5

Designing Accelerated Edge AI Systems with Model Based Methodology

Petri Solanti[1] and Russell Klein[2]

[1]Siemens EDA, Germany
[2]Siemens EDA, USA

Abstract

Deploying AI at the edge can be challenging. AI algorithms are very compute intensive. In the data centre, multiple large, power-hungry GPUs are often employed. However, edge systems typically have constrained compute capabilities and limited power. Further, many systems need to deal with size, weight, cost, thermal, and other limitations. Successfully deploying AI while meeting these limitations requires a holistic analysis of the system. A Model-Based Cybertronic System Engineering (MBCSE) methodology enables modelling and analysis of complex systems at a high abstraction level. It can be used to analytically find an optimal system architecture and hardware/software partitioning. Meeting the computational requirements may call for the development of bespoke machine learning accelerators. These are complex dedicated compute resources that deliver parallel computation, local data buffers, and some level of programmability. Designing an optimal accelerator architecture can be accomplished with an AI assisted High-Level Synthesis (HLS) process to efficiently explore the design space.

This paper describes a Model Based Cybertronic System Engineering (MBCSE) methodology that can be used to craft a combined hardware/software implementation of an inferencing algorithm, balancing performance, power, cost, and other key design metrics. It begins with an algorithmic analysis, determining areas of significant complexity. This is followed by allocation of functions to physical computation elements, targeting

both board-level and chip-level placement. During the allocation phase, complex algorithms may be mapped to bespoke accelerators that will be synthesized from the algorithmic description using high-level synthesis. Finally, an analysis of design is performed to ensure that all design metrics are met.

Keywords: edge AI, system design, system optimization, high-level synthesis.

5.1 Introduction and Background

Edge systems are often limited in several dimensions. Power consumption and compute capabilities are often limited in support of form-factor, weight, cost, mobility, and other requirements. This makes deploying AI on these systems challenging, as AI algorithms typically consume significant compute resource and power. One way to mitigate this is to architect the system with the compute resources that meet, but do not materially exceed, the requirements for the AI processing needed.

AI processing can be performed on different types of compute resources. For example, inferences can be run on general-purpose processors, graphics processing units (GPUs), arrays of multiply/accumulate processing elements, FPGA fabric, or bespoke hardware accelerators. Each of these represents compromises for the system designer. For example, deploying AI on a general-purpose processor typically will require around 100 watts of power or more. This results in a heavy battery, limited battery life, and potential cooling issues. But it significantly eases the development efforts, as the same software and machine learning frameworks can be used in the edge design as was used by the data scientists in the data centre. A bespoke hardware accelerator will be orders of magnitude more efficient but requires a custom IC development effort.

Developers need a way to understand the impact of their design trade-offs early in the design cycle to find the optimal architecture for the system that addresses the myriads of design constraints imposed by the business, regulatory, competitive, and other forces influencing the design of the system. Yet, finding a suitable methodology that can address all the design needs is tricky.

Model-based Systems Engineering has been used for software development for 25 years. Unified Modelling Language (UML) [1] was the first standardized graphical modelling language for specification, construction, documentation and visualization of software intensive systems. Because

UML was originally developed for modelling SW systems, it was not suitable for general system modelling. System Modelling Language (SysML) was based on UML but targeted more to the needs of general systems engineering. It is widely used for modelling system functionality but has severe capacity limitations.

Systems with functionality implemented in both SW and bespoke HW blocks are called Cybertronics Systems [2]. These are difficult to model with UML or SysML, which are targeted to single-domain modelling. Finding the optimal HW/SW partitioning needs an extensive design space exploration and capability to analyse different metrics like processor and bus utilization, task latency, power consumption or network load. The modelling methodology must support clear separation of function from structure, function allocation to structural elements and mapping of structural elements to any target technology. One of the methodologies developed for this purpose is Architecture Analysis and Design Language (AADL) [3]. It is used to model the SW and HW architectures of embedded real-time systems. AADL is a useful methodology for HW/SW system modelling and analysis, but it is lacking capabilities to the operational and functional analysis and modelling of cyber-physical systems.

