applied sciences

MDPI

Article

Speed Optimization in DEVS-Based Simulations: A Memoization Approach

Bo Seung Kwon [1], Young Shin Han [2,*] and Jong Sik Lee [1]

[1] Department of Computer Engineering, Inha University, 100 Inha-ro, Michuhol-gu, Incheon 22212, Republic of Korea; meno9428@gmail.com (B.S.K.); jslee@inha.ac.kr (J.S.L.)
[2] Frontier College, Inha University, 100 Inha-ro, Michuhol-gu, Incheon 22212, Republic of Korea
* Correspondence: hanys@inha.ac.kr

Featured Application: This study is proposed to improve performance in order to simulate a system model using DEVS methodology, which requires a large hierarchical structure like HDL simulation.

Abstract: The DEVS model, designed for general discrete event simulation, explores the event status and time advance of all DEVS atomic models deployed at the time of the simulation, and then performs the scheduled simulation step. Each simulation step is accompanied by a re-exploration the event status and time advance, which is needed for maintaining the casual order of the entire model. It is time consuming to simulate a large-scale DEVS model. In a similar vein, attempts to perform an HDL simulation in a DEVS space increase simulation costs by incurring repeated search costs for model transitions. In this study, we performed a statistical analysis of engine behavior to improve simulation speed and we proposed a DP-based memoization technique for the coupled model. Through our method, we can expect significant performance improvements that range statistically from 7.4 to 11.7 times.

Keywords: DEVS formalism; simulation speedup; scheduling; memoization

Citation: Kwon, B.S.; Han, Y.S.; Lee, J.S. Speed Optimization in DEVS-Based Simulations: A Memoization Approach. *Appl. Sci.* **2023**, *13*, 12958. https://doi.org/10.3390/app132312958

Academic Editor: Benoit Iung

Received: 8 November 2023
Revised: 25 November 2023
Accepted: 28 November 2023
Published: 4 December 2023

1. Introduction

Discrete event system specification (DEVS) formalism is a methodology that precisely models the behavior of complex hierarchical systems, which describes state changes around the exchange of messages with sources for irregular events occurring inside or outside the system, which provides a clear understanding of the characteristics and patterns of various system behaviors. The powerful benefits of these event-based simulations allow DEVS formalism to find applications in a variety of areas, including communication models, logistics systems, production, transportation, and war games [1,2].

The beauty of DEVS formalism stems from the fact that we can model more complex and realistic systems by combining time flow and event-driven transition rules based on a finite state machine. Because the model can be modularized into certain units, hierarchical system structure is possible; DEVS-based modeling allows large-scale systems to be divided into multiple subsystems and it is designed from the smallest units to be integrated or reused [3].

To design and run DEVS models to obtain simulation results, an execution environment that provides a hierarchical simulation algorithm is required. Open-source tools such as DEVSimPy and DEVSuite are already available [4,5]. However, depending on the domain characteristics and simulation purposes of the designed DEVS model, there is a need to lighten the simulation engine or to add complex functions. Thus, there have been various efforts to change the DEVS model structure, simulation engine and environments, and architectures to perform domain-appropriate simulation and improve performance [6,7].

Deviating from hierarchical structures and applying distributed frameworks have been popular topics, and the DEVStone metric exists as a performance measurement indicator for DEVS-based simulators for this purpose [8,9]. DEVS formalism allows almost any discrete event-based simulation to be performed, and various approaches have been attempted to design a purposeful DEVS model. Various approaches have been attempted to design fit-for-purpose DEVS models. In particular, modeling for hybrid simulations and the expansion into unnamed aerial vehicles based on Cell-DEVS have remained popular until recently [10,11].

In our previous work, we designed the RTL-DEVS model to enable HDL simulation in DEVS space and we showed that it is compatible with Verilog at a minimum level [12]. Verilog is a hardware description language for designing digital circuits and systems. It includes a module-based hierarchical design starting from basic logic circuits to complex cells, comprising a discrete event-based simulation where the state changes by signal input and output, and it has features such as time management to respond to transmission delays. Although not all of these are completely consistent with DEVS formalism, they have compatible features in many areas, so they have been used in a form such as DHMIF (DEVS-based hardware model interchange format), which is a hardware exchange format [13]. Table 1 describes equivalent operation characteristics and incompatible characteristics of DEVS and Verilog.

Table 1. Compatible and incompatible features of DEVS and Verilog.

Compatibility	Function	Description
Functionally Compatible	Event-based behavior	DEVS: Discrete event-based. Verilog: Response to digital signals.
	Modulization	Support modularized structure.
	State-based behavior	Operates around transitions between states.
	Time-driven	Delay; timer; event scheduling for system behavior modeling.
Functionally Incompatible	Language structure	DEVS: Set theoretical formalism. Verilog: Programming language style syntax.
	Scheduling mechanism	DEVS: Complex event scheduling and processing mechanisms. Verilog: Operates according to signal changes.

Previously, there have been continuous efforts to supplement RTL simulation or reduce its cost based on DEVS formalism [14,15]. Due to cost, training, and scalability issues, there have been various attempts with Verilog compatibility simulations, including MyHDL and PyMTL3 [16,17], with a desire in the research community to stop using the commercial simulator and instead perform RTL simulations through discussion and collaboration in the open-source community; for example, studies have used the open-source toolkit Pyverilog for prototyping [18]. Performing the simulation of register transfer level circuits using DEVS formalism has the advantage of allowing the use of simulations through a simplified format prior to full-scale design and simulation using commercial tools. Simplified simulations in DEVS space can only be performed on each model when the next event occurs, i.e., when clock changes in the hardware occur. Therefore, if a completely identical simulation environment based on DEVS formalism was designed, it would ideally exhibit a significant cost reduction. Of course, performing a double HDL simulation at the level possible with an open-source simulator increases costs. Therefore, we are developing a DEVS-based simulation environment compatible with Verilog for function libraries that are difficult to develop or to apply in open-source simulators. Our goal is to develop a co-simulation system that performs expensive functions in the RTL-DEVS environment that are relatively easy to develop, without requiring the use of commercial RTL design tools or the development of additional branches of open-source simulators.

Therefore, even for incompatible features, it must be ensured that the results of simulation in DEVS space are equivalent to the results of simulation in the Verilog simulator. It requires a stepping stone to provide compatibility between Verilog and DEVS. It has

been verified through various research and joint development projects that the Verilog description can be transmitted to other tools or FPGA tools through XML [19,20], and that the biggest problem, the difference in language structure, was resolved through the direct development of an XML-based Verilog parser. However, the DEVS simulation engine's scheduling mechanism is more complex than Verilog's, and a serious level of overhead occurs depending on the node depth of the layered modules. Recently released high-performance semiconductor chips have already become complex in structure, containing anywhere from billions of transistors to over 10 billion depending on their purpose. To design and verify such complex chips, a vast amount of logic and modules are required, and the code complexity of Verilog has also increased.

