IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 8, 2022, accepted February 21, 2022, date of publication February 24, 2022, date of current version March 11, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3154014

Modeling and Simulation of System Bus and Memory Collisions in Heterogeneous SoCs

JOOHO WANG[1], YUNGYU GIM[2], SUNGKYUNG PARK[3], (Senior Member, IEEE), AND CHESTER SUNGCHUNG PARK[1], (Senior Member, IEEE)

[1]Department of Electrical and Electronics Engineering, Konkuk University, Seoul 05029, South Korea
[2]System LSI Business, Samsung Electronics, Inc., Hwaseong-si, Gyeonggi-do 18448, South Korea
[3]Department of Electronics Engineering, Pusan National University, Busan 46241, South Korea

Corresponding author: Chester Sungchung Park (chester@konkuk.ac.kr)

This work was supported by the Samsung Research Funding and Incubation Center, Samsung Electronics, under Project SRFC-IT1802-10.

ABSTRACT A system simulator is proposed and developed, which can help to optimize design parameters and hence minimize the number of collisions. In order to search the optimal design parameter combination which meets the user requirement, the proposed simulator has some knobs: partitioning between software and hardware, scheduling the operations in the system, and memory merging, all of which can be adjusted to predict collisions and search the optimal architecture. Also, design parameters can be adjusted sequentially to cover all design options and estimate the predicted performance for each option. The proposed system simulator is evaluated with an example signal processing algorithm, orthogonal matching pursuit (OMP) algorithm. Performances of four cases of the OMP algorithm are predicted by the proposed simulator and in turn are compared with the actual performances on ZedBoard. The proposed simulator can predict the performance of heterogeneous systems on chips with under 5% error for all the candidate architectures for OMP while taking the system bus and memory conflicts into account. Moreover, the optimized heterogeneous SoC architecture for the OMP algorithm improves performance by up to 32% compared with the conventional CAG-based approach. The proposed simulator is verified that the proposed performance estimation algorithm is generally applicable to estimate the performance of any heterogeneous SoC architecture. For example, the estimation error is measured to be no more than 5.9% for the convolutional layers of CNNs and no more than 5.6% for the LDPC-coded MIMO-OFDM. In addition, the optimized heterogeneous SoC architecture improves performance by up to 48% for the third convolutional layer of AlexNet and 56% for the LDPC-coded MIMO-OFDM. Lastly, compared with the conventional simulation-based approaches, the proposed estimation algorithm provides a speedup of one to two orders of magnitudes. The source code is available on the GitHub repository: https://github.com/SDL-KU/HetSoCopt.

INDEX TERMS Collision, conflict, design space exploration, hardware accelerator, heterogeneous SoC, modeling, performance prediction and estimation, system simulator.

I. INTRODUCTION

Demand for heterogeneous systems on chips (SoCs) is increasing these days owing to the impending end of Moore's law and Dennard scaling. A heterogeneous SoC includes more than one hardware accelerator which is dedicated to intensive computation such as a large volume of convolution operations or matrix multiply operations. Hardware accelerators can supplement processors and enhance the overall system performance by adopting parallelism rather than sequential operations [1]. The field programmable gate array (FPGA) [2], the application-specific integrated circuit (ASIC) [3], and the graphics processing unit (GPU) [4] in a broad sense are some examples to implement parallel-processing hardware accelerators. Owing to the superiority of the hardware accelerator in a specific domain over the processor, the number of hardware accelerators is increasing these days [5].

Meanwhile, hardware accelerators need more time for their development and verification and have lower flexibility, compared with software. Moreover, interface circuits are added to connect hardware accelerators to the on-chip interconnect or

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

bus and the paths of data becomes more complex. In turn, leveraging hardware accelerators to enhance the system performance may make the system bus more complicated and increase data collisions in memory and the system bus. Consequently, the actual performance of the overall system can be quite lower than the predicted performance if memory and system bus collisions are not properly taken into account. Thus predicting the collisions in the system bus and memory and obtaining the system performance at early stages are important. For these, a system simulator is needed, which can help to optimize design parameters and hence minimize the number of collisions. To reduce design turnaround time and also explore system level trade-offs of SoCs, design space exploration and performance prediction by using system simulators early in the design flow are indispensable for successful SoC prototyping.

The Zynq platform [6] is widely used to verify many heterogeneous SoC simulators [7]-[11]. The Zynq platform has a dual-core ARM processor tightly coupled with a field-programmable gate array (FPGA). The FPGA can be used as a reconfigurable accelerator to efficiently implement some hardware functionality. The ARM side of the SoC is known as a processing system (PS), whereas the FPGA side is termed programmable logic (PL). When an application is ported to the Zynq SoC, it can run as a pure software implementation on the PS. To gain performance, compute-bound parts of an application can be mapped as hardware accelerators on the PL. For example, [7] predicted the performance of a heterogeneous SoC consisting of a DMA-controlled hardware accelerator, a single processor core, a DRAM subsystem, and on-chip buses using a SystemC TLM-based simulator. The results predicted using the SystemC TLM simulator were compared with those obtained using the Zynq platform. [8] introduced gem5-aladdin, which integrates the gem5 system simulator with the Aladdin accelerator simulator to enable the simulation of SoCs with complex accelerator system interactions. The gem5-aladdin simulator showed that it validated against the Xilinx Zynq platform and achieved less than 6% error. Specifically, the gem5-aladdin simulator measured and utilized the execution time information of the ZedBoard [12] for the CPU (i.e., Cortex-A9) to model the cache line latency. In [9], the requirements for memory, computation, and flexibility of the system were summarized for mapping a CNN on embedded FPGAs. Based on these requirements, they proposed Angel-Eye, a programmable and flexible CNN accelerator architecture simulator, along with a data quantization strategy and compilation tool. The design strategy obtained using the Angel-Eye simulator [9] was implemented on the Zynq platform, and the actual performance gains were evaluated. Next, [10] proposed a performance estimation algorithm to optimize the communication schemes (CSs), which are defined by the number of direct memory access controllers (DMACs) and the bank allocation of DRAM. Using the communication bandwidths of CPs obtained from prior full-system simulations based on the Zynq platform, the proposed performance estimation algorithm can predict the communication performance of CSs more accurately than conventional performance estimation algorithms. In [11], a revised roofline model was proposed to estimate the performance of a DMA-controlled accelerator, considering the impact of DRAM latency. The roofline model proposed in [11] was verified using the Zynq platform.

In this work, a system simulator which can predict the performance of heterogeneous SoCs and model dynamic effects such as collisions is proposed. Static analysis-based techniques typically provide sufficient efficiency in terms of time cost but the accuracy is low. On the other hand, simulation-based techniques may generally provide satisfactory accuracy at the expense of lower efficiency than static analysis based techniques. The proposed simulator can help optimize design parameters and accordingly minimize the number of conflicts or collisions while exploring a large design space. Both speed and accuracy are valued by rapid design space exploration and consideration of dynamic effects such as collisions. To search the optimal design parameter combination which meets the user requirement, the proposed simulator has some knobs (based on the information extracted from the target system): partitioning between software and hardware (or mapping operations into software or hardware), scheduling the operations (or changing the order of operations) in the system, and memory merging (separate memories or shared memory), all of which can be adjusted to predict collisions. Also, design parameters can be adjusted sequentially to cover all design options and estimate the predicted performance for each option. The simulator proposed in this study utilizes the extracted time information (some knobs) for each component such as a direct memory access (DMAC) controller, processor core, and on-chip bus through emulation based on the Zynq platform [6]. When the target application is implemented in the Altera/Intel SoC platform [13], if the time information of the Altera/Intel SoC platform is extracted in pre-measured and applied to the proposed simulator, the performance of the target application implemented in Altera/Intel SoC Platform can be easily predicted.

The proposed simulator in this study is used to seek the optimal condition which meets the system requirement, while using minimal number of hardware accelerators. Pipelining is assumed for the implementation of a given algorithm which is subdivided into multiple steps of operations. The SoC performance is predicted while the allocation of operations to hardware accelerators and processors is changed and the order of operations is adjusted, whereby the types of collisions will vary. Namely, both partitioning and scheduling are considered in the proposed simulator to recommend some design parameter combinations to achieve the optimal system performance. The execution time of each operation step varies as the partitioning varies and the type of data collision varies (and hence the execution time) as the scheduling varies.

The proposed system simulator is evaluated with an example compressive sensing algorithm, orthogonal matching pursuit (OMP), which is subdivided into 5 steps. The OMP algorithm estimates the channel in the LTE wireless standard

with a 5MHz channel bandwidth and multiple hardware accelerators are employed to implement the algorithm. The data type, the size of data, the order of data, the access type (read or write), and timing parameters are the inputs to the simulator. Performances of 4 cases of the OMP algorithm are predicted by the simulator and those are compared with the measured performance in ZedBoard with the Xilinx Zynq 7020 SoC chip [12]. The execution time of the optimal OMP architecture, obtained from the proposed simulator, is 10295 cycles whereas the execution time from the actual implementation on ZedBoard with the Zynq SoC chip is 10788 cycles, leading to a 4.8% error. The proposed system simulator can predict the performance of heterogeneous SoCs with under 5% error while taking the system bus and memory collisions into account. Moreover, the proposed simulator is verified that the proposed performance estimation algorithm is generally applicable to estimate the performance of any heterogeneous SoC architecture. For example, the estimation error is measured to be no more than 5.9% for the convolutional layers of CNNs and no more than 5.6% for the LDPC-coded MIMO-OFDM. Moreover, the optimized heterogeneous SoC architecture improves performance by up to 48% for the third convolutional layer of AlexNet and 56% for the LDPC-coded MIMO-OFDM.

We summarize the novelty of this study as follows:

- We propose a novel system simulator to optimize a heterogeneous SoC, which is defined by the number of hardware accelerators and processors. The novel performance estimation algorithm of the simulator proposed herein evaluates the performance of a heterogeneous SoC architecture, based on the timing information of accelerators and processors, measured on commercial SoC platforms, such as the Zynq 7020 SoC chip on ZedBoard [12] and AccTLMSim [7].
- The performance estimation algorithm of the proposed simulator can search the design space by reflecting all combinations of memory merging, hardware-software partitioning, and scheduling to find the optimal heterogeneous SoC architecture. In addition, the proposed performance estimation algorithm primarily considers the performance impact of both the memory latency and bus protocol overhead on a hardware accelerator.

The paper is organized as follows. Section II reviews some of the related works and summarize our contributions. System modeling preliminaries are accounted for in Section III where the system, the bus, and data transfer types are addressed, followed by transfer delay types which are extensively explained in particular. Section IV deals with the performance estimation process of the performance estimation method in terms of modeling using an example of a signal processing algorithm. Simulation methods, results, and analysis are elaborated on in Section V with case studies. Finally, the conclusion and future work are given in Section VI.

II. RELATED WORK

A multitude of studies have addressed design space exploration and performance estimation of heterogeneous SoC architectures [14]–[18]. Scheduling and timing analysis of on-chip communication, factoring in software parts such as interrupt service routines and device drivers were presented in [14], [15], wherein the buffer resource, bus contentions, and bus sharing were also taken into consideration to enhance accuracy. In [16], a static performance estimation method was used to first reduce a large design space, and then the reduced space was explored using a trace-driven memory simulator to select the optimal on-chip communication architecture. Because memory accesses are involved in bus contention, memory allocation was also considered in [16]. In today's complex heterogeneous embedded SoC design, it is imperative to model and simulate the system of interest at a high level. A methodology and framework to perform this with analytical modeling and multi-objective optimization were presented in [17], wherein candidate architectures were explored at different levels of abstraction. Dataflow graphs in [17] represented the transformation of coarse-grained events in an application into fine-grained events in the architecture. A system-level modeling and simulation environment was specified in [18], wherein pruning of the design space and calibration of timing parameters by coupling with low-level simulators or prototype implementations were discussed.

In our work on heterogeneous SoC modeling and simulation, we assumed a bus compliant with the AMBA AXI bus protocol [19]. In [10] and [20], a SystemC TLM-based simulator was used to evaluate and predict the performance of a hardware accelerator implemented with an AXI bus-based SoC architecture based on memory access modeling, including the memory access type, burst type, and access latency. In [21], the design space for the AMBA hierarchical shared bus was explored, and the execution time of each bus architecture was estimated by pipelining and accounting for burst transactions. The execution order and amount of transfer data were also considered during design space exploration. However, dynamic effects such as scheduling and hardware-software partitioning at the memory and system bus were not taken into account in [10], [20], [21].

A performance estimation method based on a tree model was presented in [22] to determine the relationship between SoC parameters and performance, as well as to improve the prediction accuracy during design space exploration, using parameter ranking to guide the design. A high-level analytical tool to evaluate the communication and computation overheads of heterogeneous systems was presented in [23], wherein dataflow and loop pipelining within the high-level synthesis framework were considered with the assumption of an AXI4 stream protocol. In [23], dynamic data dependencies were considered, and the communication and computation times were estimated. A design space exploration method was introduced in [24] with task scheduling, which is based on a traffic-aware priority-based earliest-finish-time algorithm,

to achieve a high level of core utilization. Guidance on how to offer a well-designed scratchpad on-chip memory system in terms of memory capacity and number of memory banks was provided in [24]. There have been several reports on the optimization of the bank allocation of DRAM [25], [26] in a heterogeneous SoC architecture. Most proposed bank allocations focus on balancing interference mitigation and bank-level parallelism. For example, in [25], DRAM banks were dynamically partitioned according to application profiling (e.g., memory-intensive vs. non-intensive). In [26], locality-aware bank partitioning (LABP) was proposed to improve dynamic bank partitioning by mitigating the inference caused by non-intensive applications, for example, by separating their banks from that of memory-intensive high-row buffer locality applications. However, the proposed bank allocations are targeted at general-purpose processors (e.g., CPUs) running different applications. Specifically, optimization of bank allocation for application-specific hardware accelerators has not been considered.

