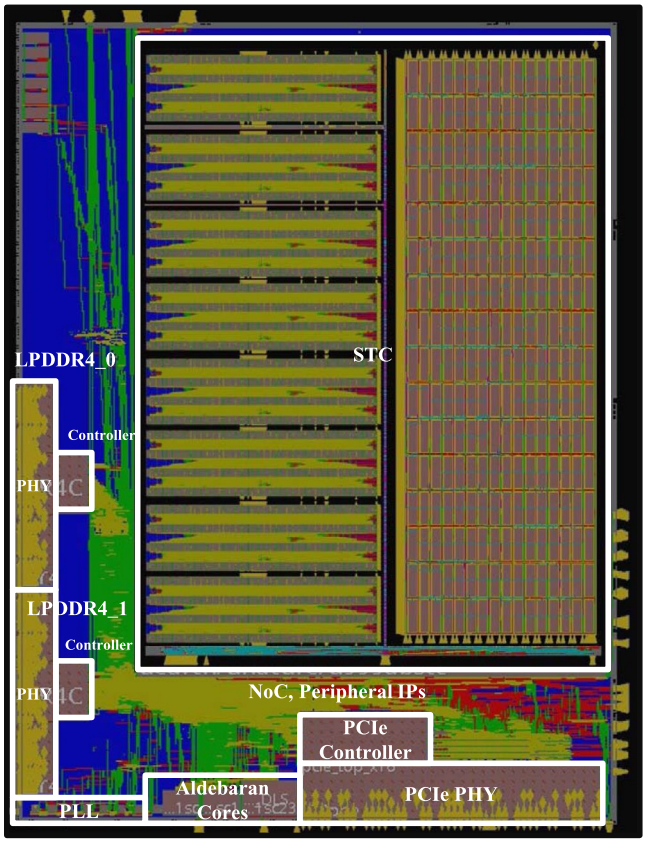

FIGURE 11 AB9 SoC layout [Colour figure can be viewed at wileyonlinelibrary.com]

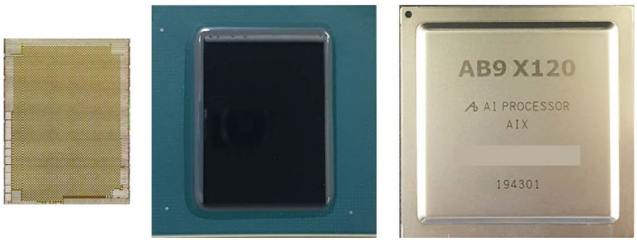

FIGURE 12 AB9 (A) die, (B) die on package, and (C) processor [Colour figure can be viewed at wileyonlinelibrary.com]

environment that includes a compiler and simulator engine. With a custom design targeting NN acceleration, the STC accelerator and its development environment are optimized to enhance performance and efficiency while maintaining programmability to accommodate various NN types. The AB9 SoC uses TSMC 28-nm process technology with a chip area of 17 × 23 mm^2.

ORCID

Yong Cheol Peter Cho https://orcid.org/0000-0002-3947-6984
Hyunmi Kim https://orcid.org/0000-0003-4105-7639
Jinho Han https://orcid.org/0000-0002-0655-320X

REFERENCES

1. ETRI Technology, Aldebaran microcontroller SoC for mobile robot (low power MCU core technology), 2017, available at https://www.etri.re.kr/eng/bbs/view.etri?b_board_id=ENG03&b_idx=16719
2. J. Han et al., *A 1GHz fault tolerant processor with dynamic lock-step and self-recovering cache for ADAS SoC complying with ISO26262 in automotive electronics*, in Proc. IEEE Asian Solid-State Circuits Conf. (Seoul, Rep. of Korea), Nov. 2017, pp. 313–316.
3. Y. Jia, *Learning semantic image representations at a large scale*, Ph.D. Thesis, EECS Department, Univ. of California, Berkeley, May 2014.
4. S. Gupta et al., *Deep learning with limited numerical precision*, Int. Conf. Mach. Learn. **37** (2015), 1737–1746.
5. J. Redmon and A. Farhadi, Yolo9000: Better, faster, stronger, 2016, available at https://arxiv.org/abs/1612.08242, preprint.
6. J. Kim, J. K. Lee, and K. M. Lee, *Accurate image super-resolution using very deep convolutional networks*, in Proc. IEEE Conf. Comput. Vision Pattern Recognit. (Las Vegas, NV, USA), 2016, pp. 1646–1654.
7. A. Ignatov et al., *AI benchmark: All about deep learning on smartphones in 2019*, in Proc. IEEE/CVF Int. Conf. Comput. Vision Workshop (Seoul, Rep. of Korea), Oct. 2019, pp. 3617–3635.
8. AI-Benchmark, available at http://www.ai-benchmark.com
9. J. Johnson. Benchmarks for popular CNN models, available at https://github.com/jcjohnson/cnn-benchmarks
10. Coral, Edge TPU performance benchmarks, available at https://coral.ai/docs/edgetpu/benchmarks/
11. T. Narayan and Intel AI Academy, A comparison of performance of deep learning models on Edge using Intel Movidius Neural Compute Stick and Raspberry PI3, available at https://medium.com/intel-student-ambassadors/object-detection-a-comparison-of-performance-of-deep-learning-models-on-edge-using-intel-f66eb7f45b17
12. S. Hossain and D. Lee, *Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with GPU-based embedded devices*, Sensors **19** (2019), no. 15, 3371:1–3424.
13. J. Guerreiro et al., *Modeling and decoupling the GPU power consumption for cross-domain DVFS*, IEEE Trans. Parallel Distrib. Syst. **30** (2019), no. 11, 2494–2506.