Due to the module complexity of the RTL design stage, numerous atomic models and coupled models are used when performing an HDL simulation with a DEVS-based simulation engine. The entire model is designed hierarchically, so one module (coupled model) contains another coupled model, and this is repeated recursively. When performing a simulation based on an existing simulator or scheduling algorithm, the time for simulation increases exponentially depending on the complexity of the model.

In other research, the issue that hierarchical simulations generate significant overhead depending on their depth was already raised, and performance improvements of up to seven times were obtained by using event lists and flattening [21,22]. Extracting recurring patterns from systems that use event-based state transition models can have a significant impact on performance improvements [23]. However, for an HDL simulation, the extended RTL-DEVS model uses two buffers to process RHS and LHS separately, and the wire model that connects logic also has one buffer. In this process, an event with a lifetime of 0, which is a scenario called zero-time transition, occurs repeatedly to ensure signal transmission without delay. As a result, a bottleneck occurred in the process of searching for the node that should perform the next event. The challenge is to improve simulation speed while maintaining the hierarchical structure of the model. The memoization method, which caches the results of a function call and allows for a quick response to the same input, is a very classic technique. Nevertheless, due to the high efficiency of the idea, it continues to be actively discussed in the overall hardware/software field [24].

In this paper, to solve this problem, we analyzed the behavior of existing simulators using statistical methods and modeled the correlation between the DEVS model configuration and the simulation time.

As a result of modeling and simulation, we confirmed a linear increase in simulation time depending on the depth of the coupled model. Based on statistical insights, we propose an improved simulation model structure where a memoization technique is applied based on dynamic programming. Section 2 provides a brief introduction to DEVS and describes the RTL-DEVS architecture. Section 3 covers the existing DEVS simulation engine, and Section 4 describes our improved DEVS simulation engine applying the memoization technique and discusses the experimental results. In Section 5, we explain how the memoization technique reduces the search cost in a DEVS coupled model. Finally, Section 6 concludes the study and also briefly introduces future research directions.

2. DEVS Methodology

DEVS (discrete event system specification) is a formalized modeling methodology for discrete event systems proposed by P. B. Zeigler [25]. This methodology clearly expresses the structure and behavior of the system mathematically, modularizing and combining individual models into objects in a hierarchical framework based on set theory. Each DEVS model consists of components such as states, inputs, outputs, state transition functions, and timer functions, making it particularly suitable for event-based simulation. The basic DEVS formalism includes an atomic model, which is the minimum unit of the system and can no longer be decomposed, and a coupled model that can be expressed by combining the atomic model and another coupled model into one module.

2.1. Atomic Model

The atomic model is always the base model of the hierarchy in DEVS and represents the behavior of the system. The mathematical expression of the atomic model M is as follows:

$$\begin{gathered} M = <X, S, Y, \delta int, \delta ext, \lambda, ta> \\ X \text{ is the set of input values.} \\ S \text{ is a set of states.} \\ Y \text{ is the set of output values.} \\ \delta int: S \to S \text{ is the internal transition function.} \\ \delta ext: Q \times Xb \to S \text{ is the external transition function.} \\ Q \in \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\} \text{ is the total state set, e is the time elapsed since} \\ \text{last transition, and Xb denotes the collection of bags over X.} \\ \lambda: S \to Yb \text{ is the output function.} \\ ta: S \to R + 0, \text{ is the time advance function} \end{gathered} \tag{1}$$

The atomic model expresses the state of the target system as a set S. In general, if there is no external input X during the time advance inside the model, the λ function is executed and the output Y is sent. Afterwards, the internal state transition function δint is executed to change the state of the current model, and the time advance for when the next output will occur is determined through the ta function. The output Y acts as input X to another atomic model, and if input X occurs during a time advance of a fixed state, the external state transition function δext is executed to change the state and perform a predetermined process.

2.2. Coupled Model

The coupled model internally connects multiple atomic models or coupled models. Through this, a larger system can be expressed hierarchically in the form of relationships between nodes and child nodes as a tree structure. The mathematical expression of the coupled model DN is as follows:

$$\begin{gathered} DN = < X, Y, D, EIC, EOC, IC, SELECT > \\ X \text{ is the input events set.} \\ Y \text{ is the output events set.} \\ D \text{ is the set of all component models in DEVS.} \\ EIC \subseteq X \times \cup i X_i \text{ is the external input coupling relation.} \\ EOC \subseteq \cup i Y_i \times Y \text{ is the external output coupling relation.} \\ IC \subseteq \cup i X_i \times \cup i Y_i \text{ is the internal coupling relation.} \\ SELECT: 2^M\text{-}\varphi \to D \text{ is a function which chooses one model when} \\ \text{more than 2 models are scheduled simultaneously.} \end{gathered} \tag{2}$$

EIC (external input coupling) describes all possible couplings between the input X coming from outside the coupled model and the input X_i of each internal model. Similarly, external output coupling (EOC) represents the coupling of the entire output event set Y of the coupled model with the event output Y_i of each internal model. IC (internal coupling) expresses the coupled relationship between two atomic models within a coupled model or other coupled models, and it represents the input event set X_i and output event set Y_i of all internal models. SELECT determines the priority of event processing for a set of models that are simultaneously active at the same time due to input/output events. If more than one model fires events at a time, the order in which the events are processed can cause confusion.

2.3. Coordinator for Hierarchical Simulation

Models written in DEVS formalism are executed using a simulation algorithm that solves the DEVS model structure and helps communicate hierarchically. Figure 1 shows

the relationship between the DEVS model and the DEVS-based simulator. In the DEVS Model space, {ABCD, AB, CD} describes the coupled model, and {A, B, C, D} describes the atomic model. In the simulator, the notation C: describes the coordinator corresponding to each coupled model, and {A′:D′} describes the atomic model on which the simulation is performed. Where C describes the coordinator assigned to each coupled model, and the coordinator is mapped 1:1 to each layer of the DEVS model. In conclusion, the atomic model performs a simulation, and the coordinator corresponding to the coupled model processes the message exchange between each model based on the internal relationships of the DEVS model. The coordinator will be discussed further in Section 3.