5.2 Model Based Cybertronic Systems Engineering

Challenges of the cybertronics systems engineering are diverse. It begins with the system context that can be a network, a computing enclosure with multiple Printed Circuit Boards (PCB), a single PCB, a System-on-Chip (SoC), a Field Programmable Gate Array (FPGA) or embedded SW. The physical system sets the functional constraints that define the requirements for the cybertronics subsystems. Furthermore, the individual algorithms communicating with each other and consuming computing resources, can be allocated to different processing elements. The design space is huge and finding the optimal architecture is difficult.

Another challenge is the variety of the design domains that are usually tightly coupled. An architectural component like PCB contains other architectural components that are cybertronics subsystems themselves such as 3DIC or SoC. SW functionality can be allocated onto multiple processors that are potentially in different subsystems. In such cases the network of data exchanges between the functions must be allocated to physical interconnect that can be a network segment, platform bus like PCIe, network-on-chip, or something similar, depending on the implementation technology.

Third challenge is the fragmented organization of the domain specific development processes. Communication between the teams is negligible because the different terminologies and specifications are interpreted differently. This makes maintaining the integrity of the system difficult.

Model Based Cybertronic Systems Engineering is a new model-based methodology developed for modelling and analysis of cybertronics systems. It borrows concepts from multiple modern systems engineering methodologies. The main target of this new methodology is to unify the modelling methodologies on different system levels and enable seamless communication between the architects in the different implementation domains and the design teams.

The first methodology is called HW/SW co-architecting [4]. It is based on C++ function tree breakdown, where the functions are allocated to SW that is executed on one or more processors or HW accelerators that are implemented by using HLS. Because the C++ source code can be used both as SW code and input for the HLS, the mapping decisions can be done late in the design cycle. This approach works nicely for algorithms that are available, or can be translated into, C++, like streaming video or AI algorithms.

The second methodology used in MBCSE is ARChitecture Analysis and Design Integrated Approach, ARCADIA [5]. It is a very simple static information model methodology for system modelling according to the INCOSE SE handbook. Arcadia can be used to model the functional, logical, and physical architecture of the system using standardized artifacts. The functions and structures are kept separated, but the allocation of functions, exchanges, and components to structures is supported. Arcadia methodology has no technology binding nor simulation capabilities itself, so the designer can specify any target technology or simulation by using properties. It also has capability to transition a component to a subsystem in a separate project. These three capabilities make Arcadia a perfect methodology for cybertronics system modelling.

Yet, a comprehensive system modelling methodology must enable design space exploration with different types of simulations. Property Model Methodology [6] is a dynamic, simulation-based methodology for requirements driven system modelling and analysis. Simulations are needed to validate the functional correctness of the system model, to analyse the system performance of different architecture options and verify the implementation. PMM and Arcadia complement each other and form a basis for a cybertronics system modelling and validation methodology.

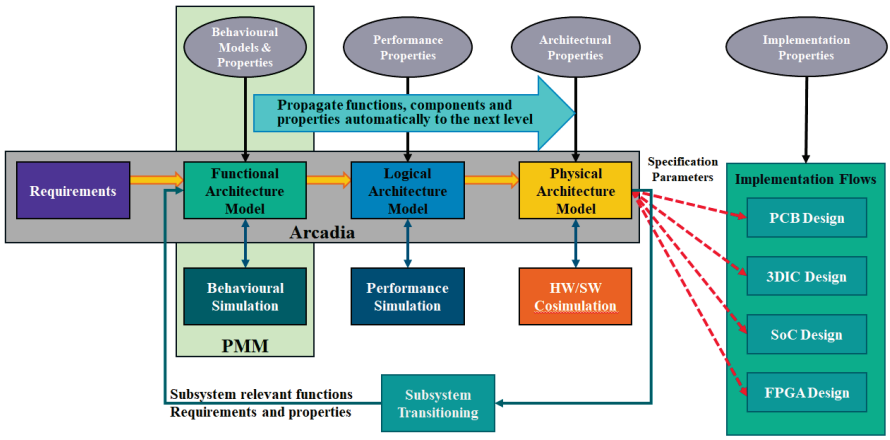

Figure 5.1 Model-Based Cybertronics Systems Engineering Methodology

MBCSE [7] workflow is described in the Figure 5.1. The horizontal Requirements-Functional-Logical-Physical (RFLP) architecture process is based on Arcadia. The transition from one design layer to another one is automated to maintain the integrity between the layers.