AUTHOR BIOGRAPHIES

Yong Cheol Peter Cho received his BS degree in computer engineering from the Pennsylvania State University in 2005, his MS degree in computer engineering from the University of Southern California in 2009, and his PhD degree in computer engineering from the Pennsylvania State University in 2012. He is currently a senior researcher with the AI SoC Research Department at the Electronics and Telecommunications Research Institute in Korea. His research interests include application-specific hardware accelerators, embedded systems, and processor design.

Jaehoon Chung received his BS and MS degrees in computer engineering from Korea University, Seoul, Rep. of Korea in 2015 and 2017, respectively. He joined the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea in 2017 and is currently a researcher with the AI SoC Research Department. His current research interests include deep learning accelerators and low-power design.

Jeongmin Yang received his BS and MS degrees in electrical engineering from KAIST, Daejeon, Rep. of Korea in 2012 and 2014, respectively. Since 2014, he has been working for the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea, where he is now a senior researcher with the AI SoC Research Department. His main research interests are VLSI design and AI processor architecture.

Chun-Gi Lyuh received his BS degree in computer engineering from Kyungpook National University, Daegu, Rep. of Korea in 1998. He received his MS and PhD degrees in electrical engineering and computer science from the Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea in 2000 and 2004, respectively. He joined the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea in 2004 and is currently a principal researcher with the AI SoC Research Department. His current research interests include mobile deep learning processor hardware and software development environments.

HyunMi Kim received her BS and MS degrees in electronic engineering from Inha University, Incheon, Rep. of Korea, in 2004 and 2006, respectively, and her PhD degree in computer software from the University of Science and Technology, Daejeon, Rep. of Korea, in 2018. Since 2012, she has been with the Electronics and Telecommunications Research Institute and is currently with the AI SoC Research Department as a senior engineer. Her research interests are in neural network systems, which include AI processors and DL compilers, SoC architecture design, optimization algorithms for SoC systems, and signal processing for multimedia applications.

Chan Kim received his BS degree in electrical engineering from POSTECH in 1991 and his ME degree in electrical and electronic engineering from KAIST in 1993. From 1993 to present, he has been working at the Electronics and Telecommunications Research Institute in Daejeon, Rep. of Korea. His main research interests are communication systems such as ATM, Ethernet, 3G-LTE, multigig wireless, 3D-TVs, and processor systems for automotive or deep learning. He used to develop hardware (ASIC/board/driver) for almost 15 years but these past years, he has been working on various system and application software for developed systems (bare metal, Linux, BusyBox, RTEMS, QEMU, etc).

Je-seok Ham received his BS degree in electronics engineering from Kyungpook National University, Daegu, Rep. of Korea, in 2017. He received his MS degree in bio and brain engineering from the Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea in 2019. Since 2019, he has been with the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea, where he is now a researcher. His main research interests are the BLAS library for parallel programming and hardware accelerators for HPC processors.

Minseok Choi received his BS and MS degrees in electrical and electronics engineering from Korea Advanced Institute of Science and Technology, Rep. of Korea in 1997 and 1999 respectively. He joined the Electronics and Telecommunications Research Institute, Rep. of Korea in 1999 and is currently a principal member of the research staff. He has special interests in on-device deep learning processor hardware and software development.

Kyoungseon Shin received his BS and MS degrees in electrical engineering from Chonbuk National University, Jeonju, Korea in 1989 and 1991, respectively. His MS work focused on built-in self-test circuit design for the fast testing of megabit DRAM. From 1991 to 1999, he worked at LG Semiconductor Co., Ltd., Korea. While working for LG Semiconductor, he was involved in designing microcontroller units and the Micom Development System. He joined the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea in 1999 and is currently a

principal member of the research staff. His current research interests include the development of artificial intelligence processors, low-power embedded processors, and autonomous driving processors.

Jinho Han received his BS, MS, and PhD degrees from Korea Advanced Institute of Science and Technology, Rep. of Korea in 1998, 2001, and 2020, respectively. He has been with the AI Processor Research Section at the Electronics and Telecommunications Research Institute (ETRI), Rep. of Korea, since 2001. At ETRI, he is a section leader and principal research staff of the AI Processor Research Section devoted to the design of the AI processor AB. He has special interests in many-core architectures, AI processor design, low-power processor design, fault tolerance design, and algorithmic optimization of circuits and systems.

Youngsu Kwon received his BS, MS, and PhD degrees from Korea Advanced Institute of Science and Technology, Rep. of Korea in 1997, 1999, and 2004, respectively. He had been with Microsystems Technology Laboratory, Massachusetts Institute of Technology as a postdoctoral associate from 2004 to 2005 and designed three-dimensional FPGAs. He has been with the AI SoC Research Department at the Electronics and Telecommunications Research Institute (ETRI), Rep. of Korea, since 2005. At ETRI, he is a director and principal research staff of the AI SoC Research Department devoted to the design of the AI processor AB. He has special interests in many-core architectures, AI processor design, low-power architecture design, computer-aided design, and algorithmic optimization of circuits and systems. He received the Presidential Prize from the Korean government in 2016, official commendations from the Ministry of Science and ICT as well as the Ministry of Industry in 2016, the Excellent Researcher Award from the Korea Research Council in 2013, the Industrial Contributor Award from the Korean Federation of SMEs in 2013, and medals from Samsung's Thesis Prizes in 1997 and 1999.