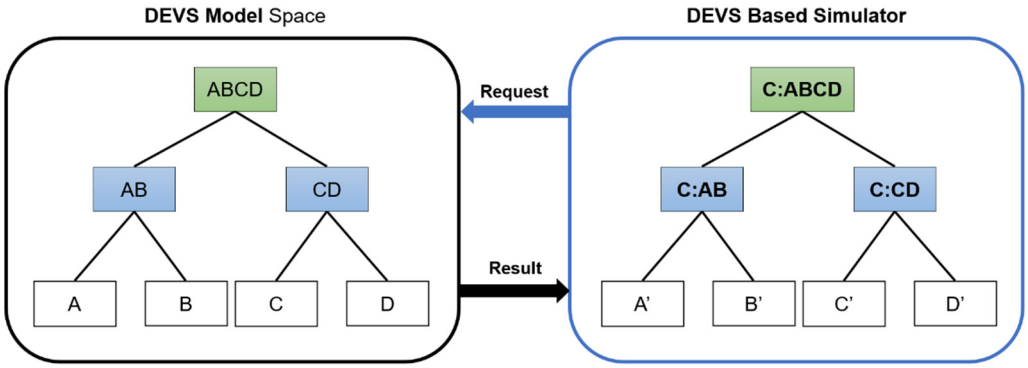


Figure 1. Execution environment using hierarchical scheduling.

3. HDL Simulation Using DEVS Methodology

In our previous research, the main proposal was how to connect the Verilog module to the DEVS atomic model. Because Verilog and the basic DEVS model have different structures, we first had to create an RTL-DEVS atomic model that satisfies HDL requirements so that the input and output of the digital signals are compatible [8].

3.1. RTL-DEVS Atomic Model

Figure 2 explains the single RTL-DEVS atomic model. When digital logic consisting of 0 or 1 is transmitted to the model's event input X, the model stores the value of the corresponding signal in the register. The input event causes the DEVS model's δext to be performed, resulting in internal transition behavior. Two main functions operate during the internal transition process: (1) A check of whether the digital signal currently contained in the register is the same as the value of the previous event input X, which was the condition for δext. If a new event input of the existing input and the current register input are not the same, the model does not update the state of the state buffer connected to the output unit, so new values are not output. (2) For compatibility with Testbench, each model has an edge checker buffer. When the current state of the state buffer is reversed, it is processed as rising or falling.

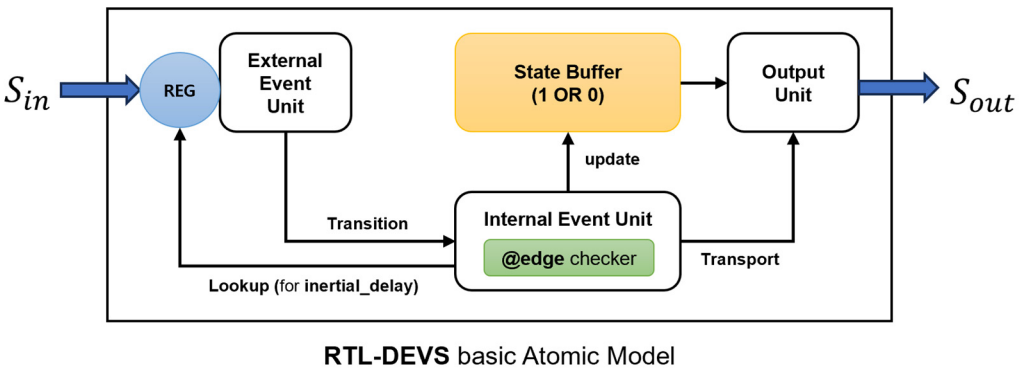


Figure 2. RTL-DEVS atomic model structure.

Figure 3 explains an example of a 4-bit full adder in the RTL-DEVS model where a simple logic module is expressed as an atomic model. In previous research, we tested some models that could serve as basic modules to handle multiple inputs and outputs by expanding the internal registers of the atomic model into vector form to reduce simulation costs. There were no major problems with these simple logic gates or simple combinational logic circuits, but problems arose while developing the Verilog parser and performing larger-scale simulations to check compatibility. The wire connecting each atomic model is not implemented separately, but a message I/O function, commonly called a bag, is used in the DEVS engine to enable the transmission of electrical signals. The number of atomic models is increasing, and a zero-time transition, in which one atomic model has a time advance of 0, generates continuous input and output, resulting in too much message I/O at the same time, resulting in confusion in the message transfer interface of the DEVS engine and no longer ensuring proper operation.

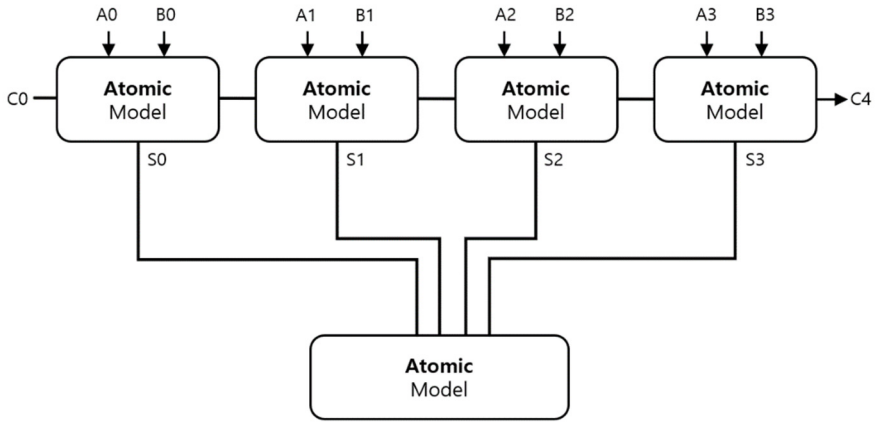
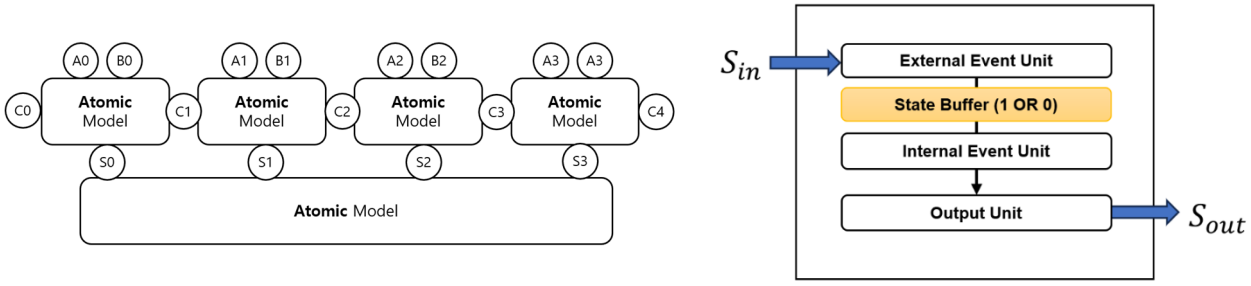


Figure 3. 4-bit full adder expressed as RTL-DEVS.