The vertical pillars for each Arcadia layer represent the PMM concepts used to import behavioural, performance, and implementation information to the Arcadia model and interfacing to the different simulations. Complexity management with subsystem transition is part of the Arcadia methodology.

The third integral part of the MBCSE methodology is interface to all cybertronics implementation domains. To keep the number of details to a manageable level in the system model the transition to a domain specific design flow is a design step, not an automatic model generation.

The MBCSE methodology allows modelling any type of system. Therefore, it can be used for modelling of cyber-physical systems with, for example, mechanical, mechatronics, electrical, and physical subsystems as well. Each model can be specified to be an abstract model without any technology binding, or it can be bound to a target technology like a platform network or PCB, where the components in the model are physical components at this model level like a processor, SoC, or FPGA. But the component is a top-level functional model of a separate subsystem that is decomposed individually. Yet, all functions, requirements, and properties related to the top-level are maintained there and propagated to the subsystem automatically.

5.3 Designing Edge AI Systems with MBCSE Methodology

Edge AI systems have a special character. They are power and computing resource limited and usually have hard real-time requirements. At the same time, they apply multiple cascaded, or nested, AI algorithms. Therefore, they are perfect candidates for model-based design methodology described in Figure 5.2.

The design process begins in the traditional AI algorithm development environment like TensorFlow, Caffe, or PyTorch. Once the algorithm is optimized for accuracy and mathematical complexity, the network is translated to C++ using one of the Python-to-C converters. The generated C++ code is validated against the original Python model to ensure its correctness.

The same floating-point C++ model is a functional description of the AI system functionality and is used as a starting point for the architecture analysis and development. In the functional analysis the AI algorithm can be treated as one entity, but for a more detailed analysis it must be broken down to smaller entities that usually are the individual layers of the neural network. Arcadia methodology enables function breakdown into any granularity. For example, in the top-level system context the AI algorithm can be one function, which is broken down into the individual layers in the lower subsystem level. Additional information like kernel size or number of channels can be added to the model as properties.

In the logical architecture phase, the functions are allocated to logical components and the data exchanges between them through logical channels. This is the first attempt to allocate the functionality to an implementation

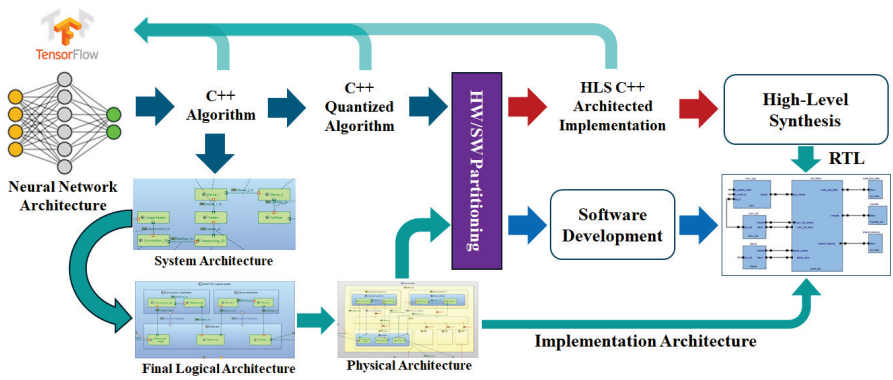

Figure 5.2 MBCSE Process for AI system design

architecture. The amount of information at this point is still very small and it is easy to change the logical allocation by just assigning a function to another logical component. This architecture exploration phase enables a performance simulation to analyse the latencies, interconnect traffic, processor utilization, and many other metrics. The performance model can be parameterized to test what-if exploration of design alternatives as shown in Figure 5.3. Here, the convolution function is changed from a SW implementation to a HW accelerator resulting in a 96% reduction in the latency of the computation without changing the performance model.

