LUTein: Dense-Sparse Bit-slice Architecture with Radix-4 LUT-based Slice-Tensor Processing Units

Dongseok Im and Hoi-Jun Yoo
Korea Advanced Institute of Science and Technology (KAIST)
{*dsim, hjyoo*}@*kaist.ac.kr*

2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 979-8-3503-9313-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/HPCA57654.2024.00063

***Abstract*—Bit-slice architectures have been developed to support various bit-precision and data sparsity of deep neural networks (DNNs). However, because of low-bit precision and a wide range of sparsity of bit-slice computations, bit-slice architectures are challenging in multiplier-, processing element (PE)-, core-, and software (SW)-level designs. First, data sparsity causes a power trade-off between the Radix numbers of a multiplier, and the previous multipliers cannot take advantage of all sparsity ranges. Second, a bit-slice PE which integrated massive low-bit multiplier-and-accumulate (MAC) units brings about large data transactions compared to a fixed bit-width PE. Third, bit-slice core architectures only focus on either dense or sparse data computations, limiting the overall performance of bit-slice computations with a wide sparsity range. Lastly, low-bit bit-slice computations cause massive repetitive instruction fetches across hardware units. To solve the challenges, LUTein is proposed. It exploits the new lookup table (LUT)-based computing method to support the Radix-4 Modified Booth algorithm, achieving low power consumption in all sparsity ranges. Moreover, the slice-tensor PE efficiently processes slice-tensor data by sharing hardware units across the Radix-4 LUT-based MAC units. In addition, the LUTein architecture adopts a systolic datapath with a multi-port buffer to exploit both inter-PE data reuse and slice-level sparsity. Lastly, LUTein's instruction set architecture (ISA) and the hierarchical instruction decoder are introduced to alleviate repetitive instruction fetches. As a result, LUTein outperforms the state-of-the-art bit-slice architecture, Sibia, over 1.34× higher energy-efficiency and 1.78× higher area-efficiency.**

***Keywords*—Bit-scalable architecture, deep neural network, Modified Booth algorithm, bit-slice, sparsity, LUT-based computing, slice-tensor, dense-sparse core architecture**

I. Introduction

Recently, deep neural networks (DNNs) have become crucial solutions in emerging applications such as 1-D natural language processing [8], [22], 2-D image processing [13], and 3-D point cloud processing [40]. As an effective DNN optimization, quantization lowers the bit-precision of DNNs, reducing the computational costs and energy consumption of DNN executions. To support various bit-precision of quantized DNNs, many bit-slice architectures [11], [15], [16], [24], [30], [34], [36] have been developed by integrating massive numbers of low-bit multiplier-and-accumulate (MAC) units and matching bit-precision in spatial- and temporal-multiplexing methods. Moreover, sparse bit-slice architectures [11], [15] speed up quantized DNN executions by exploiting slice-level sparsity at zero and near-zero data, achieving state-of-the-art performance in both dense and sparse DNN accelerations.

A bit-slice architecture processes low-bit (e.g. 4-bit) bit-slice data with a wide range of sparsity. After bit-slice decomposition on dense DNN models, high-order bit-slices are usually sparse, whereas low-order bit-slices are usually dense. For example, a dense YoloV3 [31] model shows high sparsity ($> 70\%$) at the most significant bit (MSB)-slice while low sparsity ($< 30\%$) at the least significant bit (LSB)-slice after decomposition into 4-bit signed bit-slices as shown in Fig 1 (a). In sparse DNN models, on the other hand, the LSB-slice shows various sparsity ranges ($10 \sim 90\%$) in ResNet18 [12] input data due to the ReLU activation function in Fig 1 (b). Therefore, a bit-slice architecture should efficiently accelerate low-bit bit-slice computations with a wide range of sparsity for energy-efficient dense and sparse DNN executions.

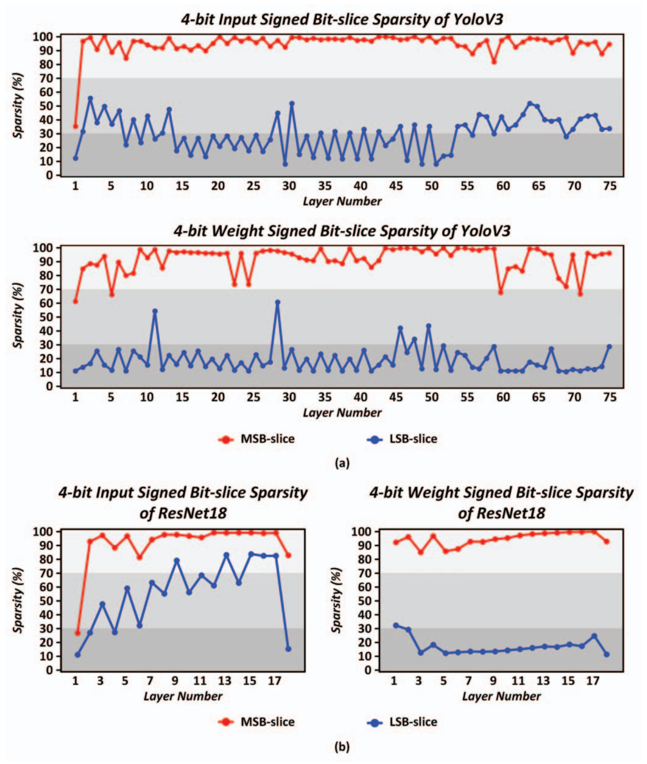


Fig. 1. A wide range of 4-bit signed bit-slice sparsity distribution in (a) a dense YoloV3 model and (b) a sparse ResNet18 model.

However, these properties cause critical challenges in energy-efficient multiplier-, processing element (PE)-, core-, and software (SW)-level designs of a bit-slice architecture.

The challenges are measured under 28 nm technology with 500 MHz of clock frequency.

Multiplier-level Challenge: A power consumption trade-off by a sparsity ratio arises at a low-bit multiplier unit. Fig. 2 (a) describes the 4-bit Baugh-Wooley multiplier [2] which is a basic array multiplier for signed multiplications. The Baugh-Wooley multiplier is sensitive to bit-toggling activity due to multiple signed multiplication stages. Therefore, frequent data change in dense data computations brings about high power consumption in the Baugh-Wooley multiplier. On the other hand, the 4-bit Modified Booth multiplier [32] in Fig. 2 (b) optimizes signed multiplication stages by encoding Radix-4 representation, consuming lower power than the Baugh-Wooley multiplier for dense data computations ($< 30\%$) as shown in Fig. 3. However, due to reduced signed multiplication stages, the Modified Booth multiplier is less sensitive to bit-toggling activity compared to the Baugh-Wooley multiplier, showing a small amount of power reduction as sparsity increases. Therefore, the Modified Booth multiplier consumes higher power than the Baugh-Wooley multiplier for sparse data computations ($\geq 30\%$). As a result, a bit-slice architecture has to integrate a new multiplier to cover a wide range of sparsity for low-power bit-slice computations.

PE-level Challenge: A bit-slice processing element (PE) integrated with multiple low-bit MAC units requires more data transactions than a fixed bit-width PE under the same bit-precision computation. When performing 16-bit fixed-point multiplication, a fixed bit-width PE integrated with a 16-bit MAC unit fetches two 16-bit operands and generates one 32-bit output. On the other hand, a bit-slice PE integrates sixteen 4-bit MAC units, requiring thirty-two 4-bit operands and producing sixteen 8-bit outputs to achieve the same throughput as a 16-bit fixed bit-width PE. Therefore, a bit-slice PE requires huge intermediate data transactions, causing high power consumption. Finally, a 4-bit bit-slice PE shows a $1.62\times$ larger bit-normalized area and a $1.46\times$ higher bit-normalized power consumption than a 16-bit fixed bit-width PE. For this reason, a new PE design is necessary to efficiently execute low-bit bit-slice computations.

Core-level Challenge: A sparse bit-slice core architecture consisting of multiple PEs with zero skipping units shows inefficient performance for dense bit-slice computations. In specific, a sparse bit-slice core architecture only focuses on performance enhancement for sparse MSB-slice computations, whereas it consumes $1.46\times$ higher power than a systolic array-based dense bit-slice core architecture for dense LSB-slice computations because of complex zero skipping units and low data reuse opportunities. Although a sparse bit-slice core architecture achieves $2.83\times$ higher throughput with 27.6% lower energy dissipation than a systolic array-based dense bit-slice core architecture for total DNN execution, its LSB-slice computations account for 75.9% of total energy dissipation. Consequently, a bit-slice architecture has to not only increase the performance for sparse bit-slice computations but also reduce power consumption for dense bit-slice computations to minimize total energy consumption on DNN acceleration.