3.2. Wire Atomic Model

To solve this problem, rather than arranging the atomic model on a module basis as shown in Figure 3, we converted individual wires and entire modules to the atomic model. {A0:B3}, {C0:C4}, and {S0:S4} mean wire connections between models. In Figure 4, a total of five RTL-DEVS atomic models are represented. You can see that all wire connections in Figure 3 have been modified to the wire atomic model. However, unlike the existing model, which was previously implemented as an atomic model only for modules requiring regs and wires, the new model requires each wire to also be implemented as an atomic model. In this way, each wire atomic model continues to maintain the existing value until 0 or 1 is received as the next input, and signals can be transmitted to the connected atomic model without confusion in the signal transmission order according to the DEVS simulation step. Additionally, in this process, setting the latency of individual modules becomes much easier than before. Previously, in order to output the current state of a wire in a testbench, an individual RTL-DEVS atomic model had to output the current state, but by expanding the wire itself into an atomic model, a monitor function can be added to a specific wire.

In the case of the improved model, for complex models, it now shows the same behavior as the Verilog simulation without confusion about message processing in the bag, but the disadvantage is that simple logic, which was previously expressed as one atomic model, must be expressed as two separate models. As the number of modules contained in the coupled model increases, the time required for simulation also increases linearly, so a solution to this problem was required. To solve these performance problems, an event list-based method of flattening all models with connectivity relationships has been proposed, and nearly seven times the performance of actual existing DEVSim++ has been reported, but to meet HDL requirements, hierarchical simulation conditions must be met.



(**a**) (**b**)

Figure 4. RTL-DEVS 4-bit full adder with wire atomic models. (**a**) Example of a wire-extended RTL-DEVS model; (**b**) internal structure of a wire atomic model.

4. Analysis and Experiments on the DEVS Simulation Engine

4.1. Simulation Design

To find the cause of the problem of excessively increasing simulation time, model performance evaluation was performed by coupling 10,000 basic atomic models in various ways based on the wire model. The model used in the experiment is shown in Figure 5 below. In Figure 5c, {C0:C10} each model means a coupled model of (a) and (b), and {C′0:C′10} models of (d) mean a coupled model of (c).

CLK atomic model (Testbench): It outputs rising (true) and falling (false) signals repeatedly at regular intervals. It is deployed only on the top model of the test bench.

CLK atomic model: The model stores the signal input to the clk in-port in a 1-bit buffer and immediately outputs the buffer value to the clk out-port.

Coupled model: The coupled model is responsible for routing input and output to the child atomic/coupled model. You can design a unit for a coupled model by placing a predefined Clk model inside the coupled model, and then create various types of experimental models by linking these units in a tree data structure. The figure below shows the process by which a set of clk models, a coupled model, is recursively combined into a coupled model structure.

The hierarchical coupled model structure for testing the performance of the DEVS simulation engine suitable for the modeled wire-extended RTL-DEVS model is as follows:

1. Group 10,000 CLK models into defined units to form an initial coupled model.
2. Create a high-level coupled model by grouping these initial coupled models into defined units.
3. This process is performed recursively to complete the hierarchical coupled model structure of Group 1, Group 2, . . ., Group N. The types of models created for the experiment are shown in Table 2.

Table 2. Changes in simulation time depending on the number of coupled models.

Model Name	Atomic Model	Coupled Model *	Simulation Time(s)
{10, 10, 10, 10}	10,000	1111	188
{10, 10, 100}		1101	181
{10, 100, 10}		1011	175
{100, 10, 10}		111	73
{100, 100}		101	79
{10, 1000}		1001	168
{1000, 10}		11	56
{10000}		1	59

* Number of coupled models.