The purpose of the performance analysis is to find the optimal HW/SW allocation and simultaneously design constraints for the HW designers. When the optimal architecture is found, a physical architecture model is created. It contains more detailed implementation information for creating a virtual model for more detailed analysis and HW implementation.

Parallel to the architectural exploration, the C++ functions targeted to HW acceleration are quantized by using fixed-point data types to model the HW implementation effects to the AI algorithm. This quantized C++ code is validated against the original Python model using the same testbench as earlier for functional validation.

In the next step the quantized C++ code is partitioned into HW models that are further optimized for HLS, while functions that remain as SW will go through the standard SW development process.

The HW platform integration is based on the physical architecture model. The components, ports, and interconnects can be extracted from the system

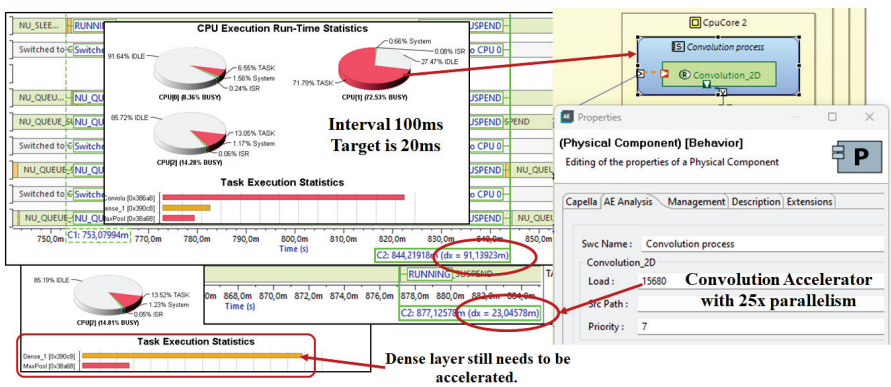

Figure 5.3 Performance exploration

model and refined in the domain specific design environment. Standard components like processors, memories, and peripherals can be taken from the library, while custom models are synthesized to RTL using HLS.

When the platform is integrated, it goes through the domain specific verification flow.

5.4 Creation of Bespoke AI Accelerator

Once the MBCSE analysis shows that a bespoke AI accelerator is needed, whether deployed as an ASIC or FPGA, a hardware design project must commence to design, verify, and integrate that hardware accelerator. MBCSE provides a set of requirements and constraints for the AI accelerator. It also defines the data flows, both the inputs and outputs, to the larger system. This constrains the interfaces used for the implementation of the AI accelerator.

While the deployment of the accelerator must meet the constraints derived from the system analysis, there is still a great deal of latitude in the implementation. The nature of hardware design is such that aspects of the design, such as power consumption or throughput, could vary by an order of magnitude or two, while meeting the remaining constraints on the system. This means that it is critical to understand both the constraints as well as the limitations of the constraints. For example, there may be a requirement that an inference must be completed in 2 milliseconds. This is a clear performance requirement. But this computation may be performed in parallel with another calculation that takes, say, 1.5 milliseconds, where the slower of the two limits the overall throughput. Or the collection of the feature data from sensors may take 1.5 milliseconds. In these cases, there will be no benefit to making the AI accelerator go any faster than 1.5 milliseconds. Implementing a faster accelerator will typically degrade other system characteristics, such as power consumption or silicon area. However, for some systems, achieving higher performance or lower power consumption is better, with no limit or point of diminishing returns. Thus, itis critical for the designer to know both the constraints and the limitations of the constraints to architect an optimal implementation. The MBCSE flow delivers this data, and it should be used to inform the design process.

With the constraint data, the hardware development effort can commence. However, a traditional hardware development cycle for either ASIC or FPGA can take months and requires a level of manual effort such that it is not practical to create multiple implementations that are meaningfully different.

In the traditional hardware design flow, an initial architecture is decided upon, and minor course corrections can be made through the design cycle. But few design teams have the resources to do true design space exploration. If the design marginally meets the requirements, it will be accepted, even if it is sub-optimal. Traditional hardware design relies on the experience and intuition to select an architecture. However, if the design team has not built a bespoke AI accelerator before, there is likely little or no experience base to draw from.