SW-level Challenge: A bit-slice architecture executes large numbers of instructions compared to a fixed bit-width architecture. The granularity of instruction becomes finer to control a fine-grained bit-slice architecture. Therefore, a bit-slice architecture executes iterative bit-slice computing and memory operations across hardware units by fetching repetitive instructions. For example, a bit-slice architecture fetches $4\times$ more instructions to execute four 4-bit MAC operations than an 8-bit MAC operation by a fixed bit-width architecture. Therefore, the CPU becomes busier to execute massive repetitive instructions of fine-grained bit-slice computations. As a result, specialized instruction set architecture (ISA) and its decoding unit for a bit-slice architecture are necessary.

This paper introduces the energy-efficient dense-sparse bit-slice architecture, LUTein, by resolving the challenges with bottom-up optimizations in multiplier-, PE-, core-, and SW-level designs.

- **Multiplier-level Design**: The Radix-4 lookup table (LUT)-based multiplier is proposed to efficiently perform the Radix-4 Modified Booth algorithm [32]. The LUT stores all Radix-4 partial products, and then the LUT address generator loads one of them by accessing the LUT based on a Radix-4 encoded operand. This LUT-based computing method constructs a parallel datapath between the LUT updater and the LUT address generator in the Radix-4 LUT-based multiplier, achieving low area and power consumption in all sparsity ranges.
- **PE-level Design**: The shared slice-tensor PE is newly introduced to minimize bit-normalized area and power consumption. The slice-tensor PE shares LUT entries and partial product accumulators across the multiple Radix-4 LUT-based MAC units while exploiting input, weight, and output data reuse. The shared slice-tensor PE efficiently computes slice-tensor data, achieving lower bit-normalized area and power consumption than a 16-bit fixed bit-width PE.
- **Core-level Design**: The dense-sparse slice-tensor core architecture is proposed for efficient dense and sparse bit-slice computations. It integrates the 2D slice-tensor PE array with a systolic datapath and supports zero-skip operations using a multi-port register file. By exploiting slice-level sparsity as well as inter-PE data reuse, the dense-sparse slice-tensor core architecture achieves low energy consumption for both dense and sparse bit-slice computations.
- **SW-level Design**: The LUTein's ISA with the hierarchical instruction decoder is developed for efficient bit-slice computations. The ISA provides group-level instructions which configure group cores simultaneously. The top instruction decoder multicasts group-level instructions to the target group cores and each core's instruction decoder configures hardware units. Finally, the LUTein's ISA with the hierarchical instruction decoder reduces the number of instruction fetches and minimizes the involvement of the CPU for bit-slice computations.

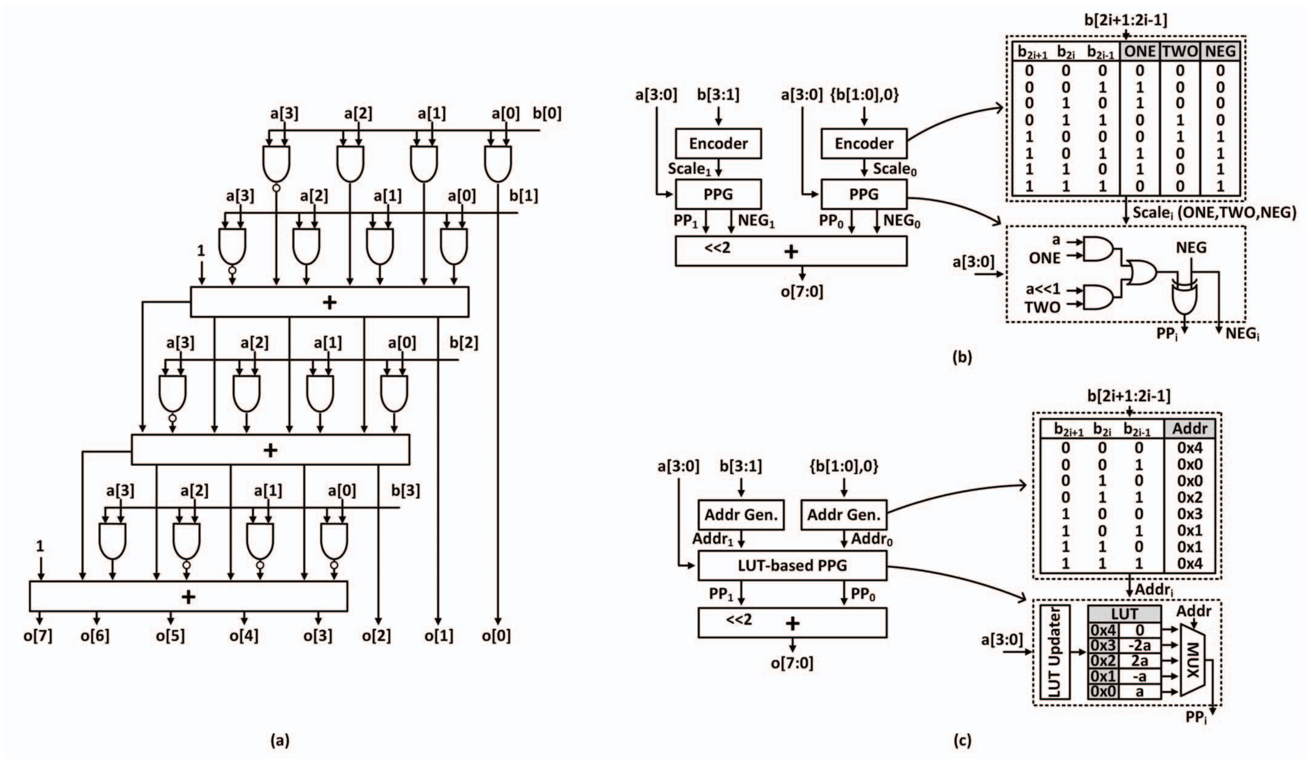


Fig. 2. The 4-bit signed multipliers: (a) the Baugh-Wooley multiplier, (b) the Radix-4 Modified Booth multiplier, and (c) the proposed Radix-4 LUT-based multiplier.

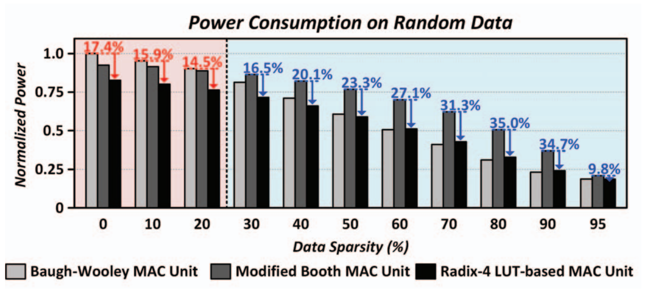


Fig. 3. Power consumption of the 4-bit Baugh-Wooley MAC unit, the 4-bit Radix-4 Modified Booth MAC unit, and the 4-bit Radix-4 LUT-based MAC unit under different sparsity.

II. LUTein Architecture

A. Multiplier-level Design: Radix-4 LUT-based Multiplier

Fig. 2 (c) illustrates the 4-bit Radix-4 LUT-based multiplier which consists of two LUT address generators and one LUT-based partial product generator (PPG) unit. The multiplier performs the Radix-4 Modified Booth algorithm [32] using the proposed LUT-based computing mechanism. Unlike the generation of three Radix-4 encoding signals (ONE, TWO, and NEG), the Radix-4 LUT-based multiplier produces an address from 0x0 to 0x4 to access five Radix-4 partial products $(a, -a, 2a, -2a, 0)$ in the LUT, where a is multiplicand data. The LUT address generator receives a group of three consecutive multiplier bits ($b[2i{+}1{:}2i{-}1]$) and generates an address of a corresponding Radix-4 encoded number. For 4-bit multiplier data $b[3{:}0]$, two LUT address generators produce two addresses about two groups of bits, $b[3{:}1]$ and $\{b[1{:}0], 0\}$, respectively. At the same time, a LUT updater in the LUT-based PPG unit pre-computes five Radix-4 partial products corresponding to 4-bit multiplicand data, $a[3{:}0]$, and stores them into the LUT. A partial product, PP, is obtained by accessing the LUT based on the address. Consequently, an 8-bit final multiplication result, $o[7{:}0]$, is completed by adding up the partial products shifting 2 bits to a high-order group.