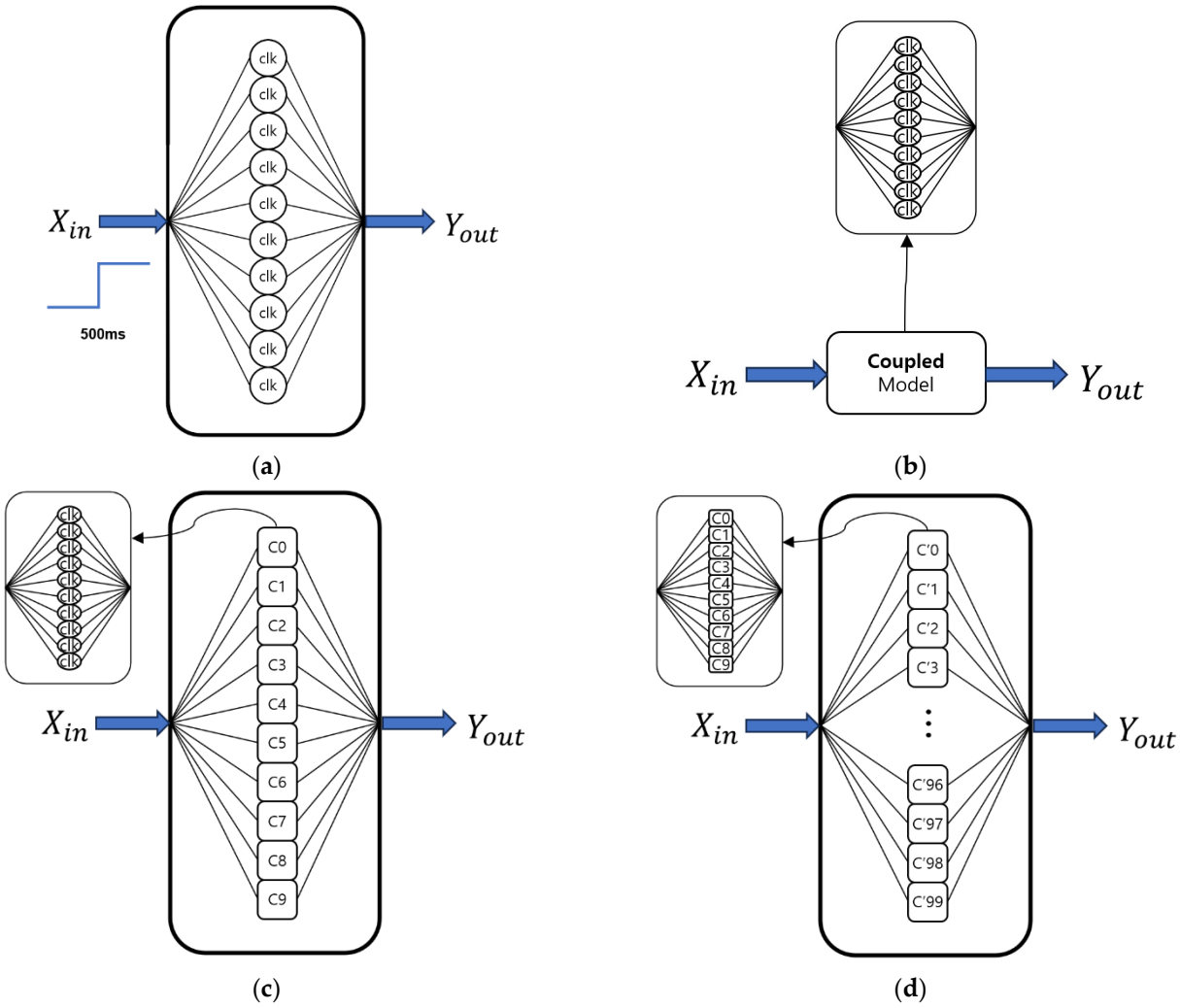


Figure 5. Coupled model structure for 10,000 atomic models. (**a**) Lowest-level coupled model. (**b**) Abstract model of the lowest-level coupled model. (**c**) Hierarchical representation of a {10:10} coupled model. (**d**) Hierarchical representation of a {100:10:10} coupled model.

For example, a model expressed as {10, 100, 10} contains 10 atomic models in 100 coupled models, and there are 10 higher-level coupled models composed of 100 coupled models.

4.2. Simulation Results and Analysis

When the total number of atomic models is the same as shown in Table 3, the Pearson correlation coefficient for the simulation time change according to the number of coupled models is 0.99702, and when the total number of coupled models is the same, the Pearson correlation coefficient for the simulation time change according to the number of atomic models is 0.99642, which can be seen as linear for both the total number of coupled models and the total number of atomic models.

Table 3. Changes in simulation time depending on the number of atomic models.

Model Name	Atomic Model	Coupled Model	Simulation Time (ms)
{10}	10	1	0.23
{100}	100		7.511625
{1000}	1000		505.214042
{10000}	10,000		59,600

In simulations based on DEVS methodology, the simulation model is determined according to the domain to be simulated, and the atomic model has the characteristic of being user-dependent. We extended the DEVS model to the RTL-DEVS model to perform an HDL co-simulation. However, it is the coupled model that connects each RTL-DEVS model, and the simulation result showed a clear increase in simulation time according to the number of coupled models, so it is necessary to improve the internal process of the coupled model to improve the simulation time from the perspective of the entire system.

4.3. Coordinator in the DEVS Simulation Engine

The DEVS-based simulation engine operates predictably and consistently when the life cycle (time advance) of each model is clearly defined and an accurate FSM (finite state machine) is defined. Performance of hierarchical simulations in the DEVS engine is controlled by a major unit called the coordinator, illustrated in Figure 1. The coordinator plays an important role in controlling the behavior and event processing priorities of the atomic model and the coupled model. The process of having the coordinator conduct the simulation can be divided into four steps.

Figure 6 shows the phase in which the coordinator finds and selects the child model with the smallest time advance within the coupled model. In the case of an atomic model, the time for the model transition is reserved, so time advance can be found by approaching constant time.

However, in the case of a coupled model, it is necessary to compare and reduce the time advance of the child node, so a graph search is performed to find and compare the atomic model with the least time advance. For models with equal time advance, the total order is determined through the SELECT function.

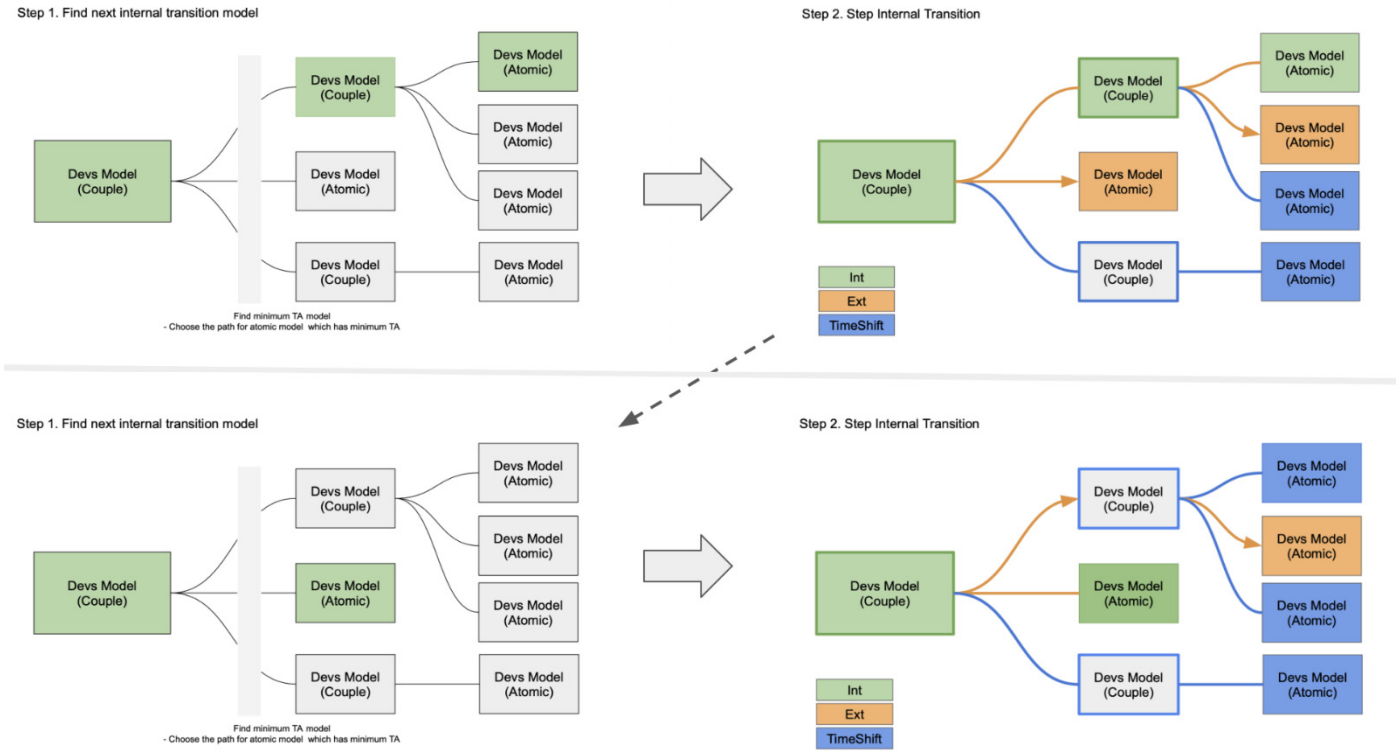


Figure 6. Process of the coordinator conducting the simulation.