5.4.1 High-Level Synthesis

High-Level Synthesis is a technology that takes an algorithm as input and produces a target technology specific implementation in the form of a synthesizable RTL design. The input algorithm is generally in the form of C++ or SystemC, but new approaches are supporting SystemVerilog and AI framework CNNs as input [8], [9].

High Level Synthesis is a well-established technology [10], available as open-source projects and from commercial EDA and FPGA suppliers. HLS works by parsing the input algorithm and creating a global control data-flow graph (CDFG) of the algorithm. This graph is analysed to determine data dependencies, parallelism in the algorithm, and potential resource sharing opportunities. The graph is then transformed into a set of state machines and data flow constructs that exactly implement the original algorithm. From this the RTL representation is constructed. The RTL implementation requires the target clock frequency, and information about the target silicon technology. This data would be in the form of an ASIC technology data for a targeted ASIC, or for FPGAs, detailed information on the characteristics of the FPGA design elements available.

The HLS tool can create a variety of implementations from a single input algorithm. For example, a multiply/accumulate loop could be fully unrolled, with a physical multiplier for every multiplication operation. Or it could share a single multiplier instance for each loop iteration. The former implementation would be fast but would be large and inflexible. The latter would be slower but would be smaller and could more easily accommodate a varying number of loop iterations. Or, a partial unrolling could be done, giving a balance between the two. Typically, the hardware engineer will provide guidance to the tool in the form of programming “pragmas” or tool settings. This gives the developer control over the level of parallelism and pipelining in the resulting implementation, See Figure 5.4.

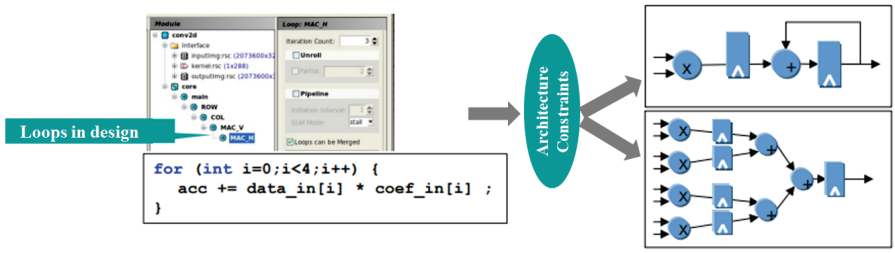

Figure 5.4 Alternative micro-architectures from a single source, based on tools settings and constraints

Since the creation of each RTL implementation is automated, the creation of several meaningfully different design options can be done quite quickly, in hours instead of months. In a traditional design flow, this would be prohibitively expensive. Each implementation can then be evaluated for power, performance, and area (PPA). From the scheduled state machine construction, the HLS tool can accurately calculate the latency for the algorithm for a given implementation. With knowledge of the target ASIC technology or FPGA device the HLS tool can ascertain the specific design primitives used to construct each design alternative which provides a good basis for determining the area. When combined with design activity (switching data), this can be used to produce an estimate of the power consumption. This power estimate will include the static power for each gate, as well as the dynamic switching power. But it will not include the parasitics, which are only available after RTL synthesis and place and route. While the power estimates will have some degree of uncertainty, they are still useful for performing comparisons between different implementations.

5.4.2 Implementation and optimization with HLS

With an HLS flow in place, deploying a bespoke accelerator is more than just running the HLS tool. There are several modifications that can be done that will improve the operating characteristics of the accelerator. These will result in an implementation optimized for the specific application. The first is to reduce the size of the network as much as possible. This involves well understood pruning techniques. Often a trained network can be reduced by 90% or more with little to no reduction in accuracy [11].

Next would be select an optimal numeric format, or "quantizing" the network. Using a more compact numeric representation will reduce the size

of memory needed for weights, biases, and intermediate results. It will also reduce the size of the operators that process that data. Multipliers will take up the most area of the computational logic of the accelerator. And memory, used for storing weight, biases, features, and intermediate results will take up the most area and consume the most power. In terms of performance, movement of the data to and from the processing elements will be the biggest performance bottleneck.