The parallel datapath between the LUT address generator and the LUT updater reduces a critical path of the Radix-4 LUT-based multiplier. In the conventional Modified Booth multiplier as shown in Fig. 2 (b), the PPG unit waits for Radix-4 encoding signals (ONE, TWO, and NEG) from the encoder unit to generate partial products, causing a long critical path delay. In the proposed multiplier, on the other hand, the LUT updater calculates all Radix-4 partial products $(a, -a, 2a, -2a, 0)$ regardless of multiplier bits, and the LUT address generator produces the address of the LUT simultaneously that selects the multiplexer (MUX) inputs. This parallel datapath changes a critical path and reduces the total delay of the multiplier. Although the Radix-4 LUT-based multiplier requires 1.20× more number of logic cells than the Modified Booth multiplier, its critical path delay is 26.0% shorter at 28

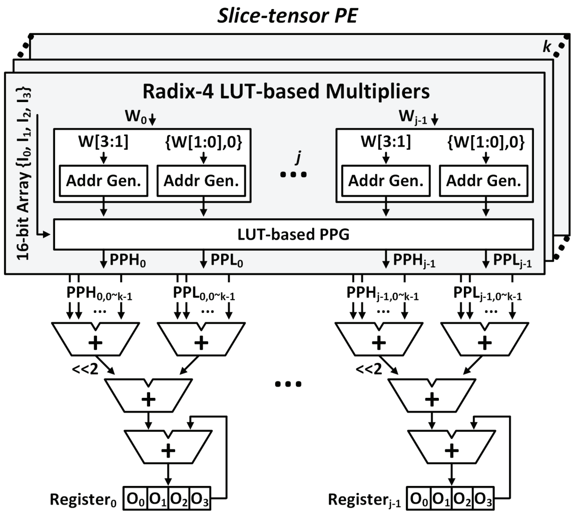


Fig. 4. Slice-tensor PE integrated the Radix-4 LUT-based multipliers with data reuse exploitation.

nm technology with 500 MHz of clock frequency. Moreover, LUT entries can be shared across multiple bit-order groups of the multiplicand. With a constrained timing path, the Radix-4 LUT-based multiplier occupies 2.2% lower the total logic cell area than the Modified Booth multiplier.

Fig. 3 shows the power consumption of 4-bit multipliers. Since a shortened critical path delay reduces the power consumption by alleviating spurious transition of digital circuits [3], the Radix-4 LUT-based multiplier achieves lower power consumption than the Modified Booth multiplier in all sparsity ranges. Moreover, this optimization alleviates the power gap against the Baugh-Wooley multiplier for sparse data computations, showing minimum power overhead ($<5\%$) in sparse ranges ($\geq$ 70%). Therefore, the Radix-4 LUT-based multiplier overcomes the power tradeoff caused by sparsity, achieving 17.4% lower power consumption than the Baugh-Wooley multiplier at 0% sparsity while 35.0% lower power consumption than the Modified Booth multiplier at 80% sparsity. As a result, the Radix-4 LUT-based multiplier achieves low power consumption in all sparsity ranges, taking an advantage of a wide sparsity range of bit-slice computations.

B. PE-level Design: Slice-Tensor PE

The slice-tensor PE is designed to efficiently process slice-tensor data with data reuse exploitation. Data reuse is a useful optimization method to save data transactions across MAC units for DNN operations [4]. For example, an input pixel is multiplied with different output channels of weights (input reuse), a weight element is multiplied with spatially different input pixels (weight reuse), and output partial sums are accumulated in a channel-wise direction to generate an output pixel (output reuse). The slice-tensor PE also exploits the data reuse method on bit-slice data. It loads spatially different input bit-slices and different output channels of weight bit-slices, as slice-tensor data, from memory, and then input and weight slice-tensor data are multicasted to local multipliers. After multiplication, local accumulators add up their output partial sums and reduced outputs are stored into memory. Therefore, the slice-tensor PE simultaneously exploits input, weight, and output reuse on slice-tensor data, alleviating redundant input, weight, and output data transactions from and to the memory.

The Radix-4 LUT-based multiplier is compatible with data reuse exploitation with shares of LUT entries and partial product accumulators. Fig. 4 describes the shared slice-tensor PE which is integrated with the multiple Radix-4 LUT-based multipliers. The LUT-based PPG unit is shared across the multiple LUT address generators. Once the LUT updates its entries to pre-computed partial products, the multiple LUT address generators produce LUT addresses for different output channels of weights and access the same LUT. For example, $2\times j$ numbers of the LUT address generators access the single LUT and generate high-order partial products (PPH) and low-order partial products (PPL) with j numbers of output channels, respectively. Then, PPH and PPL are summed up by shifting 2 bits to PPH to obtain final multiplication results. As a result, the input reuse is performed by sharing LUT entries, additionally optimizing hardware resources.

Moreover, the multiple LUT-based multipliers share accumulators to add up partial sums of different input channels immediately. Therefore, it reduces the number of accumulation registers. To further optimize accumulation, the same bit-order partial products are firstly summed up by an adder tree, and then their results are added up by shifting bits to high-order partial sums. For example, PPH and PPL with k numbers of input channels are accumulated separately in different adder trees, and then their results are added up by shifting bits as shown in Fig. 4. Therefore, the slice-tensor exploits the output reuse by sharing an accumulator with k numbers of multipliers and reducing data transactions of partial sums for accumulation.

Similarly, reusing weight data across spatially different inputs can minimize weight data transactions. However, the number of inputs that share a weight element affects zero input skipping performance. Sharing with many inputs increases the number of data reuse times, but it degrades zero input skipping performance because of coarse-grained zero-skip operations. Based on the signed bit-slice sparsity exploitation [15], the slice-tensor PE processes four input bit-slices $\{I_0, I_1, I_2, I_3\}$ as a 16-bit data array and skips their multiplications if they are all zeros. Finally, the slice-tensor PE generates four output partial sums $\{O_0, O_1, O_2, O_3\}$ as shown in Fig. 4.

Fig. 5 shows bit-normalized area and power consumption of the Baugh-Wooley PE integrated with the 4-bit Baugh-Wooley multipliers, the Modified Booth PE integrated with the 4-bit Modified Booth multipliers, and the proposed slice-tensor PE integrated with the 4-bit Radix-4 LUT-based multipliers in 28 nm technology with 500 MHz of clock frequency. All of them achieve area and power consumption reductions as the number of shared data increases. However, unsharable logic designs of the Baugh-Wooley PE and the Modified Booth

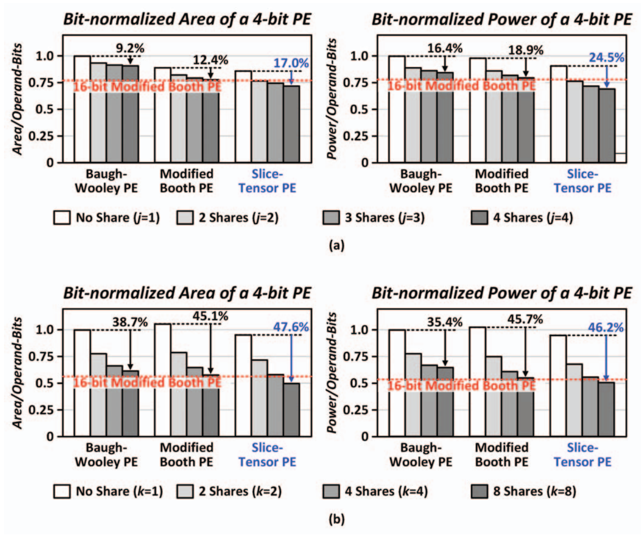


Fig. 5. Bit-normalized performance of the Baugh-Wooley PE, the Modified Booth PE, and the slice-tensor PE for (a) the input reuse by sharing the LUT and (b) for the output reuse by sharing an accumulator.

PE cause a small amount of area and power consumption reductions compared to the slice-tensor PE for the input reuse as shown in Fig. 5 (a). On the other hand, j numbers of the address generators in the slice-tensor PE share the LUT for the input reuse, showing 17.0% area reduction and 24.5% power reduction at $j = 4$. In the case of input channel accumulation (Fig. 5 (b)), addition overheads of many signed multiplication stages in the Baugh-Wooley algorithm bring about relatively a small amount of area and power reductions compared to the Modified Booth algorithm. Moreover, k numbers of LUT-based multipliers in the slice-tensor PE share an accumulator for the output reuse, achieving 47.6% area reduction and 46.2% power reduction at $k = 8$. Based on the performance results, the slice-tensor PE is designed to execute matrix multiplications between 4×8 sized input and weight slice-tensors simultaneously, and more data sharing degrades the flexibility of the PE architecture while power and area reductions saturate. It achieves 20.4% lower area and 13.6% lower power consumption than the Baugh-Wooley PE and 7.8% lower area and 29.4% lower power consumption than the Modified Booth PE at 50% sparsity whose amounts become much larger than only MAC performance in Fig. 3. As a result, the 4-bit slice-tensor PE efficiently exploits data reuse compared to the other multiplier-based PEs, finally achieving lower area and power consumption than the 16-bit Modified Booth PE.