After performing the internal transition of the DEVS model, which is displayed in green, the external transition occurs for internally coupled models, and then the time advance of the remaining models is advanced.

A problem with the existing simulation engine occurred when selecting the target model for internal transition in Figure 6.

To find the model with the least time advance, you need to know the total order for the entire DEVS model, which requires a recursive graph search, so the higher the complexity of the simulation, the higher the linear increase in the time spent on that search, as shown in Table 2. In the designed simulation environment, whenever 10,000 individual atomic models perform an internal/external transition, a time advance inquiry is requested of the connected atomic model for the entire coupled model.

This causes a very serious cost problem in the RTL-DEVS simulation model environment, where it is necessary to process a short time or zero-time advance for processing electrical signals.

To improve this problem, there is a need to improve the time spent on graph exploration. Techniques such as priority queueing and memoization can be introduced. We want to propose a solution to the problem from the perspective of dynamic programming by utilizing the memoization technique to reduce the time spent on the graph exploration of the coupled model.

5. Memoization Technique to Reduce the Search Cost in the DEVS Coupled Model

5.1. Basic Approach

In step 1 for the internal transition, if the time to find the time advance of the coupled model can be reduced to a constant time, the simulation time can be shortened by reducing the search time. To apply this method, a function to memoization the TA of the coupled model can be added to the simulator when the TA of the coupled model is updated. Having the TA of the coupled model stored in the coordinator has more advantages than expected when considering its simplicity. There are three factors that determine the model's time advance: internal transition, external transition, and time shift.

Internal Transition: In the case of the internal transition, the atomic model determines the time advance in constant time according to the change in FSM. The coupled model performs the internal transition on all sub-models, then external transition on connected models, and then the time shift operation on the remaining models. After this processing process is completed, the TA of all child nodes is calculated and the minimum value is reflected in the memoization.

External Transition: In the case of internal transitions, the atomic model determines the time advance in constant time according to the change in FSM. Coupled model performs transition operations of event target models, receives time advance values, and reflects the minimum value in the memoization.

Time Shift: When internal transition and external transition occur, a simple time shift operation is performed on all unconnected child nodes. The minimum value is reflected in the memoization.

5.2. Model Transition Behaviors Using the Memoization Technique

The internal transition (coupled model) transition algorithm shown in Algorithm 1 and the external transition (coupled model) algorithm shown in Algorithm 2 represents the pseudocode of external and internal transitions. In the case of the time shift operation, it was omitted because it simply performs a function that subtracts the time advance of the model. time_advance() performs the function of receiving the TA of the model. self.select() describes the ability to receive priority target models at the current simulation time.

Internal_transition() returns the output Y of the target model and receives the current model and the coupled model. In this process, models with X_in connected to Y_out are assigned to the cyx_group, and models with X_in connected to X_in of the coupled model are assigned to the cxx_group.

Algorithm 1: Internal Transition (Coupled Model)

Result: Update min_ta of the coupled model with the value of self.ta_cache

```
1.  ta = self.time_adavanced();
2.  target = self.select();
3.  output = target.internal_transition();
4.  all_group = self.childrun();
5.  cyx_group = self.cyx_group(target);
6.  oth_group = all_group−
7.  cyx_group—target;
8.  min_ta = target.time_advanced();
9.  min_ta = cyx_group.external_transition(ta, output).reduce(min_ta);
10. min_ta = oth_group.time_shift(ta).reduce(min_ta);
11. self.ta_cache = min_ta;
```

Any other group is placed in oth_group. After traversing the connection model cyx_group and calling the external_transition() function, min_ta is compared with the time advance of the models in cyx_group that have completed the transition, and the smallest value is stored in min_ta. It traverses the remaining models oth_group and reduces oth_group.ta by the ta of the coupled model, compares the time advance of the reduced models, and stores the smallest value in min_ta.

Algorithm 2: External Transition (Coupled Model)

Input: Event event.
Result: Update min_ta of coupled model with the value of self.ta_cache.

```
1.  all_group = self.children();
2.  cxx_group = self.cxx_group(event);
3.  oth_group = all_group—cxx_group;
4.  min_ta = self.time_advanced()—shift;
5.  min_ta = cxx_group.external_transition(shift, event).reduce(min_ta);
6.  min_ta = oth_group.time_shift(shift).reduce(min_ta);
7.  self.ta_cache = min_ta;
```

The ta_cache of the coupled_model becomes the smallest min_ta value. Figure 7 describes the situation in which the lower lever DEVS models {M'1, M'2, W1, W2} of the Coupled Model M1 all belong to the cyx_group, putting the ta of {W1, W2} in the Min_ta. Similarly, for Coupled Model M0 with {M1, M2} as the lower DEVS model, we have the ta of M1 as Min_ta.

5.3. Experiment and Evaluation

In this sub-section, we discuss the results of simulating the same set of DEVS models that were simulated in the original DEVS engine by applying the coupled model processing method of the coordinator with the memoization technique.

To find the cause of the problem of excessively increasing simulation time, the model performance evaluation was performed by coupling 10,000 basic atomic models in various ways. Table 4 summarizes the simulation results of the coupled model with memoization applied. By applying the memoization technique to the couple model, the overall simulation time speed was improved by about 10 times. The model configuration with the least performance improvement was {1000, 11}, which used only 11 coupled models. On the other hand, the configuration with the highest performance improvement was {10, 10, 10, 10}, which shows that the greater the number of coupled models and the more complex the hierarchical structure, the higher the performance improvement efficiency.

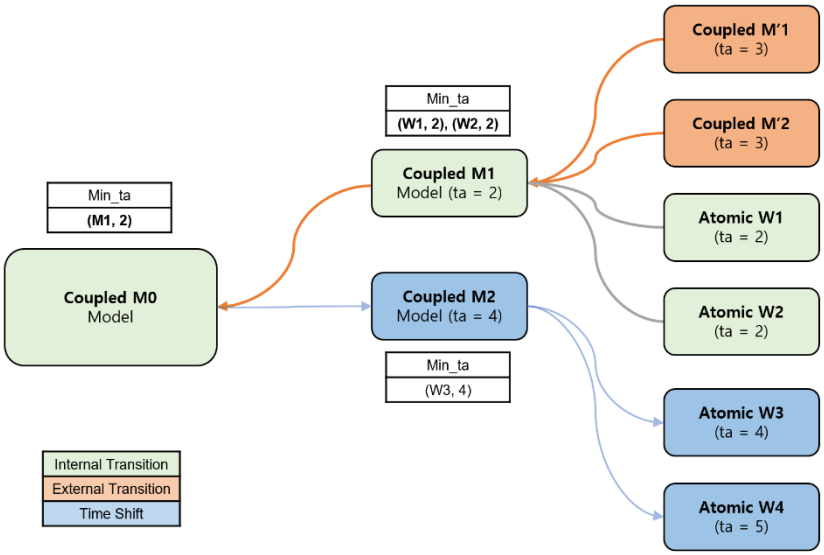


Figure 7. Min_ta is updated in DEVS models and frozen models are bypassed (time shift).