In machine learning frameworks running on general purpose computers or GPUs, numbers are stored and operated on as 32-bit IEEE format floating-point numbers. But, for most AI processing, values usually small in magnitude, with a significant portion being between -1.0 and 1.0. Which means that a 32-bit floating point representation is excessive, resulting in significant unused range, which is wasteful in the design. Eliminating these makes the design smaller and more efficient. AI algorithms usually do not need to dynamic range supported by floating point numbers. Converting to a smaller numeric format such as fixed point, reduced precision floating point representations like bfloat16 [12] or posits [13] will result in a reduction in memory storage requirements, faster movement of the data, and smaller operators in hardware. Using quantized aware training, the size of representation can often be reduced by a factor of 4 or more [14]. By moving from a 32-bit floating point number to an 8-bit fixed point number will reduce the size of the multipliers by $\sim$97%, with a roughly proportional reduction in the power consumption. In an HLS flow the developer can choose to create a much smaller and more efficient design, or the designer could use the silicon area to add more multipliers and execute them in parallel, delivering increased performance.

By changing both the neural network architecture, through pruning, and the underlying math, through quantization, the accuracy of the network will change. Thus, it is critical that throughout the pruning and quantization steps that the accuracy is verified. Quantization and pruning actions will impact each other and can be traded-off against each other. For example, reducing the size of features or weights will reduce the accuracy of the neural network. Increasing the number of layers or channels within each layer will increase the accuracy of the neural network. Careful balancing of the pruning and quantization will result in an optimal implementation. There are AI approaches being researched for finding an optimal quantization and neural network architecture, as this is an unbounded search space it is an ideal application for neural networks [15].

While optimizing the neural network, it is important to consider the impact of these changes on PPA. HLS tools that provide PPA estimates, as described earlier, allow the developer to make informed trade-offs between micro-architectural alternatives in a way that is simply not possible in a traditional RTL design flow.

5.4.3 Verification and integration

Once the accelerator is created, the next step is verification of the accelerator and integration into the larger system. Verification can be done with both formal methods and dynamic verification. One approach is to formally prove equivalence between the algorithmic description and the synthesized RTL. Some HLS tools provide this capability. However, it is unusual for formal equivalence to fully verify the accelerator. To complete the verification, the algorithmic model can be instrumented to collect stimulus and expected responses that can be used in dynamic simulations. The combination of static and dynamic approaches will often reach full verification coverage much faster than manual methods.

Integrating the accelerator into the larger system means creating bus and memory interfaces to the external circuitry, in addition to wired connections. Most HLS tools have interface synthesis, which will create commonly used bus protocols and supports memory interfaces. These will be supported for both pre- and post-synthesis models, enabling common testbenches and verifications strategies across abstraction levels.

5.5 Exemplary Results

To illustrate these concepts, we describe an exemplary system where this design flow is applied. The system is a handheld address scanner that reads postal codes off address labels on letters and packages. There are multiple design domains in this project: the physical design of the package/enclosure, the design of the printed circuit board (PCB) electronic sub-system, and a system-on-chip (SoC). All these design domains need to support multiple requirements: weight, battery life, performance, accuracy, communications, security, and more. Balancing these requirements and demands require a holistic approach that embodies the multiple design domains and facilitates communication between divergent design teams.

For the electronics subsystem implementation there are three options: using a standard processor on a PCB, using a standard microcontroller

and FPGA for accelerating some functions, or a custom SoC. Using the performance simulator in the cybertronics module level, which contains all electronics and SW functions, the design space was narrowed down to SoC. A standard processor solution can't meet the timing and power consumption requirements. Also, the thermal issues preclude this option. Using a microcontroller and FPGA would meet the timing requirements, but the power consumption of the FPGA exceeds the required limits, therefore the only implementation option is an SoC.