When it comes to the estimation of communication performance, there are two different approaches: static analysis-based approach and simulation-based approach. The static analysis-based approach tends to be faster than the simulation-based approach, but the estimation accuracy may not be sufficiently high to drive the design of communication architectures. In particular, the static analysis-based approach cannot accurately capture the dynamic nature of communication bandwidth. To improve accuracy, a static analysis-based approach is often combined with a set of traces extracted from simulations. In [14], a hybrid trace-based performance estimation algorithm was proposed to estimate the performance of bus-based communication architectures using a communication analysis graph (CAG). In [27], a simulation-based performance estimation was proposed based on the observation of actual traffic tracking of the application of each core (e.g., bus master). In [28], in order to estimate the performance of bus-based communication architectures, a static analysis based on a modified queueing model was incorporated into the schedule-aware performance estimation. [29] proposes to estimate the memory latency based on the statistics of different access conditions. In [30], bus-based on-chip communication architectures were explored early in the design flow using a modeling abstraction, called the cycle count accurate at transaction boundaries. The abstraction used in [30] can speed up the simulation by a factor of 2 when compared with the bus cycle accurate abstractions. However, in [27]–[30], neither the dynamic effects such as collisions or conflicts at the memory and the system bus, nor the operation scheduling was considered.

However, the simulation-based approach is sufficiently accurate to capture the dynamic communication bandwidth. Thus, most reports on bank allocation rely on performance evaluation using full-system simulators in conjunction with DRAM simulators. For example, in [25], the proposed bank allocations were evaluated using gem5 in conjunction with open DRAM simulator. In [7], a full-system simulator modeled the communication bandwidth of a DMA-controlled accelerator from/to the DRAM subsystem through an on-chip bus on a transaction level. However, a simulation-based approach may often be too time-consuming to explore a broad design space for communication architectures.

We summarize the contributions of this study as follows:

- In order to improve the performance of the heterogeneous SoC architecture, compared with that of a CAG-based optimization [14], which considers only memory merging, the proposed simulator considers both hardware-software partitioning and scheduling. The optimal combination of design options suggested by the proposed simulator can improve the performance of a heterogeneous SoC architecture by up to 32
- Compared with the conventional simulation-based [7] approach, the proposed performance-estimation algorithm provides a speedup of two orders of magnitude. The simulation-based approach is sufficiently accurate to capture the dynamic nature of the communication bandwidth, but it often takes a prohibitively long time to simulate. According to our experiments, the conventional simulation-based approach takes at least a few hours to evaluate a single heterogeneous SoC architecture with hundreds of different combinations of hardware-software partitioning, memory merging, and scheduling. Using the conventional simulation-based approach, we run the full-system simulator proposed in [7], once for each design point, taking approximately 12.5 seconds per design point. By contrast, to minimize the simulation time and maintain estimation accuracy, the proposed performance estimation algorithm constrains the use of an evaluation board (e.g., ZedBoard [12]) to evaluate the time information of the heterogeneous SoC architecture. Because a few tens of time information are sufficient to express most of the heterogeneous SoC architectures of interest, the extra simulation time required to obtain the time information of each hardware component becomes negligible, particularly in the case of broad design space (i.e., a space of hundreds of design points).
- Compared with static analysis-based bus performance estimation [28] and statistics-based estimation [29], the proposed performance estimation algorithm can predict the communication performance of heterogeneous SoC architectures more accurately because it considers both the bus protocol overhead (bus conflict) and memory latency (memory collision) using an evaluation board (e.g., ZedBoard [12]) to evaluate the time information of the hardware and software. The experimental results show that the proposed algorithm approaches the full-system simulator [7] more closely than the conventional algorithms. For example, the proposed algorithm reduced the estimation error to 6%, whereas the conventional algorithms in [28] and [29] experienced estimation errors of 18% and 16%, respectively.

III. BASICS OF SYSTEM MODELING

In this section, modeling of systems consisting of masters, buses, and slaves is described along with operating principles and data transfer types. Based on these and performance estimation methods which will be explained in Section IV, the system simulator is developed and simulation results are obtained to predict the system performance.

A. SYSTEM UNDER CONSIDERATION

A basic system is shown in Figure 1, where a master is a processor or a direct memory access (DMA) controller and a slave is a memory or a hardware accelerator which can run operations. The number of masters and slaves is allowed to vary and a single bus is assumed between masters and slaves.

The slave is organized as first in first out (FIFO) and is subdivided into a hardware (HW) block and an AXI slave wrapper (assuming the bus complies with the AMBA AXI protocol [19]), as shown in Figure 2. The hardware block takes charge of operations or computation. The AXI slave wrapper enables operations, synchronizes input and output, and supports multiple outstanding transactions. The AXI protocol uses valid and ready handshake signals to confirm the data transfer is completed. The enable signal is generated by using valid and ready signals. A counter has a pre-determined number of input data, which triggers the operation, and a predetermined number of output data, which is produced as an outcome of the operation. The counter conducts sync between input and output such that the operation gets started only after a preset number of input data is received and the operation produces a preset number of output data after a delay. The values set up in the counter can be adjusted in. For example, when it comes to the target application implemented in the heterogeneous SoC architecture is LDPC-coded MIMO-OFDM, the amount of communication and computation required for each hardware accelerator differs based on the number of spatial streams (NSS) and the modulation coding scheme (MCS). In this case, during the initial design stage of each hardware accelerator, the maximum value of the counter is set differently so that it can respond to the change in the communication amount according to the determined NSS and MCS. The counter in Figure 2 confirms the input by using WVALID and WREADY signals in AMBA AXI protocol [19]. If the counter reaches the state that the slave is able to produce the output of the operation, the RVALID signal is asserted. The counter confirms the output data transfer by using the RREADY signal. If data are not generated in time and a read delay occurs, WREADY is deasserted to prevent new data from entering the slave. The read delay case and the write delay case are illustrated in Figure 3, where the input data are assumed to be output in the next clock cycle. In the normal case when no delay occurs, the data come in through the write channel and the outcome of the operation goes out through the read channel. In the write delay case in which an input delay occurs (by deasserted WVALID from the master), RVALID is deasserted after the current output of the operation goes out (to prevent further

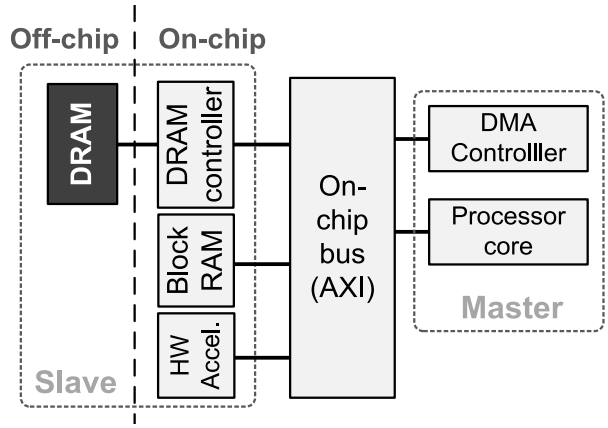

FIGURE 1. Basic system structure.

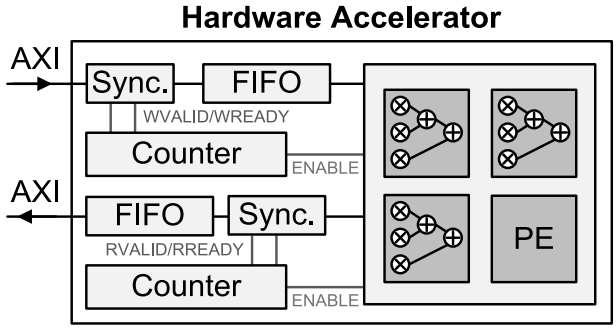

FIGURE 2. Slave (hardware accelerator) structure.

data from leaving). In the read delay case when the master is not ready to send data at the right moment and thus RREADY is deasserted, WREADY is deasserted to immediately prevent data from entering the slave.

According to the number of address channels and the number of data channels, the bus is categorized into shared address bus and shared data bus (SASD), shared address bus and multiple data buses (SAMD), and multiple address buses and multiple data buses (MAMD) [31], [32]. The SASD mode has one address channel and one data channel, where one master is unable to occupy the bus while another occupies it. The MAMD mode has multiple address channels and multiple data channels, where even if one master occupies the bus, another master is able to access a slave that is not under collision. However, it needs a large amount of resources and its design gets readily complicated. Figure 4 and Figure 5 show the SAMD mode which combines the benefits of the SASD and MAMD modes. In Figure 4, one address bus is shared by all the masters while in Figure 5, each master has a separate data bus so that simultaneous data transfers are possible. For the AXI protocol, one address is needed to send a burst and hence the required bandwidth on the address channel is lower than that on the data channel. Thus the SAMD mode typically exhibits similar performances as the MAMD mode but in a particular case of one master occupying the address channel, another may not access the channel.

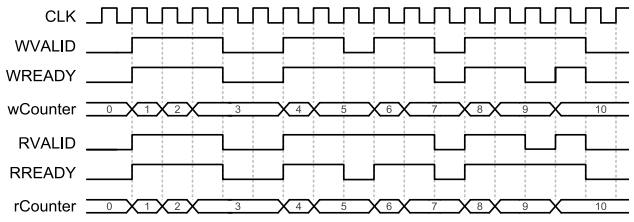

FIGURE 3. Counter operations.

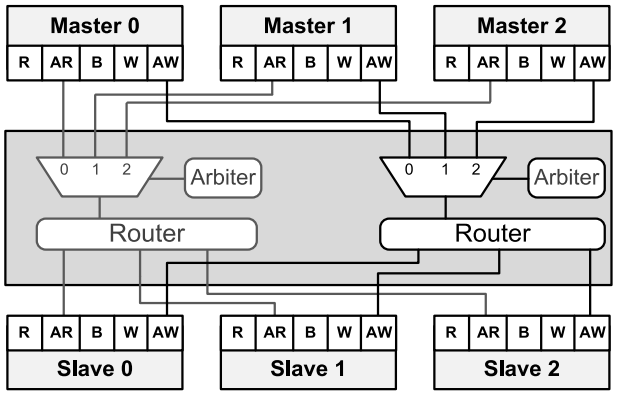

FIGURE 4. SAMD bus: address channel.

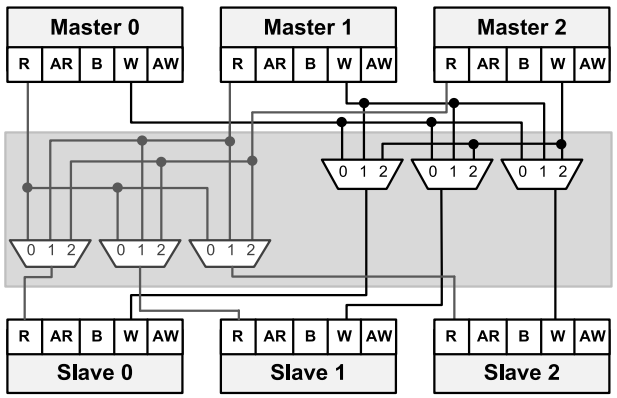

FIGURE 5. SAMD bus: data channels.

B. DATA TRANSFER MODELING IN BUS ARCHITECTURE

Figure 6 shows the process in which a processor reads data from memory, internally executes operations on them, and writes the outcome to another memory. Since the processor is able to read and write data of its own accord, it can directly access memory. The data transfer through the bus or on-chip interconnect is seen in the lower part of Figure 6 where after some amount of time elapses with the read access, the write access is initiated, yielding an overlap between the read and the write. The data transfer by the processor is seemingly clearer and simpler than that by the hardware accelerator but more time will be consumed to transfer an item of data and also the efficiency will be lower since one processor deals with all the operations internally.

Figure 7 shows the process in which a hardware accelerator is involved in the data transfer. The hardware accelerator

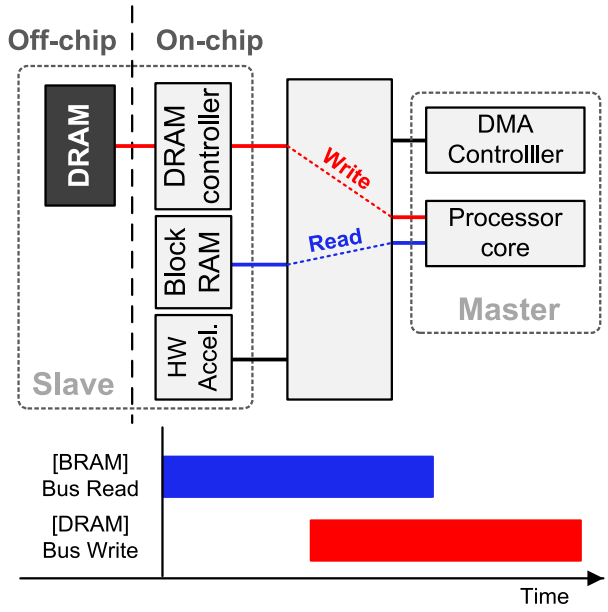

FIGURE 6. Data transfer associated with a processor core.

needs help from the DMA to transfer data. The DMA is unable to begin its activity of itself but entails control by a processor. Thus in the beginning, the processor first configures the DMA with the information needed to initially setup the data transfer (DMA set in Figure 7). Subsequently, DMA 0 which is in charge of the hardware accelerator input begins its action while reading data from memory 0. After some time passes by and the hardware accelerator begins to output the result, DMA 1 reads out from the hardware accelerator and writes in into memory 1. The data associated with this are shown as the latter read (blue) and the latter write (red) in Figure 7, where the two reads and the two writes overlap with one another. By comparison with the data transfer maneuvered by a processor, the number of data movements is doubled (as shown in the lower part of Figure 7) whereas the data transfer time is shorter owing to the higher efficiency with DMAs in general. However, the use of DMA does not always guarantee a better performance because of the doubled number of reads and writes and the overhead of the DMA set.