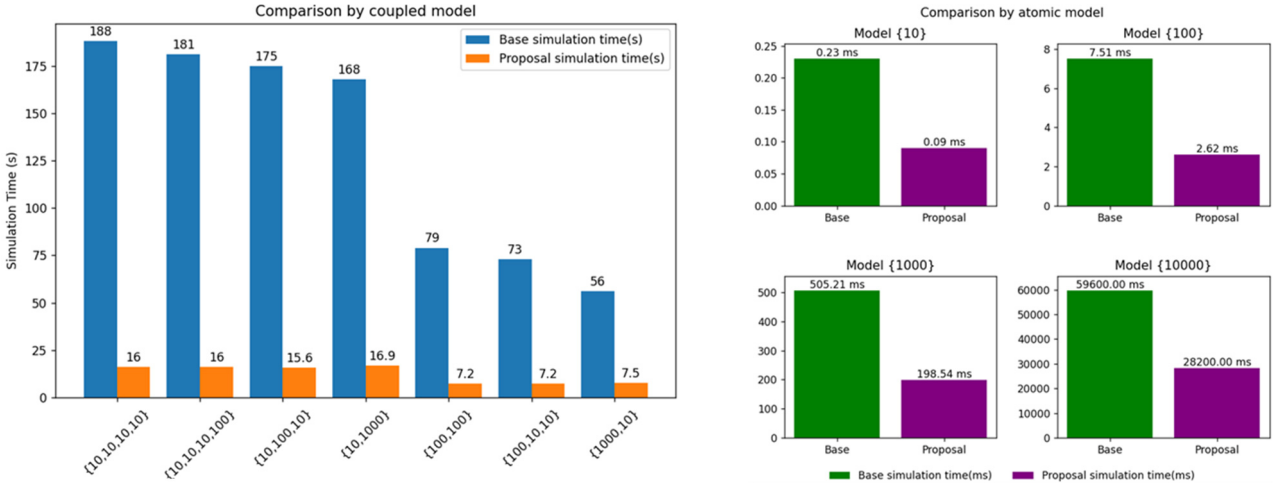
Table 4. Comparison of proposal structure simulation times and the original model (coupled).

	Atomic Model	Coupled Model	Base Simulation Time (s)	Proposal Simulation Time (s)	Improvement
{10, 10, 10, 10}	10,000	1111	188	16	×11.75
{10, 10, 100}		1101	181	16	×11.31
{10, 100, 10}		1011	175	15.6	×11.21
{100, 10, 10}		111	73	7.2	×10.13
{100, 100}		101	79	7.2	×10.97
{10, 1000}		1001	168	16.9	×9.94
{1000, 10}		11	56	7.5	×7.46

Table 5 presents the simulation results of one coupled model containing 10, 100, 1000, and 10,000 atomic models, and the model processing structure of the coordinator with the proposed memoization showed performance improvement of as little as 2.1 to as much as 2.8 times even when the child node (atomic models) is increased for one coupled model. Figure 8 visually shows the comparison results.

Table 5. Comparison of proposal structure simulation times and the original model (atomic).

	Atomic Model	Coupled Model	Base Simulation Time (ms)	Proposal Simulation Time (ms)	Improvement
{10}	10	1	0.23	0.09	×2.55
{100}	100		7.511625	2.620083	×2.86
{1000}	1000		505.214042	198.537	×2.54
{10000}	10,000		59,600	28,200	×2.11



(**a**) (**b**)

Figure 8. Simulation time comparisons of base/proposal model structures. (**a**) Visual representation of the coupled model comparison, with up to 11.7× improvement. (**b**) Visual representation of the atomic model {10} to {10000}; there is simply one coupled model, with around 2× improvement.

6. Conclusions

We have proposed the structure of the coupled model and coordinator with the applied memoization technique to improve the simulation time of DEVS and we have looked at its performance improvements.

Instead of existing algorithms that require large recursive search costs to maintain hierarchical structures and keep the casual order, time advance can be used as a cache memoization in the coupled model to achieve large performance gains for repeated I/O events in the hierarchical DEVS model.

The search and time advance function of the modified coordinator for the coupled model shows a performance difference of at least 7.4 times and up to 11.7 times compared to the conventional method, as the coupled model can return time advance without access to the sub-models via the memoization method.

However, this can show the performance differences described when extreme hierarchical simulations are required, such as the RTL-DEVS model compatible with HDL requirements, and simulations based on DEVS methodology should select or implement the appropriate execution environment depending on the domain that you want to apply.

For example, if you perform a real-time-linked DEVS simulation or co-operation with a continuous time simulation, which is a major recent DEVS methodology application, the engine improvement may not be an attractive option because there is only a performance improvement for simulation within DEVS space.

Through the continuation of the study, we want to abstract the relationship between RTL-DEVS atomic models, which were constrained by gate-level modeling, into coupled models to create an extended RTL-DEVS model at the module level. Subsequent work aims to insert an intermediate compiler logic that can serialize and deserialize Verilog languages, thereby transitioning these relationships between the DEVS model we designed and the Verilog code. Phrases generated using a Verilog parser must be restored to objects sequentially, starting from the lower levels. Therefore, we need to focus more on the EOC, IOC, IC, and hierarchical structure communication of each coupled model to counteract the block in the Verilog code syntax.

The long-term goal is to fully simulate the Verilog code in our DEVS based simulator.