C. Core-level Design: Dense-Sparse Slice-Tensor Core Architecture

Fig. 6 shows the dense-sparse slice-tensor core architecture which integrated the 2D slice-tensor PE array. It adopts a systolic PE array for efficient dense data computations [20]. In specific, the slice-tensor PE loads 4×8 sized input slice-tensor data from an input buffer (IBUF) and then passes it from left to right of the array after finishing computations. Then, the horizontal slice-tensor PEs fetch different output channels (oCHs) of 4×8 sized weight slice-tensor data from four weight buffers (WBUFs) and multiplies them with input slice-tensor data, and the output partial sums are accumulated along input channels (iCHs) to obtain reduced outputs. By using the horizontal systolic datapath, the slice-tensor PE array exploits inter-PE data reuse across the horizontal PEs.

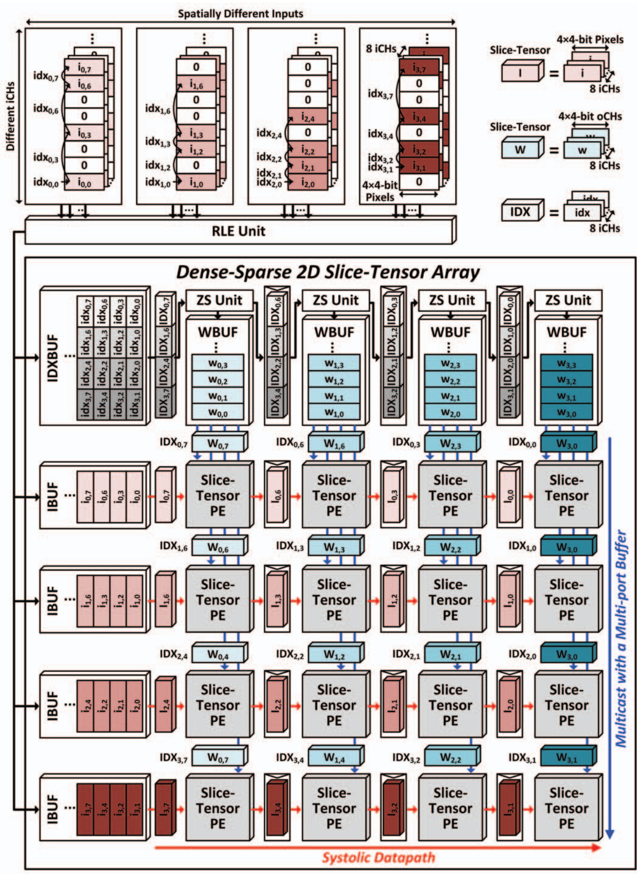


Fig. 6. Dense-sparse slice-tensor core architecture with the 2D slice-tensor PE array.

For efficient sparse data computations, the dense-sparse slice-tensor core architecture exploits input sparsity. Raw inputs are zero-encoded in the run-length encoding (RLE) unit, and only non-zero inputs and its indices are propagated to IBUFs and index buffers (IDXBUFs), respectively. Then, the 2D slice-tensor PE array performs multiplication between non-zero input data and the corresponding weight data without latency. However, an input zero-skip operation breaks a systolic datapath because of irregular weight data access corresponding to non-zero input data. To support zero-skip operations, a multicasting datapath with a multi-port buffer is adopted and supplies irregular weight data to vertical slice-tensor PEs while maintaining a horizontal systolic datapath. In specific, the four vertical slice-tensor PEs load spatially different non-zero

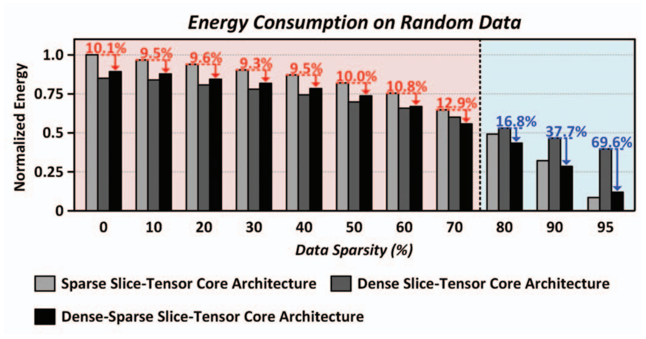


Fig. 7. Energy consumption of the sparse slice-tensor core architecture, the dense slice-tensor core architecture, and the dense-sparse slice-tensor core architecture under different sparsity.

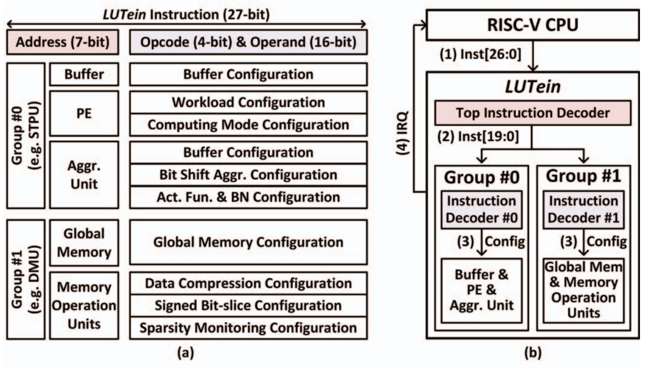


Fig. 8. (a) LUTein's instruction set architecture (ISA) and (b) the hierarchical instruction decoder.

inputs from four IBUFs and request different addresses of weight data to a single WBUF for zero-skip operations. At the same time, the zero skipping (ZS) unit loads non-zero indices (IDX) and calculates addresses of weight data to be multiplied with non-zero inputs. Then, the four vertical slice-tensor PEs simultaneously load corresponding weights from a multi-port WBUF based on the addresses. Indices also flow in a horizontal systolic datapath that the ZS unit loads non-zero indices from IDXBUF and passes them from left to right of the array. For dense data computations, the ZS units and IDXBUFs are deactivated, and each WBUF broadcasts weight data to the four vertical slice-tensor PEs, generating spatially different output partial sums. Therefore, weight-slice tensor data is reused in the spatially different input pixels in the vertical multicasting datapath. As a result, the dense-sparse slice-tensor core architecture performs both dense and sparse data computations by exploiting inter-PE data reuse and input sparsity in the 2D slice-tensor PE array.

Fig. 7 illustrates the energy consumption of the sparse slice-tensor core architecture, the dense slice-tensor core architecture, and the dense-sparse slice-tensor core architecture under different sparsity. The sparse slice-tensor core architecture is composed of the 2D slice-tensor PE array integrated with ZS units, IBUFs, IDXBUFs, and WBUFs at every PE to maximize zero-slice-skip performance. On the other hand, the dense slice-tensor core architecture focuses on dense bit-slice computations integrated with the 2D systolic slice-tensor PE array without ZS units and IDXBUFs. Each core architecture is designed in 28 nm technology with 500 MHz of clock frequency.

The dense slice-tensor core architecture consumes lower energy than the other architectures for dense data computations by exploiting inter-PE data reuse and minimizing the overheads of hardware units. However, it cannot speed up sparse data computations, resulting in higher energy consumption than the sparse slice-tensor core architecture over 80% sparsity, even showing 4.43× higher energy consumption at 95% sparsity. On the other hand, the sparse slice-tensor core architecture shows significantly low energy consumption at high sparsity ($\geq$80%) thanks to slice-level sparsity exploitation, whereas consuming high energy for dense data computations due to the complexity of the zero skipping hardware units. The dense-sparse slice-tensor core minimizes energy consumption for dense data computations using inter-PE data reuse by the horizontal systolic datapath, showing only 5% of energy consumption overhead against the dense slice-tensor core architecture and achieving 10.1% of energy consumption reduction compared to the sparse slice-tensor core architecture at 0% sparsity. Moreover, performing zero-skip operations by the vertical multicasting datapath maximizes throughput for sparse data computations, resulting in lower energy consumption than the sparse slice-tensor core architecture in all of a sparsity range except for over 95% sparsity and achieving 69.6% of energy consumption reduction compared to the dense slice-tensor core architecture at 95% sparsity. Furthermore, in a 70–90% sparsity range, the dense-sparse slice-tensor core architecture achieves the lowest energy consumption among the cores. Consequently, the dense-sparse slice-tensor core architecture achieves low energy consumption in a wide sparsity range of bit-slice computations.

D. SW-level Design: LUTein's Instruction Set Architecture

LUTein executes DNN models by retrieving instructions from the CPU. However, instructions of the CPU (e.g. RISC-V CPU) are incompatible with bit-slice-based DNN executions, which consist of a large number of iterative low-bit computing and memory operations. Since the CPU in a system-on-chip platform is busy controlling other hardware units (e.g. external memory IFs, Peri IPs, etc), LUTein has to be controlled with minimum involvement of the CPU. To solve this problem, the specialized LUTein's ISA and the hierarchical instruction decoder are provided.