Two types of collisions, memory collision and SAMD collision, will be addressed, which will incur delay. Figure 8 shows the situation that multiple masters try to access memory through a bus with an arbiter. This leads to a collision at memory if only one physical port exists in memory. Only one master is allowed to access memory at a time owing to the collision and other masters than the master granted access will all suffer some delay. The bus arbiter grants access to a master according to the priority among the masters and accordingly the read or write request is processed in sequence. If the request underway is over, then the top priority master is granted access after prioritizing the masters again.

Memory collisions may be subdivided into collisions by two DMAs, by two processors, and by a DMA and a

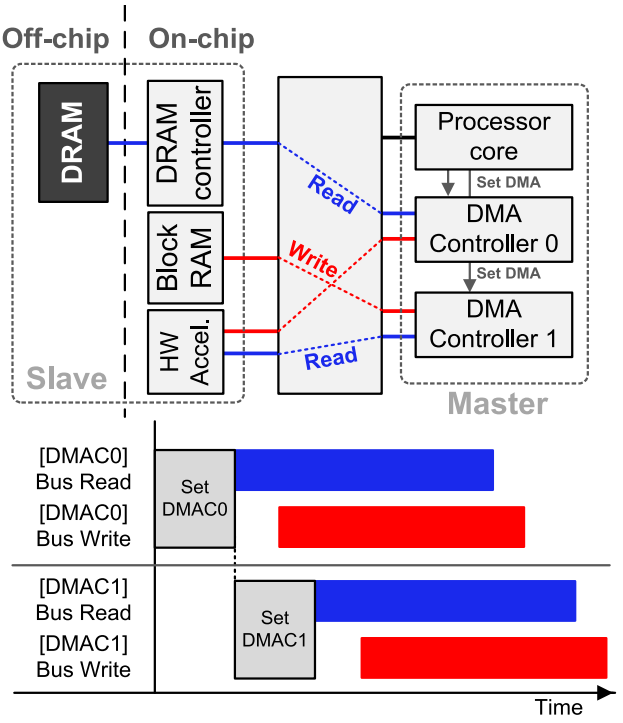

FIGURE 7. Data transfer associated with a hardware accelerator.

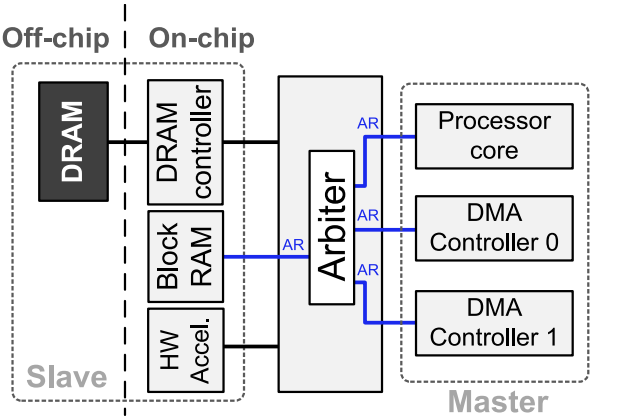

FIGURE 8. Memory collision: address channel.

processor. Assume that the time consumed on the bus is ignored and the MAMD mode is considered for the sake of examining only the memory collision situation. Also assume that the DMA uses bursts with burst length 4 and supports multiple outstanding transactions (i.e., a new transfer request is allowed before the currently ongoing transfer is terminated) whereas memory does not support multiple outstanding transactions. The processor is assumed to support neither burst transfers nor multiple outstanding transactions. Figure 9 shows two situations: two masters read from two memories (left) and two masters read from one memory, leading to collision (right). The master can be a processor or a DMA and hence three cases are considered with the two masters in Figure 9: the case with two DMAs, the case

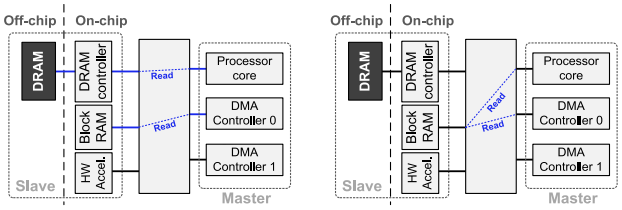

FIGURE 9. System configurations to identify memory collisions: no collision (left) and collision (right).

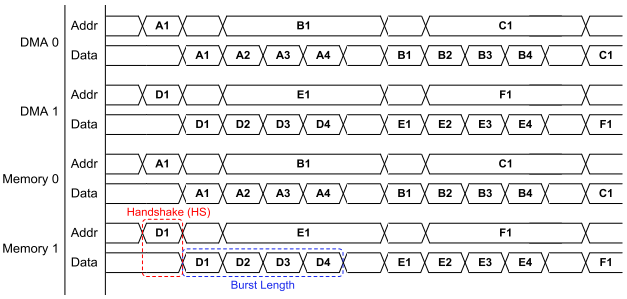

FIGURE 10. DMA Transaction with no collision.

with two processors, and the case with one processor and one DMA.

Figure 10 shows the case with two DMAs and two memories, which corresponds to the left half of Figure 9. DMA 0 and DMA 1 read from memory 0 and memory 1 by delivering addresses A1 and D1, respectively. No collision occurs since different memories are accessed and addresses A1 and D1 are delivered without delay such that memory 0 and memory 1, respectively, dispatch data burst with length 4 without delay. Owing to the allowed multiple outstanding transactions, DMAs send addresses B1 and E1 before their first data transfers are finished. However, each memory receives its next address after its currently ongoing data transfer is completed. Namely, addresses B1 and E1 are not immediately input to memories. On the other hand, the DMA data channel can transfer at almost every interval and hence the delay in the address does not impact the actual data transfer, incurring no data transfer delay in effect.

Figure 11 shows the case with two DMAs and one memory, corresponding to the right half of Figure 9, assuming DMA 0 has a higher priority. DMA 0 and DMA 1 deliver read addresses A1 and D1 to memory 0. Since the MAMD bus is unable to deliver two addresses simultaneously memory 0, address A1 of the higher priority DMA 0 is first delivered. Subsequently, DMA 0 tries to send address B1 while memory 0 dispatches data corresponding to address A1. The first address D1 of DMA 1 is delayed at memory 0 until the data transfer of DMA 0 corresponding to A1 is terminated, which is different from the situation illustrated in Figure 10. After the data transfer of DMA 1 corresponding to D1 starts, the opposite situation occurs: the second address B1 of DMA 0 is delayed at memory 0 until the data transfer of DMA 1 corresponding to D1 is finished. To sum up, memory collisions alternately incur transfer delays on DMA 0 and DMA 1 caused by each other, thus entailing more time to

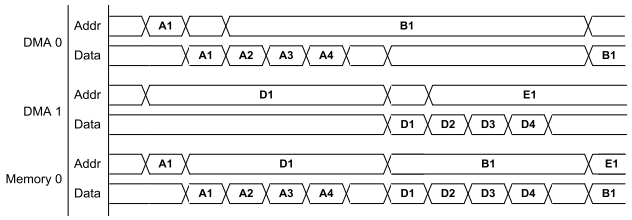

FIGURE 11. DMA Transaction with memory collision.

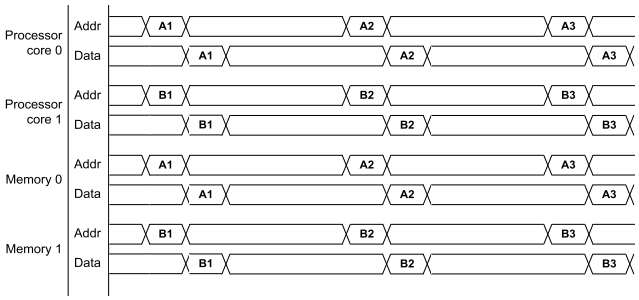

FIGURE 12. Processor core transactions with no collision.

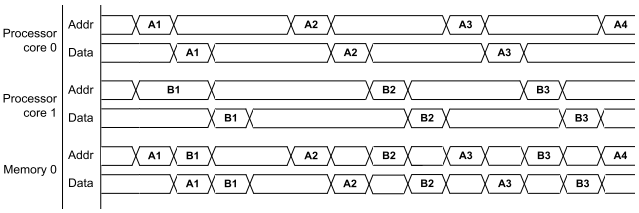

FIGURE 13. Processor core transactions with memory collision.

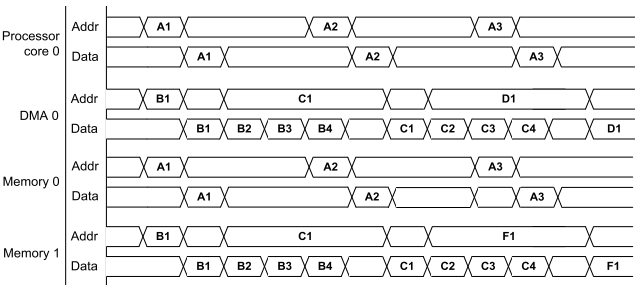

FIGURE 14. Processor core and DMA transactions in combination: no collision case.

finish memory accesses when compared with no collision (in Figure 10).

Figure 12 shows the case where two processors or PS individually read from one of the two different memories, corresponding to the left half of Figure 9. In the absence of both burst transfers and the multiple outstanding transactions, each PS transfers only a single item of data corresponding to an address and also the next address can be sent only after the current data transfer is terminated. PS 0 and PS 1 send addresses A1 and B1 to memory 0 and memory 1, respectively, to execute two reads. No collision occurs since different memories are accessed, no delay is assumed to occur on the bus, and memories dispatch data for A1 and B1 right away.

Figure 13 shows the case with two processors and one memory, the collision case, corresponding to the right half of Figure 9. PS 0 and PS 1 simultaneously send addresses

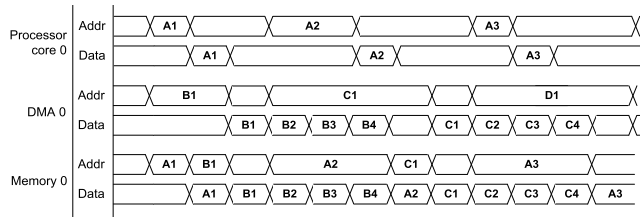

FIGURE 15. Processor core and DMA transactions in combination: memory collision case.

A1 and B1, respectively, to memory 0. Owing to the collision at memory, the MAMD bus arbiter plays a part. The higher priority PS, PS 0, first delivers A1 and receives the corresponding data. B1 of PS1 is delayed until the ongoing data transfer of PS 0 corresponding to A1 is completed. However, this delay suffered by the lower priority PS is different than that associated with the DMA. The delay associated with the processor is temporary, namely, the delay occurs only once in the beginning and does not cumulate with time. Here assume that the two processors run at the same speed and hence the read interval is the same. As a consequence, memory collisions between processor and processor will not be factored in during the performance estimation process which will explained in Section IV.

Figure 14 and Figure 15 show the cases with one processor and one DMA in combination, where no collision occurs and memory collisions occur, respectively. In Figure 14, data are read at the correct timing without any transfer delay, assuming an ideal bus. In Figure 15, by contrast, a DMA and a PS read from a memory simultaneously, leading to memory collisions. More specifically, PS 0 and DMA 0 send addresses A1 and B1 to memory 0, respectively. A collision is detected in the bus arbiter. According to the assumed priority, A1 of the higher priority PS 0 is first delivered to memory 0, whereby B1 of DMA 0 is delayed until the read of PS 0 corresponding to A1 is finished. In the middle of the read of DMA 0 corresponding to B1, PS 0 requests a new read corresponding to A2, which cannot be processed in memory 0 and thus is delayed until the read of DMA 0 is terminated. In this manner, delay is incurred alternately henceforth. The delay inflicted on DMA 0 is caused by a data transfer of PS 0 inserted between two separate DMA transfers which are based on bursts with the maximum burst length of 256. Therefore, the inserted transfer delay of PS 0 is typically negligible, compared with the burst length. On the other hand, the delay inflicted on PS 0 is as long as the burst length of DMA 0 and hence seems to be in sync with the burst transfer. This data transfer behavior is also identified in the SAMD collision which will be depicted shortly. The delay that is incurred in the SAMD mode also occurs when different memories are accessed. Thus the delay inflicted on PS 0 will be factored in when we deal with the SAMD collision shortly.