Author Contributions: Conceptualization, Y.S.H.; methodology, J.S.L.; software, B.S.K.; validation, B.S.K. and J.S.L.; writing—original draft preparation, Y.S.H.; writing—review and editing, B.S.K.; supervision, J.S.L.; funding acquisition, Y.S.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Research Foundation of Korea (NRF), grant funded by the Ministry of Science, ICT and Future Planning of Korea Government (NRF-2021R1A2C1003122).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DEVS	Discrete Event System Specification
HDL	Hardware Description Language
RTL	Resister Transfer Level
DHMIF	DEVS-Based Hardware Model Interchange Format
FPGA	Field-Programmable Gate Array
XML	eXtensible Markup Language
RHS	Right-Hand Side
LHS	Left-Hand Side
EIC	External Input Coupling
EOC	External Output Coupling
IC	Internal Coupling
FSM	Finite State Machine
TA	Time Advance

References

1. Kim, S.T.; Shin, M.S.; Ryu, K.Y.; Cho, Y.J. A Case Study on Productivity Improvement by a Discrete Event-Driven Simulation System. *J. Soc. Korea Ind. Syst. Eng.* **2015**, *38*, 149–158. [CrossRef]
2. Bae, J.W.; Lee, G.; Na, H.; Moon, I.-C. Modeling and Simulation for Effectiveness Analysis of Anti-Ballistic Warfare in Naval Vessels. *J. Korea Soc. Simul.* **2023**, *32*, 55–66. [CrossRef]
3. Nam, S.M.; Kim, H.J. WSN-SES/MB: System Entity Structure and Model Base Framework for Large-Scale Wireless Sensor Networks. *Sensors* **2021**, *21*, 430. [CrossRef]
4. Sarjoughian, H.; Chao, Z.; Scherer, G. DEVS-Suite Simulator Guide: TestFrame and Database. [CrossRef]
5. Capocchi, L.; Santucci, J.F.; Poggi, B.; Nicolai, C. DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems. In Proceedings of the 2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Paris, France, 27–29 June 2011; pp. 170–175. [CrossRef]
6. Kim, S.; Cho, J.; Park, D. Accelerated DEVS Simulation Using Collaborative Computation on Multi-Cores and GPUs for Fire-Spreading IoT Sensing Applications. *Appl. Sci.* **2018**, *8*, 1466. [CrossRef]
7. Foguelman, D.; Henning, P.; Uhrmacher, A.; Castro, R. EB-DEVS: A formal framework for modeling and simulation of emergent behavior in dynamic complex systems. *J. Comput. Sci.* **2021**, *53*, 101387. [CrossRef]
8. Kim, K.H.; Kang, W.S.; Sagong, B.; Seo, H.G. Efficient distributed simulation of hierarchical DEVS models: Transforming model structure into a non-hierarchical one. In Proceedings of the 33rd Annual Simulation Symposium (SS 2000), Washington, DC, USA, 16–20 April 2000; pp. 227–233. [CrossRef]
9. Glinsky, E.; Wainer, G. DEVStone: A Benchmarking Technique for Studying Performance of DEVS Modeling and Simulation Environments. In Proceedings of the Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications, Montreal, QC, Canada, 10–12 October 2005; pp. 265–272. [CrossRef]
10. Kim, J.-K.; Lee, E.-S.; Choi, J.-S.; Lee, J.-S. DEVS-based hybrid modeling and simulation framework in distributed environment. In Proceedings of the Korea Information Processing Society Conference, Seoul, Republic of Korea, 20–23 October 2015; pp. 1065–1067. [CrossRef]
11. Wainer, G.A. An Introduction to Cellular Automata Models with Cell-DEVS. In Proceedings of the 2019 Winter Simulation Conference (WSC), National Harbor, MD, USA, 8–11 December 2019; pp. 1534–1548. [CrossRef]

12. Kwon, B.-S.; Jung, S.-W.; Noh, Y.-D.; Lee, J.-S.; Han, Y.-S. RTL-DEVS: HDL Design and Simulation Methodology for DEVS Formalism-Based Simulation Tool. *Telecom* **2023**, *4*, 15–30. [CrossRef]
13. Kim, T.G.; Kim, J.K.; Kim, Y.G. DHMIF: DEVS-based hardware model interchange format. In Proceedings of the European Simulation Symposium, Marseille, France, 18–20 October 2001.
14. Capocchi, L.; Bernardi, F.; Federici, D.; Bisgambiglia, P.A. A DEVS-Based Modeling Behavioral Fault Simulator for RT-Level Digital Circuits. In Proceedings of the SCS Summer Computer Simulation Conference (SCSC04), San Jose, CA, USA, 25–29 July 2004; pp. 481–486.
15. Plier, T.; Schwartz, D.; Lysecky, R.; Seo, C.; Zeigler, B.P. Discrete event system specification, synthesis, and optimization of low-power FPGA-based embedded systems. In Proceedings of the 2013 International Conference on Field-Programmable Technology (FPT), Kyoto, Japan, 9–11 December 2013; pp. 98–105. [CrossRef]
16. Cauteruccio, F.; Terracina, G. Extended High-Utility Pattern Mining: An Answer Set Programming-Based Framework and Applications. *Theory Pract. Log. Program.* **2023**, 1–31.
17. Jaic, K.; Smith, M.C. Enhancing Hardware Design Flows with MyHDL. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15), Monterey, CA, USA, 22–24 February 2015; Association for Computing Machinery: New York, NY, USA, 2015; pp. 28–31. [CrossRef]
18. Jiang, S.; Pan, P.; Ou, Y.; Batten, C. PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification. *IEEE Micro* **2020**, *40*, 58–66. [CrossRef]
19. Takamaeda-Yamazaki, S. Pyverilog: A Python-Based Hardware Design Processing Toolkit for Verilog HDL. In *Applied Reconfigurable Computing*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9040. [CrossRef]
20. Rose, J.; Luu, J.; Yu, C.W.; Densmore, O.; Goeders, J.; Somerville, A.; Kent, K.B.; Jamieson, P.; Anderson, J. The VTR project: Architecture and CAD for FPGAs from verilog to routing. In Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, New York, NY, USA, 22–24 February 2012; pp. 77–86. [CrossRef]
21. Molter, H.G.; Seffrin, A.; Huss, S.A. DEVS2VHDL: Automatic Transformation of XML specified DEVS Model of Computation into Synthesizable VHDL Code. In Proceedings of the 12th Forum on Specification and Design Languages (FDL 2009), Sophia Antipolis, France, 22–24 September 2009.
22. Kwon, S.J.; Kim, T.G. Design, Implementation and Performance Analysis of Event-oriented Execution Environment for DEVS. *J. Korea Soc. Simul.* **2011**, *20–21*, 87–96. [CrossRef]
23. Chen, B.; Vangheluwe, H. Symbolic flattening of DEVS models. In Proceedings of the 2010 Summer Computer Simulation Conference (SCSC '10), Ottawa, ON, Canada, 11–14 July 2010; Society for Computer Simulation International: San Diego, CA, USA, 2010; pp. 209–218.
24. Suresh, A.; Rohou, E.; Seznec, A. Compile-time function memoization. In Proceedings of the 26th International Conference on Compiler Construction (CC 2017), New York, NY, USA, 5–6 February 2017; Association for Computing Machinery: New York, NY, USA; pp. 45–54. [CrossRef]
25. Zeigler, B.P.; Praehofer, H.; Kim, T.G. *Theory of Modeling and Simulation*; Academic Press: London, UK, 2001. [CrossRef]