Fig. 8 (a) summarizes the LUTein's ISA. All LUTein hardware units are controlled by a 27-bit instruction (7-bit address, 4-bit opcode, and 16-bit operand). A 7-bit address indicates group cores or individual cores. Therefore, group-level instructions configure group cores simultaneously, avoiding repetitive instruction fetches across group cores. The remaining 20-bit instruction provides hardware configurations. In specific, a slice-tensor processing unit (STPU) instruction

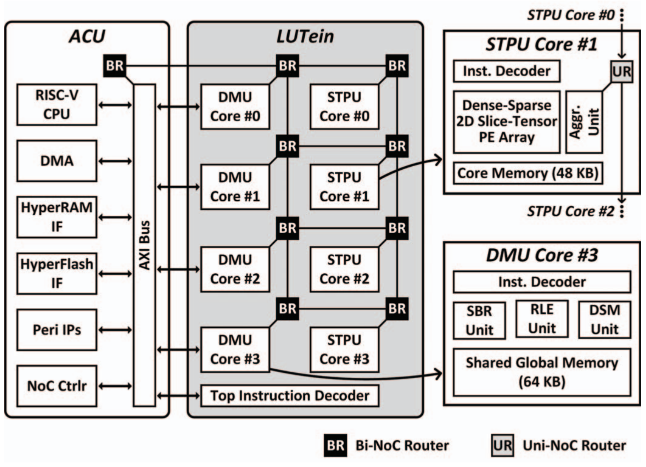


Fig. 9. Overall architecture of a LUTein evaluation platform.

defines the configuration of buffers, PEs, and aggregation units for bit-slice computing operations. A data management unit (DMU) instruction specifies bit-slice memory operations and configures global memory and memory operation units.

Fig. 8 (b) shows a mechanism of the hierarchical instruction decoder. (1) LUTein fetches a 27-bit instruction from the RISC-V CPU. (2) The top instruction decoder reads a high 7-bit instruction and sends the remaining 20-bit instruction to target cores based on a 7-bit address. The top instruction decoder multicasts the 20-bit instructions to group cores simultaneously if their configurations are the same. (3) The instruction decoder in each core decodes the 20-bit instruction and configures the core by writing them to configuration registers. After configurations, the run instruction activates the PEs to perform matrix multiplications. The core itself counts addresses of memory and buffer to load data without involving the RISC-V CPU. (4) After reaching an end address of workloads, LUTein raises an interrupt to the RISC-V CPU, and the RISC-V CPU then transfers the next instruction streams. LUTein fetches the run instruction again without reconfiguring the PEs for the next input or weight data if the workload configuration is the same as the previous one. The LUTein's ISA is compatible with the hierarchical instruction decoder, reducing 64.5% of instruction fetches during 7-bit matrix multiplications. Consequently, the specialized LUTein's ISA and the hierarchical instruction decoder reduce the number of instruction fetches and minimize the involvement of the CPU.

E. Overall Architecture of LUTein

Fig. 9 describes the overall architecture of LUTein that adopts proposed multiplier-, PE-, core-, and SW-level designs for efficient dense and sparse 4-bit bit-slice computations. LUTein consists of a quad-core slice-tensor processing unit (STPU) and a quad-core data management unit (DMU). A single STPU core integrates 2048 numbers of $4b \times 4b$ Radix-4 LUT-based MAC units with 8 KB of SRAM-based IBUFs and 2 KB of register file-based WBUFs. An aggregation unit aggregates outputs of the slice-tensor PEs and applies activation functions and batch normalization on final convolution outputs. A 48 KB core memory temporally stores intermediate data and output data and is also used as a prefetch memory for IBUFs and WBUFs.

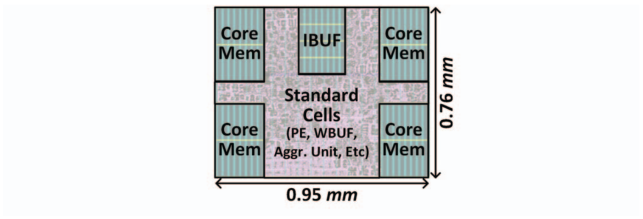


Fig. 10. Layout image of the STPU core.

The DMU integrates a total of 256 KB global memory and the bit-slice memory operation units [15] such as the signed bit-slice representation (SBR) unit, the zero run-length encoding (RLE) unit, and the dynamic sparsity monitoring (DSM) unit. Each unit executes memory-intensive operations by directly accessing the global memory without occupying the network-on-chip (NoC). The global memory is shared across all cores and stores intermediate data if the core memory is fully utilized.

LUTein adopts the heterogeneous NoC [15] for flexible and efficient data transmissions. The bidirectional NoC (Bi-NoC) connects all cores including external hardware units in a 2D mesh. It unicasts or multicasts data in any direction among the cores. On the other hand, the unidirectional NoC (Uni-NoC) connects aggregation units across STPU cores. The Uni-NoC transfers a large bit-width of partial sum data to the adjacent aggregation unit with optimized bandwidth for aggregation.

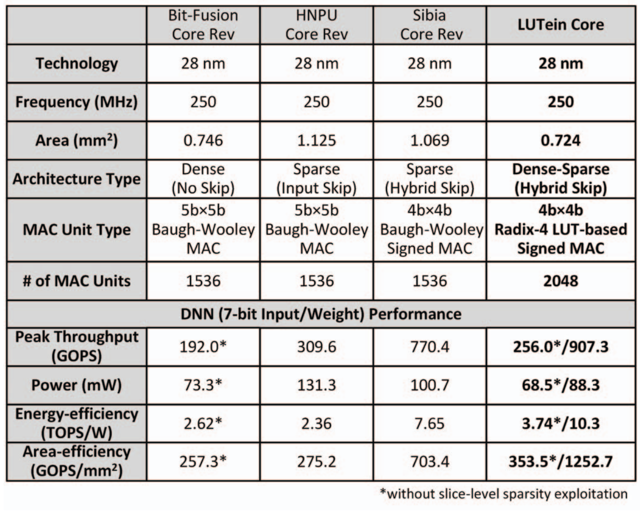
III. Evaluation

A. Simulation Environment

LUTein is designed in RTL Verilog and synthesized in Samsung 28 nm CMOS technology. Fig. 10 illustrates the post-layout of the LUTein's STPU core after the place-and-route process. The STPU core shows $0.76mm \times 0.95mm$ of area. In the placement, the core memory SRAM is divided into four partitions that are placed in each corner of the STPU core. IBUF SRAM is placed in the top center. Standard cells (PE, WBUF, aggregation unit, etc) are placed in the rest area. In the routing, the top power metal is the IA layer, and the top signal metal is the $M7$ layer.

LUTein is evaluated using the evaluation platform. The evaluation platform consists of an architecture control unit (ACU) and LUTein as shown in Fig. 9. The evaluation platform is designed in all RTL Verilog. Therefore, the LUTein architecture is evaluated through cycle-accurate RTL simulations and power measurement simulations using the Synopsys PrimeTime PX. The evaluation platform is compatible with the extension of additional functional units by connecting them to an ACU's AXI bus. For example, the evaluation platform supports end-to-end executions of VoteNet and DGCNN

TABLE I
PERFORMANCE COMPARISON AMONG BIT-SLICE ACCELERATORS.



	Bit-Fusion Core Rev	HNPU Core Rev	Sibia Core Rev	**LUTein Core**
Technology	28 nm	28 nm	28 nm	**28 nm**
Frequency (MHz)	250	250	250	**250**
Area (mm^2)	0.746	1.125	1.069	**0.724**
Architecture Type	Dense (No Skip)	Sparse (Input Skip)	Sparse (Hybrid Skip)	**Dense-Sparse (Hybrid Skip)**
MAC Unit Type	5b×5b Baugh-Wooley MAC	5b×5b Baugh-Wooley MAC	4b×4b Baugh-Wooley Signed MAC	**4b×4b Radix-4 LUT-based Signed MAC**
# of MAC Units	1536	1536	1536	**2048**
DNN (7-bit Input/Weight) Performance				
Peak Throughput (GOPS)	192.0*	309.6	770.4	**256.0*/907.3**
Power (mW)	73.3*	131.3	100.7	**68.5*/88.3**
Energy-efficiency (TOPS/W)	2.62*	2.36	7.65	**3.74*/10.3**
Area-efficiency (GOPS/mm^2)	257.3*	275.2	703.4	**353.5*/1252.7**

*without slice-level sparsity exploitation

benchmarks by integrating a 3D point processing unit [14], which accelerates a neighbor search and 3D point sampling operations. The evaluation results report the performance of the LUTein architecture excluding the ACU performance.