The AXI bus may be subdivided into the MAMD bus, the SAMD bus, and the SASD bus. The SAMD bus operates while having no practically noticeable performance difference from the MAMD bus, except for the multiple

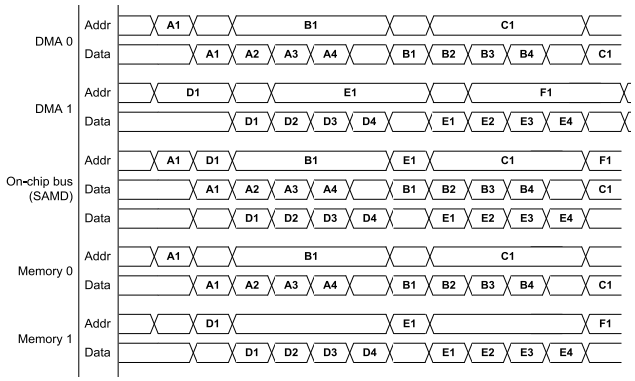

FIGURE 16. SAMD collision between DMAs.

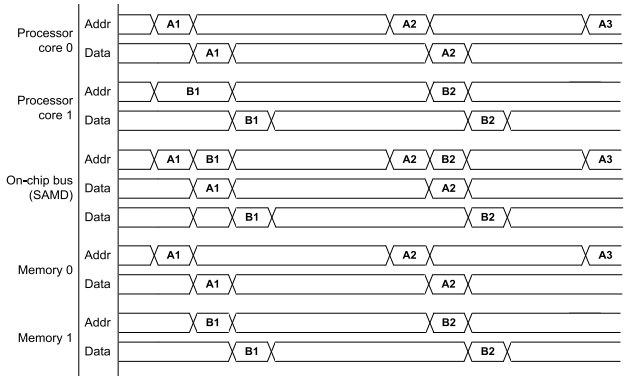

FIGURE 17. SAMD collision between processor cores.

outstanding transaction situation. The SAMD collision may occur between masters which operate in conjunction with one SAMD bus. It arises when the number of multiple outstanding transactions supported by the master is larger than that supported by the slave. Assume that the DMA supports up to 2 multiple outstanding transactions and its burst length is 64. Also assume that the processor supports neither multiple outstanding transactions nor burst transfers. Slave is also assumed to not support multiple outstanding transactions. Figure 16 shows two DMAs access their individual slaves through the SAMD bus. Namely, DMA 0 and DMA 1 send addresses A1 and D1 at once to hardware slaves HW 0 and HW 1, respectively. Since only one address channel exists in the bus, A1 of DMA 0 is first delivered to HW 0, according to the priority. Before the current data transfer is completed, the next address can be sent to the hardware since multiple outstanding transactions are supported. DMA 0 and DMA 1 send addresses B1 and E1, respectively, to the bus which then tries to deliver the address that has arrived earlier of the two, B1, to HW 0. However, as HW 0 is under data transfer, the request is not acceptable and hence B1 occupies the bus till the transfer is over, by which E1 of DMA 1 is delayed. Because the DMA transfer is in units of bursts, though the address is delayed, the current data transfer is ongoing and the next data will not be delayed. In brief, the SAMD collision between DMAs is typically on a negligible level.

Figure 17 shows the situation when two processors access their respective slaves via the SAMD bus. The processor,

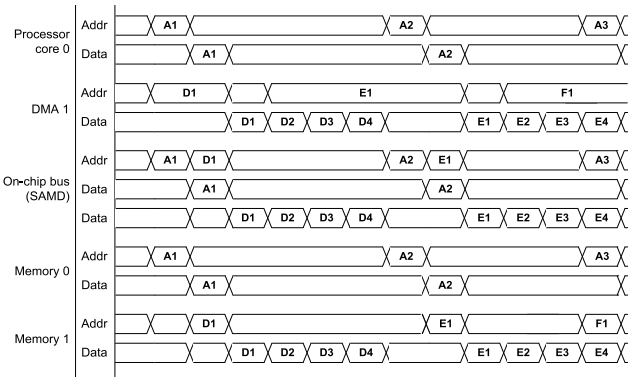

FIGURE 18. SAMD collision between processor core and DMA.

dissimilar to the DMA that supports multiple outstanding transactions, sequentially executes instructions to fulfill individual reads and writes such that it does not occupy an address channel for a long time. As a consequence, the SAMD collision does not show up except for the case when simultaneous bus requests occur. Moreover, the collision occurs only once in the beginning and hence impacts negligibly on the overall execution time.

Figure 18 shows the situation when collisions occur between DMA and processor during transactions through the SAMD bus. The processor, PS 0, and DMA 0 request addresses A1 and B1, respectively, to read data. PS 0 having the higher priority first occupies the bus and transfers its address to HW 0, followed by the address of DMA 0. DMA 0 sends C1 corresponding to the second burst before the first burst is finished but C1 is not to be processed in the slave and thus occupies the address channel of the bus. After this, PS 0 tries to send the new address A2 to HW 0 but A2 has to wait since the shared address channel of the bus is already occupied. When the first DMA burst is finished and thus HW 1 can afford to process new data, then HW 1 receives C1 that has occupied the address channel of the bus and finally A2 of PS 0 pending goes up on the bus. As above, the DMA transfer typically does not suffer a delay whereas the processor transfer is subject to the delay that is proportional to the DMA burst length since the transfer of the processor goes on in sync with the burst transfer of the DMA.

IV. PERFORMANCE ESTIMATION METHODS

Performance estimation methods in this work are explained in terms of partitioning, scheduling, and memory merging on the transaction level. The modeling for performance estimation is depicted in Subsection A and the process of performance estimation is described in Subsections B and C.

A. MODELING

First, some assumptions are made to model a digital system based on, say, a signal processing algorithm. A system is subdivided into individual operation units or functional units which are executed in a block-interlacing manner [33], as illustrated in Figure 19. After the S1 operation is finished

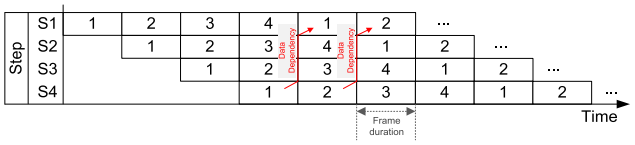

FIGURE 19. Block-interlacing.

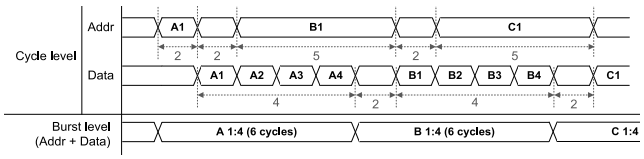

FIGURE 20. Data transfer by the burst.

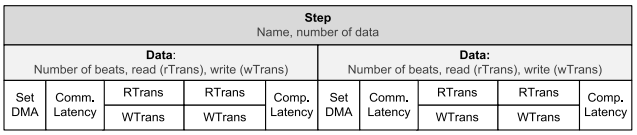

FIGURE 21. Transaction-level model structure of the operation unit or functional unit.

with data 1, its output data are delivered for the S2 operation and at the same time new data 2 are fed for the S1 operation. In this manner, all the operations from S1 to S4 do their works with their individual data at every moment. No data dependency exists between operation units within a frame.

The data movement arising in each operation unit is expressed in units of bursts, as shown in Figure 20. The burst length in the processor is 1 and that in the DMA is predetermined. The data to be transferred are matched to the burst length: if more data are to be transferred than the burst length, appropriate number of bursts are created and the remnant data shorter than the burst length is carried by an extra burst. In this way, the conventional data transfer time by the cycle is represented by the burst in this work for performance estimation, whereby the overall simulation time is reduced. The time required to send a single data beat is multiplied by the number of beats in a burst to obtain the burst-unit data transfer time.

It is assumed that up to two processors are employed in our systems during performance estimation but by necessity more than two processors can be employed. The two processors are allowed to execute their respective operations at the same time. Alternatively, one processor is allowed to perform an operation while the other is engaged in the DMA control for hardware utilization.

System modeling is based on the elements that influence the data transfer when hardware accelerators are included in the system. The memory collision when two bus masters access one memory and the SAMD collision when masters access the shared address channel in the bus are taken into account in the modeling. Furthermore, in the context of collisions, the optimal combination of partitioning, scheduling, and memory merging is sought for.

One thing to consider in an algorithm for system optimization is whether each operation is executed by the processor or by the hardware accelerator. This is called partitioning. To employ the hardware accelerator, the DMA is typically utilized to put data into the hardware whereas the DMA setting prior to the data transfer is taken care of by the processor. In this manner, the data computation and transfer can be accelerated with a higher speed than in the case of the processor alone, apart from the overhead of DMA setting. All the possibilities as regards partitioning are covered in this work, namely, from all operations carried out by processors to all operations conducted in hardware.

The order or sequence of operations impacts the execution time of the given system and therefore the optimization process in view of the so-called scheduling is addressed in this work. Each operation will necessitate its relevant data and depending on whether the processor or the hardware is employed for the operation, the execution time will vary. Depending on the scheduling or the operation sequence, memory collision or SAMD collision may occur owing to interaction between different operations in the sequence. The optimal scheduling or operation sequence exhibiting the shortest execution time is sought after in this work in view of collisions or conflicts between operations.

Memory merging is considered in this work for optimization. Generally, each data item has its lifetime during which it should not be overwritten or altered in memory but be retained. Accordingly, each data item needs its own memory region during its lifetime. On the other hand, to reduce the number of memory blocks being used, a memory region may be allocated to two different data items. In this case, if two different operations read their data items located in the same memory region, a collision occurs in this region, retarding the execution and potentially affecting the system performance. If the effect on the execution time is negligible and the gain of fewer memory blocks is large, memory merging may be adopted to reduce system resources. Memory merging may be considered partly or overall, depending on the predicted performance.

Figure 21 shows the transaction-level model of the operation unit considered in an algorithm. The basic unit of partitioning is the operation unit or functional unit. The C struct, step, is at the top level, where the information including data parameters, time parameters, and DMA parameters are set as user inputs. The struct, data, is to model the input data and the output data in each operation unit. At the bottom level, read and write transactions, *Rtransaction* and *Wtransaction*, are used to model data transfers according to the processor and hardware characteristics. Besides, the DMA setting time and the DMA FIFO latency are modeled in case of the hardware use and the computation time is modeled as well. Each step, data, and transaction has its start time and length information, e.g., the number of beats.

B. PERFORMANCE ESTIMATION ALGORITHM

The performance estimation process proposed in this work is shown in Algorithm 1. Initially, user inputs and the schedule are set up, followed by description of constraints of hardware accelerators and processors together with depiction of

Algorithm 1 Performance Estimation

Input: NoPart, NoStep, NoSched, NoFunc and MemMerge
Output: T

```
for p = 1:NoPart do
    for s = 1:NoSched do
        for m = 1:MemMerge do
            idealArch = heterogeneousSoC (p, s, m);
            swConstArch = swCosntraints (idealArch);
            hwConstArch = hwCosntraints (swConstArch);
            busConstArch = busConstraints (hwConstArch);
            T = performance(busConstArch);
        end for
    end for
end for
```

memory and bus conflicts or collisions. Algorithm 1 proposed in this study can predict the performance of a heterogeneous SoC architecture. This algorithm considers hardware-software partitioning, memory merging, and scheduling in each loop, to determine the optimal design combination when implementing a heterogeneous SoC using a hardware accelerator.

First, the partitioning loop determines that the primary thing to consider in an algorithm for system optimization is whether each operation is executed by the processor or by the hardware accelerator. Through this loop, it is possible to consider the cases where all functions of the target application to be implemented in a heterogeneous SoC are mapped with the software domain, and the cases where all functions are mapped with the hardware domain. The total number of partitioning (*NoPart*) considered in the proposed estimation algorithm is determined by the power of 2 (i.e., 2^{NoStep}). Next, each function (i.e., *step*) of the target applications determines the scheduling in which each function mapped to the hardware domain or the software domain is executed. This loop considers the effects of memory and bus conflicts that occur when each function shares one memory simultaneously. In addition, it can be considered that the execution time of this loop differs depending on the domain in which each function is to be performed. The total number of scheduling, considered in the proposed estimation algorithm, is determined by a factorial (i.e., *NoFunc!*) of the number of functions. Finally, a memory merging step is performed to predict the performance of the heterogeneous SoC architectures according to the number of on-chip memories. In general, on-chip memory collisions degrade the performance of accelerators implemented with heterogeneous SoC architectures. In other words, if the variables of different functions do not use the same on-chip memory, the performance of the accelerator implemented with a heterogeneous SoC architecture can be improved. However, using a large amount of on-chip memory can increase the hardware complexity of the heterogeneous SoC architecture that needs to be implemented. Therefore, in this loop, we merge on-chip memory to lower the complexity of the heterogeneous SoC architecture while performing memory merging to gain almost the same performance improvement as having an on-chip memory for each function of the target application. We assume three situations in this loop: 1) If all functions share the same memory (*m=0*). 2) If all functions share two memories (*m=1*). 3) if all functions use different memories (*m=2*).