The SoC implementation was thoroughly analysed using the performance simulation and the optimal architecture contains a microcontroller and two bespoke accelerators: one for convolution and max pooling and another for the two dense layers. Based on the performance analysis, the convolution accelerator can use 4 milliseconds for one feature map and the dense accelerator 3 milliseconds including data transfer times. The HLS design space exploration was used to optimize the latency, area and power consumption. Because the data and weights are stored locally in the accelerators, the processing time is dominant in the latency and the target latency can be reached with a minimum resource implementation.

5.6 Conclusion

Complexity of the AI algorithms in the Edge systems is growing rapidly. Using general purpose processors or GPUs is no more practical because of the high computational load, power consumption limitations, and timing constraints. Most of the current design tools and methodologies focused on automating the flow from AI algorithm development to implementation, but they do not take in account the system-level aspects. The edge systems require a careful analysis of the different implementation options, which needs a new holistic approach that can handle the complexity of the system level architecture.

Model-based cybertronics systems engineering methodology enables the analysis and partitioning of the algorithms on multiple hierarchy levels enabling distribution of algorithms across a multi-device system and optimizing the device architecture to provide optimal HW resources for the execution of the given algorithm. The C++-based modelling approach allows free function allocation to different processing elements and creating bespoke accelerators using HLS. This capability streamlines the development process and enables late changes of function allocation between HW and SW.

Acknowledgements

Special thanks to Ahmed Hamza, Brendan Hall and Susanna Solanti-Iltanen for their contribution to the methodology development.

References

[1] "Unified Modeling Language Specification Version 2.0", https://www.omg.org/spec/UML/2.0/, July 2005

[2] M. Eigner, C. Muggeo, T. Dickopf, K.-G. Faißt, "An Approach for a Model Based Development Process of Cybertronic Systems", *58th Ilmenau Scientific Colloqium,* Ilmenau, Germany, Sept. 2014

[3] P. Feiler, B. Lewis, S. Vestal, E. Colbert, "An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering", *IFIP World Computer Congress*, 2004

[4] P. Solanti, R. Klein, "Model-Based Approach for Developing Optimal HW/SW Architectures for AI systems", *DVCon Europe 2023*, Munich, Germany, Oct. 2023

[5] J.-L.Voirin, "Model-based System and Architecture Engineering with the Arcadia Method", *ISTE Press,* London & Elsevier, Oxford, 2017

[6] P. Micouin, "Property-Model Methodology: A Model-Based Systems Engineering Approach Using VHDL-AMS", *Systems Engineering,* Vol. 17-3, pp. 249–263, 2014

[7] S. Solanti-Iltanen, B. Hall, P. Solanti, "Model-Based Cybertronics Systems Engineering (MBCSE)", *INCOSE International Symposium,* Dublin, Ireland, July 2024

[8] D. Chen, "Lightning Talk: The Next Wave of High-level Synthesis," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2023, pp. 1-3, doi: 10.1109/DAC56929.2023.10247880.

[9] L. E. Pozzoni, F. Ferrandi, L. Mendola, A. A. Palazzo and F. Pappalardo, "Using High-Level Synthesis to model System Verilog procedural timing controls," *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10136907.

[10] A. Takach, "High-Level Synthesis: Status, Trends, and Future Directions," *IEEE Design and Test*, April 1^{st}, 2016, Volume 33, pp. 116-124

[11] S. Han, H. Mao, W. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding," May 2^{nd}, 2016, *International Conference on Learning Representations (ICLR 2016)*

[12] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, andD. Mansell, "Bfloat16 processing for neural networks," in *IEEE 26thSymposium on Computer Arithmetic (ARITH*), 2019

[13] G. Genkina "Posits, a new Kind of Number Improves the Math of AI", *IEEE Spectrum*, September 2022

[14] C. Coelho, et al. "Automatic deep heterogeneous quantization of Deep Neural Networks for ultra-low area, low-latency inference on the edge at particle colliders," *arVix:2006.10159v2*, 11/2020

[15] H. Cai, C. Gan, T Wang, Z. Zhang, S. Han, "Once-for-All: Train One Network and Specialize it for Efficient Deployment," *arVix:1908.09791v5*, 8/2019