The ACU consists of the RISC-V CPU, direct memory access (DMA), an external memory interface, peripheral IPs, and a network router, all of which are connected by an AXI bus. The RISC-V CPU executes DNN models by running the program compiled with the RISC-V ISA. The LUTein architecture retrieves instruction streams from the ACU through an AXI bus and raises an interrupt after finishing the allocated workloads. The external DRAM and Flash memories are modeled by the Cypress Semiconductor's HyperRAM [7] and HyperFlash [6], respectively. The ACU communicates with the LUTein architecture through an AXI bus and the interconnect NoC. For external memory access, the ACU directly transfers data between external memory and LUTein's global memory through an AXI bus. For internal data transmission, the ACU's NoC controller (NoC Ctrlr) configures the LUTein's interconnect NoC and transfers high bandwidth data among LUTein's hardware units.

B. Overall Performance Comparison with Bit-slice Accelerators

Table I shows performance comparisons with previous bit-slice accelerators. All of the accelerators are synthesized in 28 nm Samsung technology with 250 MHz of clock frequency to match the same conditions for a fair comparison. Bit-fusion [36], HNPU [11], and Sibia [15] integrated the Baugh-Wooley MAC units while LUTein exploits the Radix-4 LUT-based MAC units. In terms of a bit-width of a MAC unit, Bit-fusion and HNPU exploits a $5b \times 5b$ MAC unit for 4-bit bit-slice computations because of sign-bit extension to process both signed and unsigned bit-slices. On the other hand, Sibia and LUTein adopt a $4b \times 4b$ signed MAC unit which does not require sign-bit extension in MAC units for all 4-bit signed bit-slice computations thanks to the SBR algorithm [15].

Bit-fusion is a dense bit-slice architecture and cannot accelerate sparse high-order bit-slice computations, resulting in low performance of the total 7-bit DNN execution in terms of throughput, energy efficiency, and area efficiency compared to sparse bit-slice architectures. With the bottom-up optimizations of the LUTein architecture, LUTein shows 3.0% of the smaller logic area even integrating 1.33× more MAC units than the dense Bit-fusion architecture. As a result, LUTein with the deactivation of slice-level sparsity exploitation achieves 1.33× higher throughput, 6.6% lower power consumption, 1.43× higher energy efficiency, and 1.38× higher area efficiency than the Bit-fusion architecture.

With slice-level sparsity exploitation, the dense-sparse LUTein architecture outperforms the sparse HNPU and Sibia architectures. Moreover, LUTein increases throughput by skipping zero bit-slice computations caused by near-zero data and supporting hybrid skipping that exploits more sparse data between input and weight. Therefore, LUTein shows 2.93× higher throughput and 32.7% lower power consumption than HNPU. In comparison with the state-of-the-art Sibia architecture, the LUTein architecture integrates more MAC units with a small logic area, resulting in 1.18× higher throughput while consuming 12.2% lower power than the sparse Sibia architecture. As a result, LUTein outperforms Sibia over 1.34× higher energy efficiency and 1.78× higher area efficiency.

C. Comparison with Bit-slice Accelerators on Various DNN Benchmarks

LUTein evaluates various DNN benchmarks that have various network structures (e.g. transformer, convolutional neural network (CNN), shared multi-layer perceptron (MLP), and depthwise separable convolution) and various bit-precision (e.g. 7-, 10-, and 13-bit) and various sparsity (10$\sim$90%) among different layers. Therefore, these benchmarks bring meaningful performance evaluations of hardware architectures. All of the benchmarks are applied post-training quantization (PTQ) with minimum accuracy degradation ($< 1\%$).

Albert [22] and Vision Transformer (ViT) [9] models are transformer-based networks that consist of attention modules and linear layers. The modules and layers of an Albert model are quantized into 7-, 10-, and 13-bit precision with low sparsity (10$\sim$20% on average) for the 1-D General Language Understanding Evaluation (GLUE) tasks [39] (MNLI, QQP, SST-2). Similarly, a ViT model is quantized into 7- and 10-bit precision with low sparsity (10$\sim$20% on average) for the 384×384 sized ImageNet dataset [33]. ResNet18 is a general CNN model with residual blocks, having 7-bit precision with sparse data ($\sim$50% on average) for ImageNet dataset [33]. YoloV3 has 75 numbers of convolution layers, and they are quantized to 7-bit precision with low sparsity ($\sim$30% on average) for COCO dataset [25]. MonoDepth2 is a fully convolutional encoder-decoder network. An encoder network has 7-bit precision with the ReLU activation function ($\sim$50% sparsity on average). On the other hand, a decoder network has

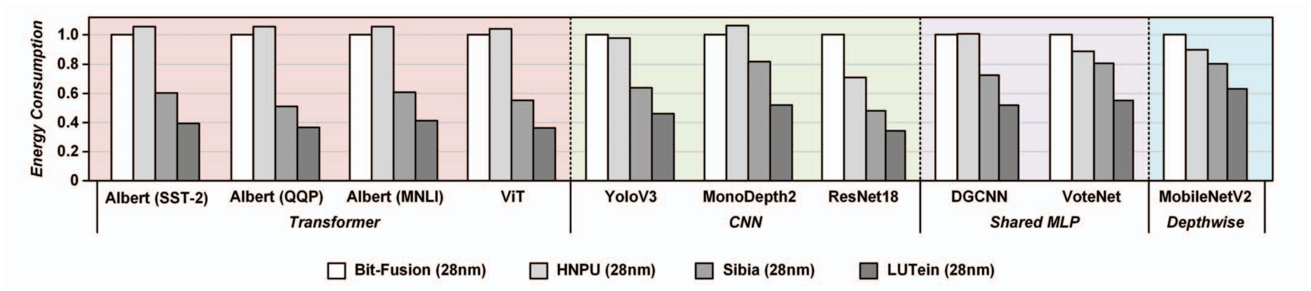


Fig. 11. Energy consumption comparison among bit-slice accelerators for various DNN benchmarks.

7-bit precision of weight data and 10-bit precision of input data with low sparsity (∼20% on average) because of the ELU activation function for NYU-Depth v2 dataset [28]. MobileNet2 is a light-weight network that consists of depthwise separable convolution layers. It requires 10-bit precision (∼40% sparsity on average) for ImageNet dataset [33]. DGCNN is a graph neural network for 3D point cloud data. It consists of shared MLPs with 7-bit precision and low sparsity (∼20% on average) for ModelNet40 dataset [41] VoteNet is a 3D point cloud-based neural network, having shared MLPs with 7-bit and high sparsity (∼50% on average) for SUN RGB-D dataset [37].

Fig. 11 shows the energy consumption comparison among bit-slice accelerators for the various DNN benchmarks. The sparse HNPU architecture [11] consumes more energy than the dense Bif-fusion architecture [36] for dense DNN benchmarks such as Albert, ViT, MonoDepth2, and DGCNN because the hardware complexity of slice-level sparsity exploitation causes high power consumption. With the help of the SBR algorithm, the Sibia architecture [15] improves zero-slice-skip performance by exploiting slice-level sparsity on both positive and negative near-zero data, showing lower energy consumption than the Bit-fusion architecture on all of the DNN benchmarks. The dense-sparse LUTein architecture, on the other hand, takes advantage of both slice-level sparsity exploitation and inter-PE data reuse, achieving the lowest energy consumption among the bit-slice architectures. In specific, LUTein efficiently executes low sparsity and various bit-precision of transformer models, achieving $1.53\times$, $1.39\times$, $1.47\times$, and $1.52\times$ lower energy consumption than Sibia for Albert (SST-2, QQP, MNLI) and ViT, respectively. Similarly, LUTein accelerates various sparsity of CNN models, achieving $1.39\times$, $1.57\times$, and $1.40\times$ lower energy consumption than Sibia for YoloV3, MonoDepth2, and ResNet18, respectively. In the performance of shared MLP networks, LUTein achieves $1.40\times$ and $1.46\times$ lower energy consumption than Sibia for DGCNN and VoteNet, respectively. LUTein shows $1.27\times$ lower energy consumption than Sibia for MobileNetV2, which is the smallest energy reduction among the DNN benchmarks due to low hardware utilization when executing depthwise separable convolution. Consequently, with the bottom-up hardware optimizations, the dense-sparse LUTein architecture surpasses the Bit-fusion, HNPU, and Sibia architectures on various structures, various bit-precision, and various sparsity of DNN models.

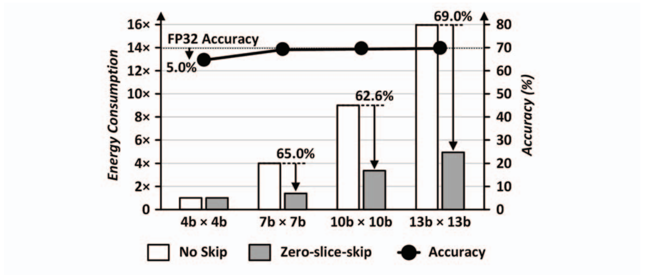


Fig. 12. Energy consumption and DNN accuracy analysis of an FC layer execution in ResNet18 with 4-bit, 7-bit, 10-bit, and 13-bit precision.