When design options are determined by the index of the three loops, an ideal heterogeneous SoC architecture (*idealArch*) that does not reflect memory collisions and bus collisions is returned by a predefined function. The returned ideal heterogeneous SoC architecture (*idealArch*) is used as an input to predefined functions (*swConstraints*) to reflect the constructs related to the software and hardware domains. This function returns the architecture (*swConstArch*), wherein time information (e.g., memory load/store time information) is updated to the processor core in a heterogeneous SoC architecture. The heterogeneous SoC architecture, which reflects software domain time information, is used as an input for predefined functions to reflect constructions related to hardware domains. This function returns the architecture (*hwConstArch*), which updates information regarding the hardware accelerator to be designed in the heterogeneous SoC architecture (e.g., burst length and computation time). Next, in the heterogeneous SoC architecture (*hwConstArch*), which is an output from the hardware domain function (*hwConstraints*), the time information of memory collisions and bus collisions is not reflected. Therefore, it uses the collision modeling function (*busConstraints*), written in Algorithm 2, to update the heterogeneous SoC architecture (*busConstArch*) latency due to memory collision and bus collision. Finally, the performance (T) of the heterogeneous SoC architecture, reflecting all the constraints, is returned by the predefined perf function. The performance estimation algorithm proposed simulates all design points for a heterogeneous SoC architecture that can be obtained through each design option of hardware-software partitioning, memory merging, and scheduling. Among the simulation results of all heterogeneous SoC architectures obtained through the combination of design options, the

Algorithm 2 Bus Constraints

Input: hwConstArch
Output: busConstArch

```
for p = 1:NoStep-1 do
    if hwConstArch.master[s] = DMA && hwConstArch.master[s+1] = DMA then
        if hwConstArch.master[s].memory == hwConstArch.master[s+1].memory then
            busConstArch = memoryConflicts (hwConstArch, s, s+1);
        else
            busConstArch = samdConflicts (hwConstArch, s, s+1);
        end if
    else
        busConstArch = samdConflicts (hwConstArch, s, s+1);
    end if
end for
```

optimum design point is determined to have the minimum execution time and minimum hardware complexity. The execution time of many design points can be quickly estimated according to the combination of design options because the time information of each component of each heterogeneous SoC is obtained through emulation in the target platform (e.g., ZedBoard [12]). Based on this information, it is possible to calculate the execution time of combinations for all possible heterogeneous SoC architectures.

In summary, the time information of each domain (hardware and software) is reflected in an ideal heterogeneous SoC architecture, to which three design options are applied. Additionally, the latency due to bus and memory collisions is reflected in the heterogeneous SoC architecture. We measured the performance of a heterogeneous SoC architecture that reflects all these factors.

C. CASE STUDY: ORTHOGONAL MATCHING PURSUIT (OMP) ALGORITHM

The design space is explored over all partitioning and scheduling methods. A detailed explanation is given with an example application, orthogonal matching pursuit (OMP) [34], [35], as follows. Figure 22 shows user inputs (right) of the OMP algorithm (left) (based on least squares to iteratively recover sparse data), which is the initial phase to obtain the necessary data preceding the simulation run. The OMP algorithm will be revisited in Section V. To implement a digital system architecture from an algorithm, parameters are needed at the initial phase: To predict the overall execution time, basic information and the required time of each operation in the system should be fed as user inputs. Namely, the number of operations (the number of computations), the number of input and output data used in each operation, the order of input and output data (in/out, timing), and information on times (time parameters) such as the operation time consumed inside the hardware accelerator, the required time for the processor to transfer a data item, and the required time for the hardware to transfer a data item.

Next, schedule setting is conducted in Algorithm 1. Based on the number of operations, all the sequences of operations are explored and generated beforehand and called afterward to predict each performance. In case of a system with, for instance, three operations, six distinct sequences exist.

Subsequently, the constraint on hardware accelerators is set in Algorithm 1. The number of hardware accelerators is a representative constraint. Among the operations constituting the given system, any operation(s) may be chosen to be executed by the hardware accelerator(s). In this phase, other conditions such as conflicts (or collisions) are not yet set. The number of processors needed to set DMAs for hardware accelerators is assumed to be infinite and hence all the operations are potentially allowed get started at once. Also, no bus or memory conflict is assumed to exist. Based on these assumptions and the time information from the user inputs, the execution time of each operation is predicted, which is not accurate yet. As conditions on processors in conjunction with bus and memory conflicts are taken into consideration, the system performance will get more accurate.

In succession, the constraint on processors is set in Algorithm 1. The number of processors in the system is the constraint. Processors are used for both the operations (computation) and the hardware accelerators (DMA setting). The number of operations that can be started concurrently is determined by the number of processors. On the basis of the sequence or order of operations, only after the current operation by a processor is terminated, the next operation by the processor is allowed to get started. The limited number of processors renders the system simulator more practical.

Then, memory and bus conflicts (or collisions) are considered in Algorithm 1. In this phase, which operations are affected and how large the operations are affected by each conflict can be predicted. Owing to the use of the SAMD bus, a conflict on the address channel may occur between, e.g., processor and DMA, yielding some data transfer delay in the processor. This delay may cause another (equal-length) delay of the following operation executed by the same processor. As a result, the lastly ending operation in the system may change from one to another and the frame execution time in block interlacing may become longer.

Orthogonal matchip pursuit (OMP) algorithm

Input: y, A
Output: $x^{[i]}$

1: **for** i = 1; i = i+1 till || $r^{[i]}$|| = 0 ..... [S5]
2: $g^{[i]} = A^T r^{[i-1]}$; ..... [S1]
3: $j^{[i]} = \arg\max_j |g_j^{[i]}| / ||Aj||$; ..... [S2]
4: $T^{[i]} = T^{[i-1]} \cup j^{[i]}$
5: $x^{[i]} = \arg\max_x ||y - A_{T[i]}x||$ ..... [S3]
6: $r^{[i]} = y - Ax^{[i]}$ ..... [S4]
7: **end**

Parameter

Step	Number of data	
	In	Out
1	A: 30 x 64 r : 30 x 1	g: 64 x 1
2	g: 30 x 1 A: 64 x 1	T: 5 x1
3	g: 30 x 1 A: 64 x 1 x: 64 x 1	x: 64 x1
4	g: 30 x 1 A: 64 x 1 x: 64 x 1	x: 64 x1
5	r: 30 x 1	-

components	Time parameters
Processor core	15 cycles per data
DMAC	1 cycle per data
Set DMAC	180 cycles per setting
Acclerator (computation)	300 cycles

FIGURE 22. User inputs in an example algorithm.

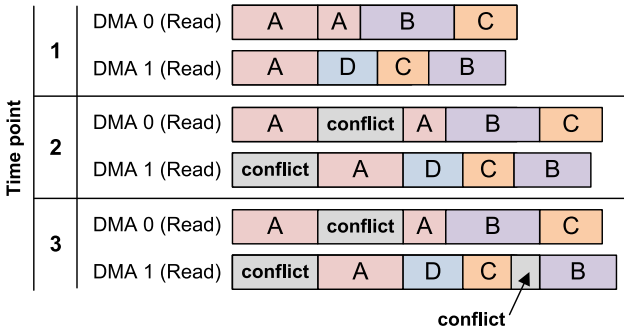

FIGURE 23. Impact of the memory conflict.

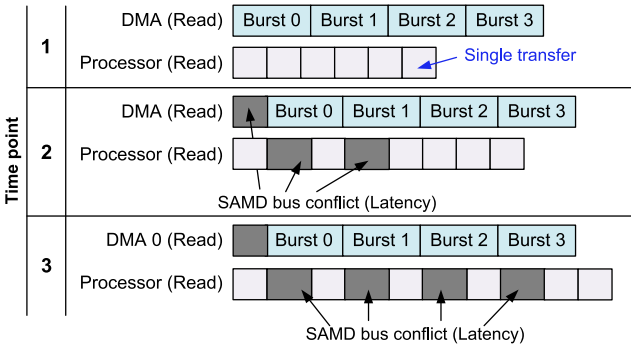

FIGURE 24. Impact of the SAMD bus conflict.

Algorithm 2 shows the means to model the memory conflict (or collision) and the SAMD bus conflict and to predict the performance. For every distinct two-step pair, where the step corresponds to an operation unit, memory and bus conflicts are examined in the pseudo-code of Algorithm 2. If the two masters are different, only the SAMD bus conflict is considered (*samdConflicts*) (Line 9). As was elaborated on in Section III.B, for the memory conflict, the conflict between DMA and DMA is significant whereas for the SAMD bus conflict, the conflict between DMA and processor is significant (Line 2). Each master is identified and the conflict is examined according to the combination or pair of masters. The smallest unit of data transfer is denoted as hwConstraint.master in Algorithm 2. In the inner-most loop, *hwConstraint.master[i]* and *hwConstraint.master[i+1]* are checked to see whether they are associated with the same memory access (*memoryConflicts*) (Line 3). If so, the access timing is identified to see whether they overlap with each other. If so as well, their access types (read or write) are identified. In the AXI protocol, the read channel and the write channel act separately and hence a conflict may not occur if both a read and a write occur simultaneously at the same memory. Lastly, if two transactions do not access the same memory at the same time, only SAMD conflicts are considered (*samdConflicts*) (Line 6).

Figure 23 shows the effect of the memory conflict between two DMAs on transactions. In the beginning, two DMAs are assumed to conduct read operations simultaneously at the same memory, memory A, as shown in time point 1 of Figure 23. As a result, a memory conflict occurs by which DMA1 read is delayed since DMA0 is assumed to have a higher priority. This is shown in time point 2 of Figure 23. DMA0 and DMA1 access memory A three times in total and at time point 2, no overlap occurs between these three accesses to memory A. However, the delay in DMA0 and DMA1 regarding memory A affects the following read operations and now a conflict is newly encountered in memory C, leading to the final situation shown in time point 3 of Figure 23.

Algorithm 3 Orthogonal Matching Pursuit (OMP)

Input: y, r, A
Output: $x^{[i]}$

1: **for** $i = 1; i = i + 1$ *till* $\|r^{[i]}\| = 0$ **do**
2: $g^{[i]} = A^T r^{[i-1]}$;
3: $j^{[i]} = \underset{j}{\operatorname{argmax}} |g_j^{[i]}| / ||A_j||$;
4: $T^{[i]} = T^{[i-1]} \cup j^{[i]}$;
5: $x^{[i]} = \underset{x}{\operatorname{argmax}} ||y - A_{T[i]}x||$;
6: $r^{[i]} = y - Ax^{[i]}$;
7: **end for**

Figure 24 shows how the transactions are affected by the SAMD bus conflict between DMA and processor. Assume that the masters individually access separate slaves during their reads. The first read of the DMA and the first read of the processor get started almost at the same time, as shown in time point 1 of Figure 24, and subsequently the SAMD bus receives the data for the DMA and the data for the processor on its two distinct data channels. At this time, the DMA which allows multiple outstanding transactions sends the address of the second read but since the corresponding slave cannot process this read address, the address is held on the bus. Owing to this address held on the bus, the address of the second read sent by the processor is not delivered immediately to the bus but delayed until the first read burst of the DMA is finished, as shown in time point 2 of Figure 24. In this manner, the read operation of the processor appears to be in sync with the data transfer of the DMA. This condition will last up to the penultimate DMA read burst and subsequently at the instant of the last DMA read burst, the DMA no longer has an address to send and hence the address channel of the SAMD bus is empty and the read operations of the processor are carried on without delay, as shown in time point 3 of Figure 24.

V. SIMULATION

To validate the proposed performance estimation methods, an example algorithm, OMP [34], [35], is taken and modeled. The OMP algorithm is utilized in compressive sensing and long-term evolution (LTE) to estimate the wireless channel. In order to prove that the proposed performance estimation algorithm is generally applicable to any other application, we have included additional experimental results on CNNs (AlexNet [36]) and wireless communications (LDPC-coded MIMO-OFDM [37], [38]). In the case of CNNs, the third layer for AlexNet is assumed as the example applications. Moreover, in the case of wireless communications, the LDPC-coded MIMO-OFDM consists of five functions, including initial synchronization, fast Fourier transform (FFT), channel estimation, multiple-input and multiple-output (MIMO) and low-density parity-check code (LDPC). The system architecture for the OMP, CNN and LDPC-coded MIMO-OFDM is implemented in the full-system simulator [7] and Xilinx Zynq 7020 SoC chip on ZedBoard [12] for comparison and verification.

A. SIMULATION METHODS

The OMP algorithm can estimate the LTE channel with a 5MHz bandwidth and is converted to a system architecture implemented on dual-core Zynq 7020 [12]. The algorithm is subdivided into 5 steps, S1-S5, corresponding to 5 operation units. Block interlacing is utilized to enable all the steps to concurrently run without data dependency. Hardware accelerators, DMAs, processors, and memories are connected to one another through an SAMD bus. Each data item is assumed to have its respective memory unless otherwise stated.

The OMP algorithm is revisited in Algorithm 3, which is equivalent to the left half of Figure 22. Five operation unit steps accompany the initialization phase: computing the correlation (S1), finding the location with the maximum correlation and adding that location to the set with locations found up to now (S2), estimating the x value by using the least mean squares method (S3), calculating the new r value by using the estimated x value (S4), and judging whether to proceed or not by using the calculated r value (S5). These steps are iterated, where if the r value in S5 is small enough to meet the performance required by the system, the x value estimated so far is printed and the algorithm is terminated. If the r value is not sufficiently small, the steps are iterated unceasingly and hence the maximum number of iterations is typically specified, which is set to 10 in this work.

TABLE 1. Simulation conditions.

Hardware Architecture	Partitioning	Scheduling	Memory
Case 1	SW: S2, S5 HW: S1, S3, S4	P1: S1 –> S3 –> S5 P2: S2 –> S4	Separate
Case 2	SW: S2, S5 HW: S1, S3, S4	P1: S1 –> S3 –> S5 P2: S2 –> S4	Shared
Case 3	SW: S2, S5 HW: S1, S3, S4	P1: S3 –> S1 P2: S5 –> S4 –> S2	Separate
Case 4	SW: S2, S4, S5 HW: S1, S3	P1: S1 –> S4 P2: S3 –> S2 –> S5	Separate

TABLE 2. Block interlacing applied to the OMP algorithm.

Step number	Iteration number															
	1					2		...	5					6		
Step 1	1	2	3	4	5	1	2		1	2	3	4	**5**	1	2	3
Step 2		1	2	3	4	5	1		5	1	2	3	**4**	5	1	2
Step 3			1	2	3	4	5		4	5	1	2	**3**	4	5	1
Step 4				1	2	3	4		3	4	5	1	**2**	3	4	5
Step 5					1	2	3		2	3	4	5	**1**	2	3	4

The simulation conditions are listed in Table 1, where 4 conditions, case 1–case 4, are shown. Case 1 is the optimal case in case that 3 hardware (HW) accelerators are used. Case 2–case 4 are the cases which deviate from the optimal by altering some condition(s). Case 2 uses a shared memory while case 1 uses separate memories for data items and hence memory conflicts become more frequent in case 2. Case 3 has a scheduling different from case 1. Namely, the order and the number of operations executed by processors P1 and P2 are changed, leading to changes in the conflict order and locations. Case 4 has a different partitioning. Namely, S4 executed by hardware is executed by software (SW), leading to the change in conflict in S4 from memory conflict to SAMD bus conflict and hence the potential increase in the execution time.

The proposed performance estimation methods are applied as follows. Block interlacing is assumed and different data items are processed in different steps in parallel. Table 2 shows the block interlacing applied to OMP. Each step operates on one of the data items 1-5 at every instant. By using the proposed methods, the performance of the last phase (or stage) of the fifth iteration (of the 10 iterations in total), which is the shaded area in Table 2, is predicted. Thus the first data item (the data entry numbered 1 in Table 2) for the fifth iteration is the target. Partitioning between processors and hardware accelerators to execute the operations

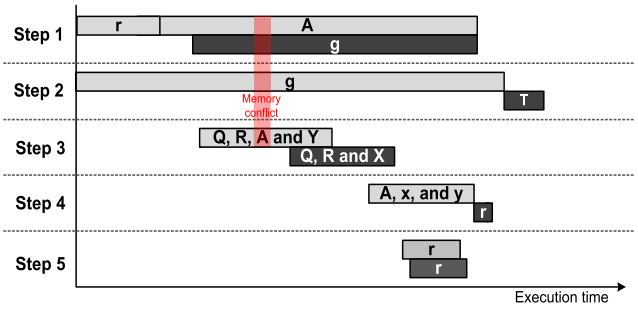

FIGURE 25. Memory conflict: Case 1.

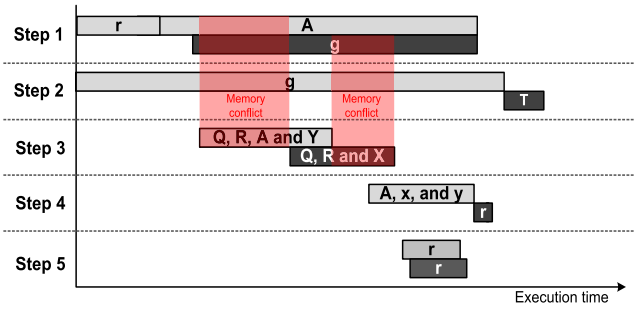

FIGURE 26. Memory conflict: Case 2.

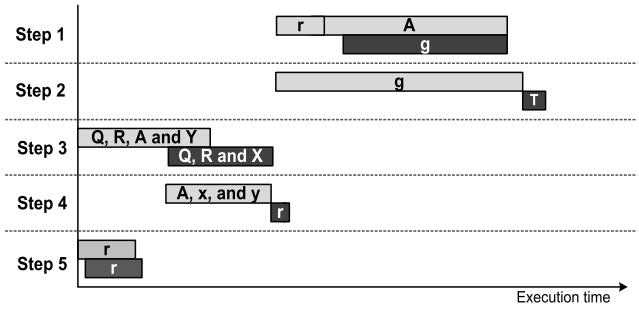

FIGURE 27. Scheduling: Case 3.

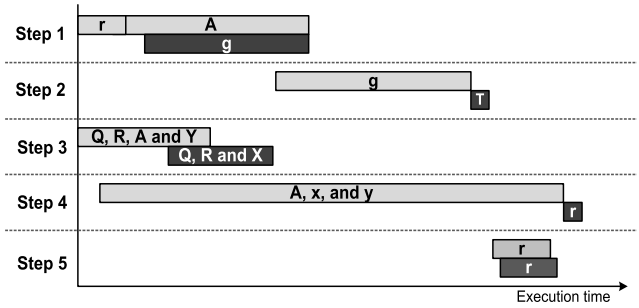

FIGURE 28. Partitioning: Case 4.

from S1 to S5 is determined and scheduling (or the operation order) is also determined to predict the overall execution time. The performance is predicted in view of the effect of memory merging (or the number of memories) as well.

B. SIMULATION RESULTS AND ANALYSIS

According to the simulation conditions mentioned above, simulation results from the system simulator are provided in this subsection. The performance of each of the modeled system architectures mapped from the example algorithm, OMP, is predicted and estimated and also compared with the result from the Zynq-on-ZedBoard implementation. First, simulation results and analysis of case 1 to case 4 in terms of memory merging, scheduling, and partitioning will be provided. Then, implementation through optimization will be explained. Lastly, simulation results will be briefly compared with those of an existing work.

Errors between the performance of the system simulator for the algorithm and the performance of the implementation in Zynq 7020 on ZedBoard are under 5% for all the simulation conditions, case 1–case 4. These errors come from the modeling based on the data transfer unit (by the burst) which exhibits lower accuracy than the cycle-accurate modeling (by the clock cycle). However, the details that impact the performance marginally are simplified in our modeling and instead the speed is enhanced by a factor of 10 to 100 in the proposed system simulator for expeditious performance prediction and large design space exploration.

Figure 25 shows the timing diagram of input and output data of each operation (e.g., S1) for case 1. In other words, for each operation or step, the upper half expresses the data fetched by the operation and the lower half expresses the outcome of the operation. The computation time is typically much smaller than the data transfer time and not visible from the outside but in case of S3, the computation time is relatively long and explicitly denoted as comp in Figure 25. In the first place, data r and then A are read by S1. As data A are read, outputs g is produced. In S2, all of data g are read and the outcome T is produced. In case of S1, data A and g can be processed concurrently since S1 is assumed to be handled by a hardware accelerator and hence two distinct DMAs take charge of the input and the output of the operation. Whereas both a read and a write are conducted concurrently in S1, either a read or a write is conducted at a time in S2 since S2 is assumed to be handled by a processor. Therefore, in case of S2, the output of the operation is produced after all the inputs are fetched. Figure 25 shows the case when each data item is allotted a separate memory. The memory conflict between two distinct hardware accelerators is only modeled and its effect is predicted in this work. Operation units or steps S1, S3, and S4 which are implemented in hardware accelerators are subject to memory conflict if the identical data item is read (or written) at the same time. Accordingly, memory conflict occurs where S1 and S3 read data A at once, which is the shaded area in Figure 25.

Figure 26 shows the memory conflict for case 2 which is defined in Table 1. The simulation with case 2 is to identify the effect of memory merging by using an integrated or shared memory instead of separate memories in case 1. In case 2, all the data are assigned one memory. All the other simulation conditions of case 2 are identical to those of case 1. The execution time will grow in case 2 because only one master

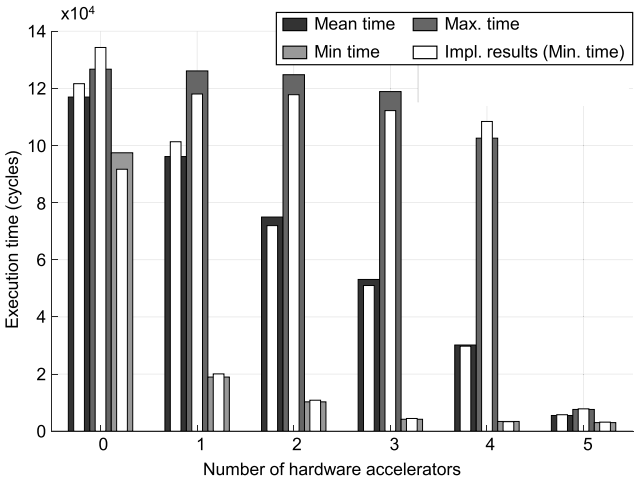

FIGURE 29. Execution time according to the number of hardware accelerators.

can access memory during a read or a write. The shaded areas where memory conflicts occur are shown in Figure 26. Memory conflicts occur between the read (or write) data transfer intervals of S1 and S3 regardless of what the data item is. The extent of memory conflict is affected by the amount of data read by S3, which is predicted to be 730 cycles from the system simulator and 620 cycles from the board implementation.

The effect of scheduling is illustrated in case 3 in comparison with case 1. As was shown in Figure 25 for case 1, S1 to S3 can operate concurrently, followed by S4 and S5. The maximum number of concurrent operations is 3. Figure 27 shows the scheduling of case 3 which was defined in Table 1. By means of the different scheduling in case 3 than in case 1, the maximum number of concurrent operations is 2 for case 3. For S2, more memory conflicts occur in case 1, leading to the execution time of 3000 cycles whereas less conflicts occur in case 3, exhibiting the execution time of 2900 cycles. However, the overall execution time is 1000 cycles longer for case 3 owing to the smaller number of concurrent operations.

Lastly, the effect of partitioning is illustrated in case 4 in comparison to case 1. Figure 28 shows the partitioning of case 4 which was defined in Table 1. The operation or step S4 in case 4 is implemented in software by processor P1 instead of hardware in case 1. This change in partitioning significantly increased the execution time of S4, yielding an increase in the overall execution time as well, which stems from the fact that the processor is slower in processing operations than the hardware accelerator.

Figure 29 shows the overall execution time as a function of the number of hardware accelerators according to the hardware and software partitioning. The optimum bar (minimum time) is obtained by choosing the minimum execution time across all combinations of scheduling and memory merging for each number of hardware accelerators. The least optimum bar (maximum time) is obtained by choosing the maximum execution time over all combinations of scheduling and

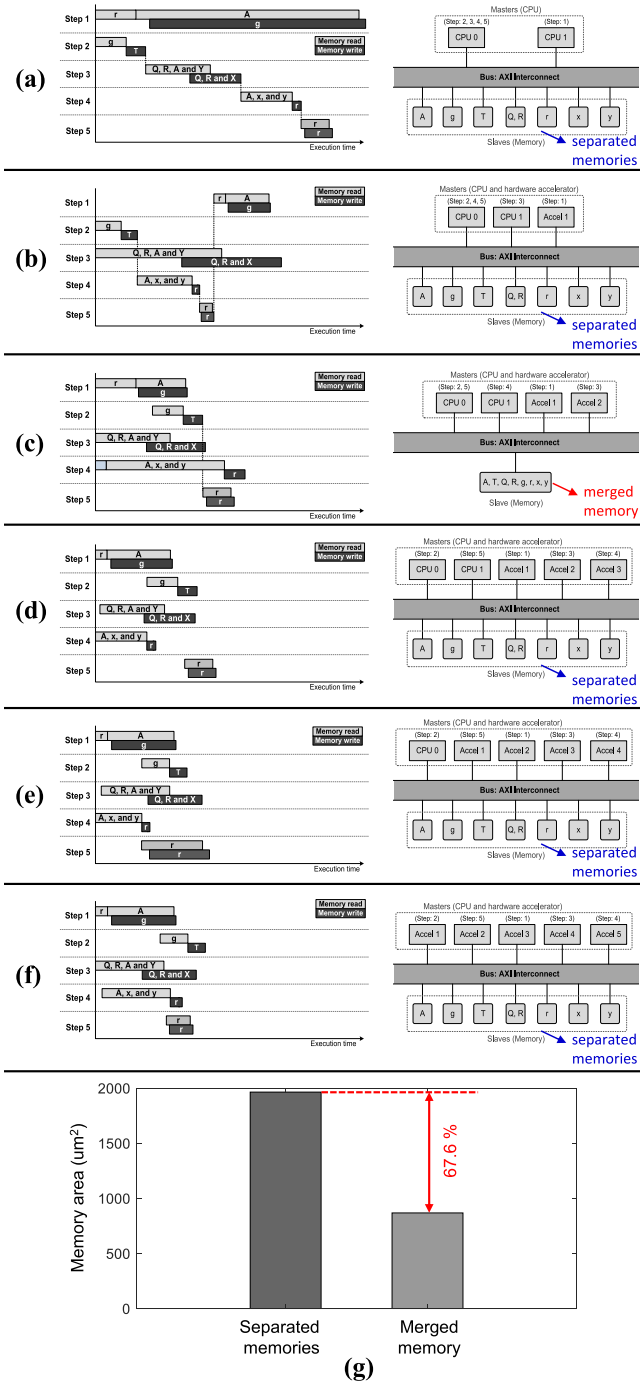

FIGURE 30. Illustration of the optimum design parameter combinations for (a) two processor cores, (b) two processor cores and one hardware accelerator, (c) two processor cores and two hardware accelerators, (d) two processor cores and three hardware accelerators, (e) one processor core and four hardware accelerators, (f) five hardware accelerators, and (g) comparison of memory area between separated memories and merged memory.