D. Performance Analysis of Bit-scalability

Bit-precision of a DNN model has an impact on the energy consumption of a bit-slice architecture. A bit-slice architecture consumes time and energy in proportion to the number of bit-slices to be processed. If a DNN model is quantized into lower bit-precision, a bit-slice architecture consumes less energy but has lower DNN accuracy. Therefore, a bit-slice architecture needs to be programmable and reconfigurable to execute any bit-precision of DNN models depending on a trade-off between energy consumption and DNN accuracy.

Fig. 12 describes energy consumption and DNN accuracy under different bit-precision of a fully-connected (FC) layer in ResNet18. Input and weight data of an FC layer are quantized into 4-bit, 7-bit, 10-bit, and 13-bit precision. Energy consumption of a bit-slice architecture increases quadratically as the number of input and weight bit-slices increases because a bit-slice architecture has to compute the increased numbers of both input and weight bit-slices. Therefore, energy consumption of $7b \times 7b$, $10b \times 10b$, and $13b \times 13b$ computations present $4\times$, $9\times$, and $16\times$ higher than $4b \times 4b$ computations, respectively. With the exploitation of slice-level sparsity and data reuse, the LUTein architecture significantly reduces energy consumption for high bit-precision computations. For example, energy consumption of $7b \times 7b$, $10b \times 10b$, and $13b \times 13b$ computations are reduced by 65.0%, 62.6%, and 69.0%, respectively. In overall DNN accuracy, ResNet18 shows negligible accuracy degradation until 7-bit quantization, but it loses 5.0% for 4-bit quantization. As a result, LUTein

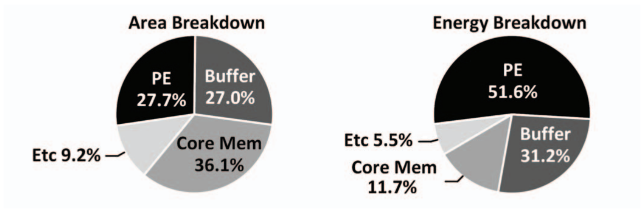


Fig. 13. Area and energy breakdown of the LUTein's STPU core.

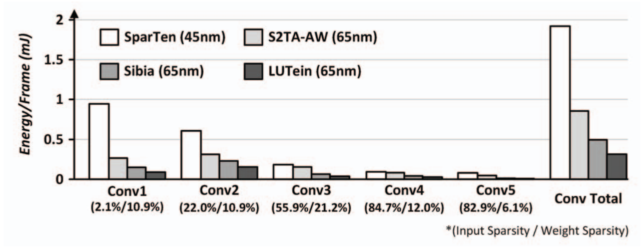


Fig. 14. Per-layer energy comparison with various types of sparse accelerators on AlexNet.

is programmable and reconfigurable for various bit-precision of DNN executions, taking advantage of a trade-off between energy consumption and accuracy.

E. Area and Energy Breakdown

Fig. 13 describes the area and energy breakdown of a LUTein's STPU core. The slice-tensor PE accounts for 27.7% of the total area, input/weight buffers take 27.0%, the core memory takes 36.1%, and the other components including the aggregation unit take 9.2%. The area of the core memory is dominant because of its large-sized memory. Energy consumption is dominated by the slice-tensor PE, taking 51.6% of the total energy. Input/weight buffers account for 31.2% of the STPU core, the core memory takes 11.7%, and the other components take 5.5%.

F. Per-layer Energy Comparison with Various Types of Sparse Accelerators

Fig 14 shows the comparison with various types of sparse accelerators in terms of per-layer energy consumption on AlexNet [33]. LUTein is re-synthesized in 65 nm technology for a fair comparison. SparTen [10] is a sparse 8-bit fixed bit-width accelerator exploiting unstructured sparsity. SparTen shows high energy consumption especially in Conv1 and Conv2 layers because it cannot exploit sparsity for dense convolution layers. Therefore, LUTein (65 nm) achieves 6.1× lower total energy consumption than Sparten (45 nm) even older technology node. S2TA-AW [26] is a sparse 8-bit fixed bit-width accelerator exploiting the structured sparsity on input and weight data. It reduces energy consumption on dense layers by using the lossy pruning method. On the other hand, the LUTein architecture exploits slice-level sparsity without any pruning method, consuming lower energy than S2TA-AW for all of the dense and sparse convolution layers. Therefore, LUTein achieves 2.7× lower total energy consumption than S2TA-AW. Sibia [15] is a sparse 4-bit bit-slice accelerator that accelerates sparse bit-slice computations on dense and sparse layers. On the other hand, LUTein efficiently processes both sparse and dense bit-slices, achieving 1.6× lower total energy consumption than sparse bit-slice Sibia architecture on the AlexNet benchmark. As a result, LUTein shows the lowest energy consumption on the DNN execution among the unstructured sparse accelerator, the structured sparse accelerator, and the sparse bit-slice accelerator.

G. Quantitative Analysis of Compatibility

Existing accelerators have supported integer-type multiplications for DNN accelerations. For example, the Google TPU [17], [18] and the NVIDIA H100 [29] GPU support massive numbers of 8-bit integer multipliers for DNN inferences. The proposed Radix-4 LUT-based multiplier is compatible with any integer-type architectures by only replacing multipliers. For quantitative analysis, the proposed 8-bit multipliers are integrated into the Google TPU-like 2D systolic array architecture that consists of $64k$ 8-bit MAC units with 28 MB memories. By only replacing existing multipliers with the proposed ones, 9.6% and 5.2% power reductions are achieved at 50% sparsity by replacing the Baugh-Wooley and the Modified Booth multipliers, respectively. Consequently, only the replacement of multipliers would achieve meaningful performance enhancement without modification of overall architectures, and further enhancement would be expected by adopting the proposed PE-, core-, and SW-level designs.

IV. Related Works

A. Bit-slice Architecture

Lowering bit-precision of operands reduces memory and computation overheads of accelerators. For example, an 8-bit input/weight quantized DNN requires almost the half data transactions and a quarter size of MAC units compared to a 16-bit input/weight DNN. Therefore, a quantization method is widely applied to DNN models, and DNN accelerators process them for high throughput and energy efficiency. However, bit-precision varies among DNN layers and models. Consequently, many bit-scalable architectures have been developed to efficiently execute various bit-precision of quantized DNN models.

A bit-slice architecture accelerates quantized DNNs using bit-slice computations. Full bit-width data is decomposed into several low-bit bit-slices, and a bit-slice architecture computes them by matching target bit-precision in spatial- and time-multiplexing methods. Bit-width of bit-slices is an important design point between hardware granularity and efficiency. For example, 1-bit [1], [19], [23], [35] (bit-serial architectures) and 2-bit [34], [36] bit-slice architectures enable fine-grained bit-precision computations, but their MAC units show inefficiency to compute high bit-precision data because of large intermediate data transactions and time-consuming computations. On the other hand, high-bit bit-slice (e.g. 8-bit [16], [24]) architectures have efficient MAC units for high bit-precision computations but lose bit-granularity. Since most DNNs cannot be quantized into below 4-bit precision, many bit-slice architectures [11], [15], [30] and LUTein have adopted 4-bit bit-slice computing architectures.

B. *Sparse Bit-slice Architecture*

Sparse bit-slice architectures [11], [15] skip zero bit-slice computations. After bit-slice decomposition, zero-bit-slices are produced at zero and near-zero data. For example, 00000110_2 of near-zero data is decomposed into 0000_2 and 0110_2 slices, a sparse bit-slice architecture enables skip high-order 0000-slice computations. Therefore, it exploits slice-level sparsity to enhance hardware performance not relying on the ReLU activation function where previous fixed bit-width zero-skip architectures [10], [26] cannot achieve high performance.

HNPU [11] skips zero-bit-slice computations by removing zero-bit-slices at the pre-fetch FIFO structure. Sibia [15] further exploited slice-level sparsity at negative near-zero data whose high-order bit-slice is 1111_2 by using the lossless signed bit-slice representation (SBR) that changes high-order 1111_2 bit-slices to 0000_2. However, they only focus on high throughput enhancement for sparse bit-slice computations by integrating complex zero skipping units, limiting the overall performance of DNN executions which have a wide range of slice-level sparsity. On the other hand, LUTein exploits both slice-level sparsity and data reuse for dense and sparse bit-slice computations, significantly reducing energy consumption in the total DNN executions.