memory merging for each number of hardware accelerators. The mean bar is obtained by calculating the average execution time for all combinations of scheduling and memory merging for each number of hardware accelerators. Generally, as more hardware accelerators are used, the execution time tends to

drop since more operations can be carried on concurrently. The optimum execution time is improved maximally, relative to the mean execution time, when only one hardware accelerator is used. This means a specific operation will occupy a large percentage of the overall execution time when processors deal with operations and also the gain will be large if the operation is managed by the hardware accelerator. In the architecture optimization process below, two hardware accelerators are assumed to be used. Figure 30 shows the heterogeneous SoC architecture of each optimum bar in Figure 29, their timing diagram and memory complexity. As mentioned earlier, the execution time tends to decrease when more hardware accelerators and memories are used in the heterogeneous SoC architecture, because more tasks can be performed simultaneously without memory collision. Additionally, since it is more efficient for the hardware accelerator to access the memory in burst units than for CPUs to read and write to the memory, the performance of the heterogeneous SoC architecture improves as the number of hardware accelerators increases. For example, in Figure 30 (a), since the steps assigned to each CPU are executed sequentially, scheduling cannot be freely performed through block-interlacing manner [33]. However, as Figure 30 (f) assumes that all steps are implemented with a hardware accelerator, block interlacing can be applied to execute all steps simultaneously and without dependency. Figure 30 shows that allocating memory for each variable further improves performance. Furthermore, as shown in Figure 30 (c), if bus masters do not access memory simultaneously by hardware-software partitioning and scheduling, hardware complexity can be reduced, and high-performance gain can be obtained through merged memory. This shows that using separate memory for a heterogeneous SoC architecture is not always optimal for execution time. Figure 30 (g) illustrates the hardware area measured by the memory model based on the commercial SRAM memory compiler provided with the TSMC 28-nm standard cell library. The simulation result indicates that the hardware area of the merged memory manner (Figure 30 (c)) is approximately 67.6% smaller than that of the separated memory manner (Figure 30 (a), (b), (d), (e), and (f)). The separate memory manner allocates additional input/output (I/O) ports than the merged memory manner, which is the primary reason the memory area is different between the two methods using the same capacity of memory. In addition, multiple separated memories make the AXI interconnect more complex. It leads to making the overall heterogeneous SoC architecture more complex. As a result, the proposed simulator can achieve a heterogeneous SoC architecture that can achieve high-performance gains with low memory complexity.

For the OMP algorithm, various system architecture candidates are searched for in terms of partitioning, scheduling, and memory merging, where two hardware accelerators are assumed to be used. The optimum architecture has the parameters listed in Table 3. To execute S1 and S3, two hardware accelerators are employed and to run S2, S4, and S5, processors are used. According to the chosen scheduling, processor P1 runs one hardware accelerator for S1 and then runs the S5 operation, followed by the S4 operation. Processor P2 first runs the other hardware accelerator for S3 and runs the S2 operation. The simulated execution time of the optimized architecture is 10295 cycles, obtained from the system simulator, and the actual execution time of the optimized architecture is 10788 cycles, obtained from the board implementation, resulting in a 4.8% error.

TABLE 3. Optimized design option.

Optimum design point	Partitioning	Scheduling	Memory Merging
	SW: S2, S4, S5 HW: S1, S3	P1: S1 –> S5 –> S4 P2: S3 –> S2	Separate

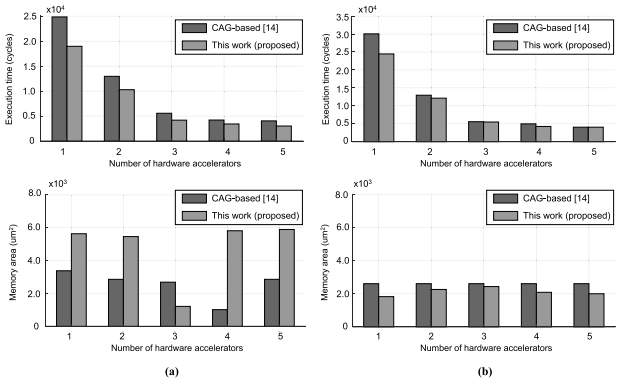

FIGURE 31. Comparison of computational complexity between the CAG-based optimization and the proposed optimization: (a) throughput optimization and (b) memory area optimization.

Some comparison is made with an existing work [14] where the performance difference according to scheduling is not considered but a predetermined scheduling is fixed during the simulation. However, if the timing of an operation to access data coincides with the timing of another operation to access the same data, a conflict occurs and the performance is impacted accordingly. Thus the effect of scheduling for the OMP algorithm is considered for comparison with the method in [14]. Assuming two hardware accelerators are used, the optimum architecture for the algorithm is shown in Table 3, which has an execution time of 10295 cycles from simulation. If the optimum scheduling is not explored but the operations are executed from S1 to S5 in order, then the execution time is 11604 cycles, even if partitioning and memory merging are the same for the two architectures. Scheduling is not considered in [14] while the optimum execution order can be explored in this work which models the data transfers with both the processor and the DMA. The execution time difference between the architecture with optimum scheduling and the architecture without scheduling is about 1309 cycles, leading to a 12.7% discrepancy if conflicts are not taken into account.

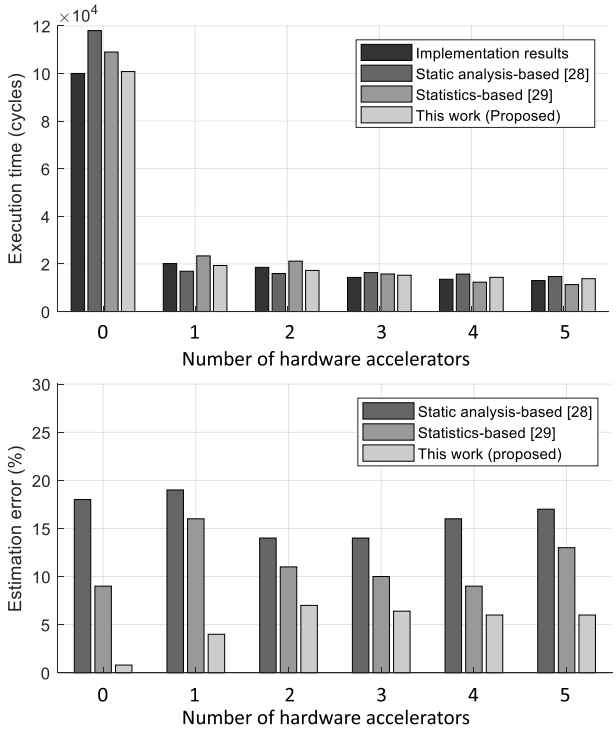

FIGURE 32. Performance estimation results of heterogeneous SoC architecture based on static analysis-based [28], statistics-based [29] and this work (proposed).

C. COMPARISON OF THE OPTIMIZATION

Figure 31 (a) compares the computational complexity when the optimization goal is to maximize the throughput, or, equivalently, to minimize the execution time. The simulation results show that the design options optimized by the proposed simulator can improve the throughput performance by up to 32%, compared to that of the CAG-based optimization. This can be explained by the fact that the proposed optimization can minimize the memory collision by optimizing both the hardware-software partitioning and the scheduling, as opposed to the CAG-based optimization. In addition, the proposed optimization can also reduces the latency induced by bus conflicts by optimizing the scheduling. However, it is shown in the figure that the throughput improvement comes at the expense of memory area, more specifically, by 61%, compared to the CAG-based optimization. However, this does not imply that the proposed optimization always results in more memory area.

Figure 31 (b) shows the computational complexity when the optimization goal is to minimize the memory area. For fair comparison, the maximum memory size has been set to 24.4 Kbytes. The simulation results show that, compared with the previous throughput optimization, the throughput improvement of the proposed optimization has been reduced to 19%. Instead, the proposed optimization can provide the average memory area saving of 20%, as shown in Figure 31 (b). In other words, the proposed optimization outperforms the CAG-based optimization in terms of both throughput and memory area, as opposed to the previous

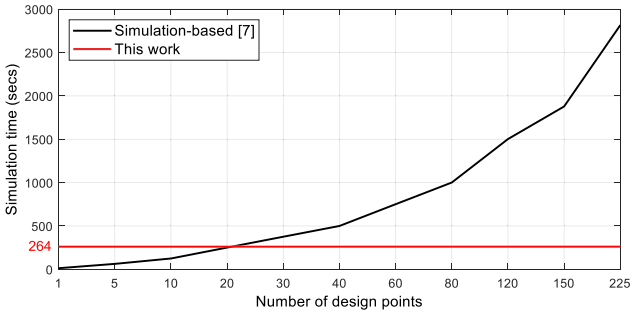

FIGURE 33. Comparison of simulation time between the proposed performance estimation (this work) and conventional simulation-based approach [7].

throughput optimization. The reason is that the proposed optimization helps determine the scheduling that minimizes the collision-induced latency in the heterogeneous SoC architecture. Therefore, it can be concluded that the proposed optimization outperforms the CAG-based optimization, regardless of the optimization goal.

Figure 32 shows that the proposed performance estimation algorithm can predict the communication performance of heterogeneous SoC architectures for the OMP algorithm more accurately compared with the static analysis-based estimation [28] and statistics-based estimation [29]. The proposed estimation algorithm considers both the bus protocol overhead (bus conflict) and memory latency (memory collision) using the evaluation board (e.g., ZedBoard [12]) for evaluating the time information of the hardware and software. The experimental results show that the proposed algorithm approaches the full-system simulator [7] more closely than the conventional algorithms. For example, the proposed algorithm reduces the estimation error to 6%, whereas the conventional algorithms in [28] and [29] experience the estimation errors of 18% and 16%, respectively.

D. COMPARISON OF SIMULATION TIME

Figure 33 compares the proposed performance estimation and the conventional simulation-based approach in terms of simulation time. As shown in figure, compared with the conventional simulation-based [7] approach, the proposed performance-estimation algorithm provides a speedup of two orders of magnitude. The simulation-based approach is accurate enough to capture the dynamic nature of the communication bandwidth, but it often takes a prohibitively long time to simulate. According to our experiments, the conventional simulation-based approach takes a few hours to evaluate a single heterogeneous SoC architecture with hundreds of different combinations of hardware-software partitioning, memory merging, and scheduling. Using the conventional simulation-based approach, we run the full-system simulator proposed in [7], once for each design point, taking approximately 12.5 seconds per design point. In contrast, to minimize the simulation time and maintain estimation accuracy, the proposed performance estimation algorithm constrains the

Algorithm 4 The Third Convolutional Layer of AlexNet

Input: NoBatch, IAs, Wts and OAs
Output: poolout

```
for b=1:NoBatch do
    convout = convlayer (IAs, Wts, OAs);
    biasout = bias (convout);
    reluout = relu (biasout);
    lrnout = LocalRespNorm(reluout);
    poolout = pool(lrnout);
end for
```

Algorithm 5 LDPC-Coded MIMO-OFDM

Input: Symbols and NoSymbols
Output: ldpcout

```
for s=1:NoSymbols do
    syncout = synchronization (Symbols);
    fftout = fft (syncout, Symbols);
    chest = channelestimation (fftout);
    mimoout = mimo (chest, fftout);
    ldpcout = ldpc (mimoout);
end for
```

use of an evaluation board [12] to evaluate the time information of the heterogeneous SoC architecture. Since a few tens of time information is sufficient to express most of the heterogeneous SoC architectures of interest, the extra simulation time required to obtain the time information of each hardware component becomes negligible, particularly in the case of broad design space (i.e., a space of hundreds of design points). In the case of thousands of design points, the performance estimation algorithm accelerates the conventional simulation-based approach by two orders of magnitude. Figure 33 shows that the extra time required for prior simulations becomes negligible as the design space increases. Thus, it can be concluded that the proposed performance-estimation algorithm can estimate orders of magnitude faster than the conventional time-consuming simulation-based approach.

E. EXTENSION TO OTHER APPLICATIONS

In this subsection, the proposed performance estimation algorithm is extended to CNNs (the third convolutional layer of AlexNet [36]) wireless communications (LDPC-coded MIMO-OFDM [37], [38]) applications. As shown in Algorithm 4, the third convolutional layer consists of five functions, including convolutional accelerator, bias, ReLU, LRN and Pooling. As shown in Algorithm 5, the LDPC-coded MIMO-OFDM consists of five functions, including initial synchronization, fast Fourier transform (FFT), channel estimation, multiple-input and multiple-output (MIMO) and low-density parity-check code (LDPC).