C. *Low-bit Signed Multiplier*

An efficient multiplier design is crucial for DNN model execution which consists of massive numbers of MAC operations. Multiplication involves two stages: partial product generation and partial product addition. A multiplier computes partial products using combinational logic circuits, and then an adder tree sums them up to obtain a final multiplication output. In digital signal processing, the Baugh-Wooley multiplier and the Modified Booth multiplier are widely used, and Synopsys provides their building block IPs in DesignWare Libraries: Carry-save array multiplier (does not require a DesignWare license) and Radix-4 Booth recoded multiplier (requires a DesignWare license), respectively.

The Baugh-Wooley multiplier [2] is a straightforward way to multiply two 2's complement operands and generate a signed multiplication output. As shown in Fig. 2 (a), the Baugh-Wooley multiplier consists of an array multiplier that each multiplicand bit is applied AND operation to multiplier bits and then its partial products are accumulated. The Baugh-Wooley multiplier performs signed multiplication by embedding a sign bit handler such as NAND gates in an array multiplier. Then, the sign of partial products is kept positive and the accumulation is performed directly. However, the Baugh-Wooley multiplication method requires high power consumption for dense data computations because frequent data change toggles many bits of logic gates. Consequently, the Baugh-Wooley multiplier shows higher power consumption than the Radix-4 Modified Booth multiplier at low sparsity ($< 30\%$) where high bit-toggling activity occurs and finally consumes 8.2% higher power at 0% sparsity in Fig. 3.

The Modified Booth multiplier [32] performs signed multiplication with the Radix-4 representation. The multiplicand is encoded to the Radix-4 as 0, -1, 1, -2, or 2 and is multiplied by multiplier bits. Therefore, it reduces half the number of partial products, reducing the critical path delay compared to the Baugh-Wooley multiplier. Fig. 2 (b) describes the 4-bit Modified Booth multiplier [5] that consists of two Radix-4 Booth encoders, two partial product generator (PPG) units, and an accumulator. The Radix-4 Booth encoder receives three consequent bits of the multiplicand, $b[2i\text{+}1\text{:}2i\text{-}1]$, and generates three Radix-4 signals (ONE, TWO, and NEG). Then, a PPG unit produces partial products based on multiplier bits, $a[3\text{:}0]$, and Radix-4 signals. An accumulator accumulates all partial products generated by PPG units and completes a signed multiplication output. The Modified Booth multiplier enables high-speed multiplications due to the reduced signed multiplication stages. However, it cannot achieve significant power reduction for sparse data computations because the Radix-4 multiplication method is relatively less sensitive to bit-toggling activity. Therefore, the Modified Booth multiplier shows $1.63\times$ higher power than the Baugh-Wooley multiplier at 80% sparsity as shown in Fig. 3.

Previous works [21], [27] optimize the partial product addition of the Modified Booth multiplier using the Wallace tree [38]. However, in the low-bit (4-bit) Modified Booth multiplier, a partial product addition unit only occupies $2.0\times$ smaller logic cell area than a partial product generation unit because it adds up small numbers of partial products. Other works [5], [42] reduce the delay of partial product generation, but their non-sharable logic designs worsen logic area and power consumption in PE-level design. On the other hand, LUTein focuses on efficient partial product generation through the sharable LUT-based computing mechanism.

V. Conclusion

LUTein is proposed to efficiently accelerate bit-slice computations through the optimizations of the multiplier-, PE-, core-, and SW-level designs. First, the LUTein's multiplier performs the Radix-4 Modified Booth algorithm using the LUT-based computing mechanism. It reduces the critical path of the multiplier, achieving low power consumption in all sparsity ranges where other multipliers show a power trade-off. Second, the LUTein's PE processes slice-tensor data by exploiting data reuse and sharing hardware units, achieving lower bit-normalized area and power consumption than the existing PE architectures. Third, the LUTein's core exploits both inter-PE data reuse and slice-level sparsity by adopting the dense-sparse 2D slice-tensor PE array, achieving low energy consumption in all sparsity ranges. Lastly, the LUTein's SW minimizes repetitive instruction fetches during bit-slice computations by adopting the specialized ISA and the hierarchical instruction decoder. Consequently, LUTein achieves state-of-the-art performance for dense and sparse DNN benchmarks.

References

[1] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 382–394.

[2] C. Baugh and B. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Transactions on Computers*, vol. C-22, no. 12, pp. 1045–1047, 1973.

[3] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital cmos circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, 1995.

[4] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 44, no. 3, pp. 367–379, 2016.

[5] K.-J. Cho, K.-C. Lee, J.-G. Chung, and K. K. Parhi, "Design of low-error fixed-width modified booth multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 522–531, 2004.

[6] Cypress Semiconductor, "512-Mb (64 MB)/256-Mb (32 MB)/128-Mb (16 MB), 1.8 V/3.0 V, HYPERFLASH FAMILY Data-Sheet," *Cypress Int.*, Inc.

[7] Cypress Semiconductor, "64Mbit HyperRAM Self-Refresh DRAM Data-Sheet," *Cypress Int.*, Inc.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[10] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 151–165.

[11] D. Han, D. Im, G. Park, Y. Kim, S. Song, J. Lee, and H.-J. Yoo, "Hnpu: An adaptive dnn training processor utilizing stochastic dynamic fixed-point and active bit-precision searching," *IEEE Journal of Solid-State Circuits*, 2021.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] A. Ignatov, J. Patel, and R. Timofte, "Rendering natural camera bokeh effect with deep learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 418–419.

[14] D. Im, G. Park, Z. Li, J. Ryu, S. Kang, D. Han, J. Lee, and H.-J. Yoo, "Dspu: A 281.6 mw real-time depth signal processing unit for deep learning-based dense rgb-d data acquisition with depth fusion and 3d bounding box extraction in mobile platforms," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 510–512.

[15] D. Im, G. Park, Z. Li, J. Ryu, and H.-J. Yoo, "Sibia: Signed bit-slice architecture for dense dnn acceleration with slice-level sparsity exploitation," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 69–80.

[16] J.-W. Jang, S. Lee, D. Kim, H. Park, A. S. Ardestani, Y. Choi, C. Kim, Y. Kim, H. Yu, H. Abdel-Aziz, J.-S. Park, H. Lee, D. Lee, M. W. Kim, H. Jung, H. Nam, D. Lim, S. Lee, J.-H. Song, S. Kwon, J. Hassoun, S. Lim, and C. Choi, "Sparsity-aware and re-configurable npu architecture for samsung flagship mobile soc," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 15–28.

[17] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–14.

[18] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[19] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–12.

[20] H.-T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[21] B. Lamba and A. Sharma, "A review paper on different multipliers based on their different performance parameters," in *2018 2nd International Conference on Inventive Systems and Control (ICISC)*. IEEE, 2018, pp. 324–327.

[22] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[23] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 218–220.

[24] C.-H. Lin, C.-C. Cheng, Y.-M. Tsai, S.-J. Hung, Y.-T. Kuo, P. H. Wang, P.-K. Tsung, J.-Y. Hsu, W.-C. Lai, C.-H. Liu *et al.*, "A 3.4-to-13.3 tops/w 3.6 tops dual-core deep-learning accelerator for versatile ai applications in 7nm 5g smartphone soc," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 134–136.

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[26] Z.-G. Liu, P. N. Whatmough, Y. Zhu, and M. Mattina, "S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 573–586.

[27] S. Nair and A. Saraf, "A review paper on comparison of multipliers based on performance parameters," *International Journal of Computer Applications*, vol. 5, no. 4, pp. 6–9, 2014.

[28] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[29] NVIDIA, "NVIDIA H100 Tensor Core GPU Architecture." [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

[30] J.-S. Park, C. Park, S. Kwon, T. Jeon, Y. Kang, H. Lee, D. Lee, J. Kim, H.-S. Kim, Y. Lee *et al.*, "A multi-mode 8k-mac hw-utilization-aware neural processing unit with a unified multi-precision datapath in 4-nm flagship mobile soc," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 189–202, 2022.

[31] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[32] L. Rubinfield, "A proof of the modified booth's algorithm for multiplication," *IEEE Transactions on Computers*, vol. C-24, no. 10, pp. 1014–1015, 1975.

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[34] S. Ryu, H. Kim, W. Yi, and J.-J. Kim, "Bitblade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[35] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 304–317.

[36] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[37] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 567–576.

[38] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on electronic Computers*, no. 1, pp. 14–17, 1964.

[39] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[40] E. Wijmans, S. Datta, O. Maksymets, A. Das, G. Gkioxari, S. Lee, I. Essa, D. Parikh, and D. Batra, "Embodied question answering in photorealistic environments with point cloud perception," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6659–6668.
[41] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.
[42] W.-C. Yeh and C.-W. Jen, "High-speed booth encoded parallel multiplier design," *IEEE transactions on computers*, vol. 49, no. 7, pp. 692–701, 2000.