Figure 34 (a) shows that the performance estimation error of the third convolutional layer of AlexNet [36] is less than 5.9% compared to the performance estimation result of AccTLMSim [7]. Moreover, it achieves a significant

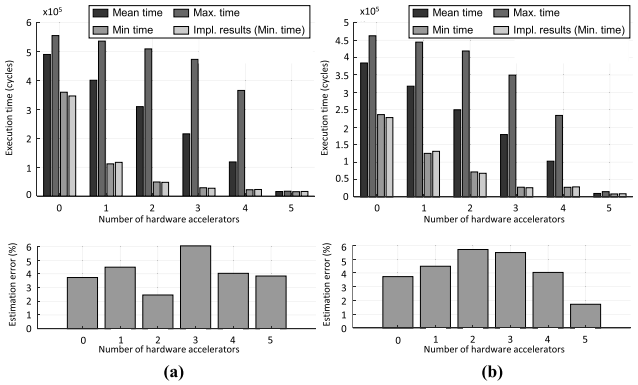

FIGURE 34. Execution time according to the number of hardware accelerators: (a) third convolutional layer of AlexNet [36] and (b) LDPC-coded MIMO-OFDM [37], [38].

performance gain i.e., 48% for the third convolutional layer of AlexNet. This is also the case when the proposed performance estimation algorithm is applied to a LDPC-coded MIMO-OFDM. Figure 34 (b) shows that the estimation error for a LDPC-coded MIMO-OFDM is smaller than 5.6% compared to the performance estimation result of AccTLM-Sim [7], and the performance gain is 56%. The experimental results show that although the optimum hardware-software partitioning, scheduling, and memory merging tend to vary with the application, the proposed performance estimation algorithm is generally applicable to any heterogeneous SoC with a reasonably small estimation error and a noticeable performance gain.

VI. CONCLUSION

The system speed can be improved by using hardware accelerators. However, if the data transfer pattern is not optimized, the heterogeneous SoC may give rise to increased conflicts or collisions, delaying the data transfer in the bus and memory and degrading the system performance more than the ideally predicted one. In this work, conflicts or collisions are modeled to predict the performance and also a system simulator is developed to cover a large design space in terms of partitioning, scheduling, and memory merging, by which the optimum architecture to minimize the execution time can be found. In a setting where multiple masters and slaves are connected to a bus, the SAMD bus conflict and the memory conflict are modeled in consideration of multiple outstanding transactions and operation units.

By using the proposed performance estimation methods, an example algorithm, OMP, for LTE 5MHz channel estimation is implemented for four cases. The performance from simulation and the performance from actual implementation are compared to validate the reliability of the system simulator. For all the simulation conditions, the error between the predicted execution time and the actual execution time is under 5%. Assuming the number of hardware accelerators is 2, the optimum architecture for the OMP algorithm is extracted. In addition, compared with the conventional simulation-based approaches, the proposed estimation algorithm provides a speedup of one to two orders

of magnitudes. In this work, by considering scheduling in the algorithm where block interlacing or pipelining is applied, the optimum system architecture can be found with an improved performance. For example, the optimized heterogeneous SoC architecture for the OMP algorithm improves performance by up to 32% compared with the conventional CAG-based approaches. The proposed simulator is verified that the proposed performance estimation algorithm is generally applicable to estimate the performance of any heterogeneous SoC architecture. For example, the estimation error is measured to be no more than 5.9% for the convolutional layers of CNNs and no more than 5.6% for the LDPC-coded MIMO-OFDM. In addition, the optimized heterogeneous SoC architecture improves performance by up to 48% for the third convolutional layer of AlexNet and 56% for the LDPC-coded MIMO-OFDM.

Lastly, it is worthwhile to mention that the estimation algorithm in the proposed simulator is generally applicable to any heterogeneous SoC architecture. In particular, the extension of the performance estimation algorithm into the emerging compute-in-memory (CiM) hardware accelerators for general matrix to matrix multiplication (GEMM) are considered to be promising for future work. Note that such a CiM-based hardware accelerators for GEMM are often equipped with DMACs [39-43]. In addition, as a standalone IP, it is connected to an off-chip memory through an on-chip bus [44, 45]. Moreover, the memory allocation (e.g., bank allocation of DRAM) tends to affect the communication performance of the emerging CiM-based hardware accelerators, as depicted in [46, 47]. Thus, we expect the performance estimation algorithm proposed in this paper to be generally applicable to the CiM-based hardware accelerators for GEMM.

REFERENCES

[1] L. Cohen, A. Nadkarni, P. Rutten, K. Stolarski, and J. Vela, "IDC's world wide computing platforms taxonomy," Int. Data Corp., Needham, MA, USA, Tech. Rep. US42024017, 2017.
[2] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 58–68, Jan. 2011.
[3] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, Apr. 2015.
[4] M. A. Suchard, Q. Wang, C. Chan, J. Frelinger, A. Cron, and M. West, "Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures," *J. Comput. Graph. Statist.*, vol. 19, no. 2, pp. 419–438, Jan. 2010.
[5] S. Y. Shao, "Design and modeling of specialized architectures," Ph.D. dissertation, Harvard Univ., Cambridge, MA, USA, 2016.
[6] *Zynq-7000 All Programmable SoC Technical Reference Manual V1.12.2*, Xilinx, San Jose, CA, USA, Jul. 2018.
[7] S. Kim, J. Wang, Y. Seo, S. Lee, Y. Park, S. Park, and C. S. Park, "Transaction-level model simulator for communication-limited accelerators," 2020, *arXiv:2007.14897*.
[8] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, "Co-designing accelerators and SoC interfaces using gem5-Aladdin," in *Proc. Int. Symp. Microarchitecture*, Oct. 2016, pp. 1–12.
[9] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.
[10] J. Wang, S. Park, and C. S. Park, "Optimization of communication schemes for DMA-controlled accelerators," *IEEE Access*, vol. 9, pp. 139228–139247, 2021.
[11] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019.
[12] Digilent. *ZedBoard Zynq-7000 ARM/FPGA SoC Development Board*. Accessed: Apr. 10, 2019. [Online]. Available: https://store.digilentinc.com/zedboardzynq-7000-arm-fpga-soc-development-board/
[13] *Altera SoC FPGAs*. Accessed: Nov. 8, 2017. [Online]. Available: http://www.altera.com/devices/processor/soc-fpga/overview/procsoc-fpga.html
[14] L. Kanishka, A. Raghunathan, and S. Dey, "System-level performance analysis for designing on-chip communication architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 6, pp. 768–783, Jun. 2001.
[15] Y. Cho, G. Lee, S. Yoo, K. Choi, and N.-E. Zergainoh, "Scheduling and timing analysis of HW/SW on-chip communication in MP SoC design," in *Proc. IEEE Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2003, pp. 132–137.
[16] S. Kim, C. Im, and S. Ha, "Efficient exploration of on-chip bus architectures and memory allocation," in *Proc. 2nd IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Sep. 2004, pp. 248–253.
[17] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, Feb. 2006.
[18] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP J. Embedded Syst.*, vol. 2007, pp. 1–11, Dec. 2007.
[19] *AMBA AXI and ACE Protocol Specification, AXI3, AXI4, and AXI4-Lite, ACE and ACE-Lite, ARM Infocenter*, ARM Ltd., Cambridge, U.K., 2011.
[20] S. Kim, S. Park, and C. S. Park, "System-level communication performance estimation for DMA-controlled accelerators," *IEEE Access*, vol. 9, pp. 141389–141402, 2021.
[21] S. Sombatsiri, K. Kobashi, K. Sakanushi, Y. Takeuchi, and M. Imai, "An AMBA hierarchical shared bus architecture design space exploration method considering pipeline, burst and split transaction," in *Proc. 10th Int. Conf. Electr. Eng./Electron., Comput., Telecommun. Inf. Technol.*, May 2013, pp. 1–6.
[22] C. Lin, X. Du, X. Jiang, and D. Wang, "An efficient and effective performance estimation method for DSE," in *Proc. Int. Symp. VLSI Design, Automat. Test (VLSI-DAT)*, Apr. 2016, pp. 1–4.
[23] M. Makni, S. Niar, M. Baklouti, G. Zhong, T. Mitra, and M. Abid, "A rapid data communication exploration tool for hybrid CPU-FPGA architectures," in *Proc. 25th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, 2017, pp. 85–92.
[24] H. Meng, H. Meng, P. Ding, M. Wang, and D. Wang, "A design space exploration method for on-chip memory system based on task scheduling," in *Proc. IEEE 9th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2018, pp. 912–915.
[25] M. Xie, D. Tong, K. Huang, and X. Cheng, "Improving system throughput and fairness simultaneously in shared memory CMP systems via dynamic bank partitioning," in *Proc. Int. Symp. High Perform. Comput. Archit.*, Feb. 2014, pp. 344–355.
[26] Y. Liu, J. Lu, D. Tong, and X. Cheng, "Locality-aware bank partitioning for shared DRAM MPSoCs," in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 16–19.
[27] S. Murali, L. Benini, and G. De Micheli, "An application-specific design methodology for on-chip crossbar generation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1283–1296, Jun. 2007.
[28] S. Kim, C. Im, and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 5, pp. 19–24, May 2005.
[29] R. V. W. Putra, M. A. Hanif, and M. Shafique, "DRMap: A generic DRAM data mapping policy for energy-efficient processing of convolutional neural networks," 2020, *arXiv:2004.10341*.
[30] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Fast exploration of bus-based on-chip communication architectures," in *Proc. 2nd IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Sep. 2004, pp. 242–247.

[31] G. D. Micheli and L. Benini, *Networks on Chips: Technology and Tools*. San Francisco, CA, USA: Morgan Kaufmann, Aug. 2006, ch. 8.

[32] S. Pasricha and N. Dutt, *On-Chip Communication Architectures*. Burlington, VT, USA: Morgan Kaufmann, 2008, chs. 2–9.

[33] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 1, pp. 74–78, Jan. 2008.

[34] P. Maechler, P. Greisen, N. Felber, and A. Burg, "Matching pursuit: Evaluation and implementatio for LTE channel estimation," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 589–592.

[35] Y. C. Eldar and G. Kutyniok, *Compressed Sensing: Theory and Applications*. New York, NY, USA: Cambridge Univ. Press, 2012.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[37] P.-Y. Tsai, P.-C. Lo, F.-J. Shih, W.-J. Jau, M.-Y. Huang, and Z.-Y. Huang, "A 4 × 4 MIMO-OFDM baseband receiver with 160 MHz bandwidth for indoor gigabit wireless communications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 12, pp. 2929–2939, Dec. 2015.

[38] T. Suzuki, H. Yamada, T. Yamagishi, D. Takeda, K. Horisaki, T. V. Aa, T. Fujisawa, L. Perre, and Y. Unekawa, "High-throughput, low-power software-defined radio using reconfigurable processors," *IEEE Micro*, vol. 31, no. 6, pp. 19–28, Dec. 2011.

[39] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable embedded microprocessor for bit-scalable in-memory computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, Sep. 2020.

[40] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, Jun. 2020.

[41] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu, and M.-F. Chang, "A 65 nm 4 Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 496–498.

[42] M. Zhu, Y. Zhuo, C. Wang, W. Chen, and Y. Xie, "Performance evaluation and optimization of HBM-enabled GPU for data-intensive applications," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1245–1248.

[43] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking high bandwidth memory on FPGAS," in *Proc. IEEE 28th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2020, pp. 111–119.

[44] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gomez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "NERO: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in *Proc. Field-Program. Log. Appl. (FPL)*, Sep. 2020, pp. 9–17.

[45] A. Kurth, W. Rönninger, T. Benz, M. Cavalcante, F. Schuiki, F. Zaruba, and L. Benini, "An open-source platform for high-performance non-coherent on-chip communication," 2020, *arXiv:2009.05334*.

[46] P. Gu, X. Xie, S. Li, D. Niu, H. Zheng, K. T. Malladi, and Y. Xie, "DLUX: A LUT-based near-bank accelerator for data center deep learning training workloads," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 8, pp. 1586–1599, Aug. 2021.

[47] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 2, pp. 420–434, Feb. 2018.

JOOHO WANG received the B.S. degree in electronics engineering from Korea Polytechnic University (KPU), Siheung, South Korea, in 2014. He is currently pursuing the M.S./Ph.D. degree in electronics engineering with Konkuk University, Seoul, South Korea. His research interests include hardware/software co-design of programmable accelerators and simulation for SoC architecture.

YUNGYU GIM received the B.S. and M.S. degrees in electronics engineering from Konkuk University, Seoul, South Korea, in 2015 and 2018, respectively. He is currently conducting research on hardware security architectures for the Android mobile OS. His research interest includes tamper-resistant integrated secure element (iSE).

SUNGKYUNG PARK (Senior Member, IEEE) received the Ph.D. degree in electronics engineering from Seoul National University, South Korea, in 2002. From 2002 to 2004, he was with Samsung Electronics, as a Senior Engineer, where he worked on the development of system-level simulators for cellular standards. From 2004 to 2006, he was with the Electronics and Telecommunications Research Institute (ETRI), as a Senior Member of Research Staff, where he worked on fiber-optic front-end IC design. From 2006 to 2009, he was with Ericsson Inc., as a Senior Staff Hardware Designer, where he worked on the design and modeling of multi-standard RF transceivers and clocking circuits. In 2009, he joined as a Faculty Member with the Department of Electronics Engineering, Pusan National University, South Korea, where he is currently a Professor. His research interests include design and modeling of SoC, hardware accelerators, and virtual platforms for neural networks and 5G.

CHESTER SUNGCHUNG PARK (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 2006. From 2006 to 2007, he was with Samsung Electronics, Giheung, South Korea. From 2007 to 2013, he was with Ericsson Research, Plano, TX, USA, as a Senior Engineer. Since 2013, he has been with the Department of Electronics Engineering, Konkuk University, South Korea, as an Associate Professor, where he is working on the design and modeling of SoC, hardware accelerators, and virtual platforms for neural networks and 5G. His research interests include SoC architecture design for artificial intelligence, processing in memory, and wireless communication.