Simulation of RPDEVS Models of Logic Gates

Christian Fiedler[1], Franz J. Preyser[1*], Wolfgang Kastner [1]

[1]Institute of Computer Engineering, Automation Systems Group, TU Wien, Treitlstraße 1-3, 1040 Vienna, Austria
[*]*franz.preyser@tuwien.ac.at*

SNE 29(2), 2019, 85-91, DOI: 10.11128/sne.29.tn.10474
Received: April 10, 2019 (Selected ASIM GMMS/STS 2019 Conference Publication); Accepted; May 20, 2019
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. This paper addresses the simulation of fundamental logic gates (e.g. `AND`, `OR`, `NOT`) using the software *PowerRPDEVS* that is based on the *Revised Parallel Discrete Event System Specification* (RPDEVS) formalism. The formal differences of the models of a `NOR` gate in RPDEVS and PDEVS are analyzed. It is further shown, which possible pitfalls may occur when connecting these logic gates with feedbacks that cause algebraic loops and in which cases these algebraic loops are resolved by the RPDEVS simulation algorithm. For this purpose, a static RS flip-flop, a triggered D flip-flop and a shift register are modeled and simulated in *PowerRPDEVS*. The results are compared to previous research about the simulation of such logic circuits in *Simulink* and *Modelica*.

Introduction

In the theory of computation, the notion of Mealy and Moore automata exists which are forms of finite state automata [1]. The difference between these two types is how the output function is defined. The output function of the Moore type depends only on the internal state of the automaton, whereas for the Mealy type it also depends on the automaton's input. The formal definition of an automaton has a lot in common with the modeling formalism *Discrete Event System Specification* (DEVS) [2] and thus, also with its extension the *Parallel Discrete Event System Specification* (PDEVS), introduced by Chow and Zeigler [3]. However, since the output function in PDEVS depends only on the internal state, in principle, in PDEVS only Moore behavior is supported [4]. For Mealy behavior, a workaround including a *transitory state* (i.e. a state with zero life time) is necessary. This means that PDEVS models which immediately react to an external event with an output first have to enter a transitory state before the output function can be used to set the output.

In *Revised Parallel Discrete Event System Specification* (RPDEVS), introduced by Preyser et al. [5], the formalism was restructured to support Mealy behavior naturally. In RPDEVS immediate reactions to external events can be modeled directly with the output function, which removes the need for transitory states in this context.

As for PDEVS, an abstract simulator for RPDEVS was defined and published in the acompanying work [6]. An implementation is provided with the program *PowerRPDEVS* that also includes a graphical model editor.

With the goal to extend the PowerRPDEVS model library, we created a library with combinational logic and sequential logic elements. Combinational logic gates output the result of a boolean operator applied onto the input values. Thus, when signal delays are not taken into account, their models are all of type Mealy. Sequential logic components, in practice, are usually designed by coupling combinational logic elements with storage elements [7]. As Junglas' findings in [8] show, this can be a cumbersome business in simulation tools.

In this work, first it is analysed how the RPDEVS models of logic gates differ from the corresponding PDEVS models. Afterwards, it is investigated how the RPDEVS simulation algorithm performs when creating sequential logic components and the results are compared to Junglas' work.

1 The PDEVS and RPDEVS Formalisms

PDEVS and RPDEVS are hierarchical modelling formalisms where models are composed of *atomics* and *couplings*. An atomic can receive inputs from other atomics or couplings, it has an internal state and can produce outputs. Couplings can be composed of atomics and other couplings. The formal definition of a coupling is omitted here, it can be found in [3] and [6].

1.1 Atomic PDEVS

In Equation (1) the definition of a PDEVS atomic is given as tuple.

$$A := < X^b, Y^b, S, \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta > \quad (1)$$

X^b … set of possible input bags
Y^b … set of possible output bags
S … set of possible (internal) states of the atomic
$\delta_{ext} : Q \times X^b \to S$ … external state transition function
where $Q = \{(s,e) | s \in S, e \in [0, ta(s)]\}$
$\delta_{int} : S \to S$ … internal state transition function
$\delta_{conf} : S \times X^b \to S$ … confluent state transition function
$\lambda : S \to Y^b$ … output function
$ta : S \to \mathbb{R}_0^+ \cup \{\infty\}$ … time advance function

The time advance function ta determines the time to live $ta(s) \in [0, \infty]$ for every internal state $s \in S$. Whenever this time expires, an internal event is triggered, which first causes the execution of the output function λ and then leads to a state transition conducted by δ_{int}. However, if at the same time an input event occurs, the state transition is performed by δ_{conf}. If the atomic is not *imminent* (i.e. it has no internal event) while an input event x^b is received, δ_{ext} is called and the atomic changes into a new state $s' = \delta_{ext}(s, e, x^b)$ without producing any output. The λ function that sets the output of the atomic is only evaluated right before an internal state transition and relies on the old state of the model. Thus, when an atomic has to respond to an input with a change in output, there has to be a state transition in δ_{ext} (or δ_{conf}) into a transitory state (i.e. a state s' with $ta(s') = 0$). This way, λ can set a new output at the same point in simulation time.

1.2 Atomic RPDEVS

$$A := < X^b, Y^b, S, \delta, \lambda, ta > \quad (2)$$

$\lambda : (Q \times X^b) \to Y^b$ … output function
$\delta : (Q \times X^b) \to S$ … external state transition function
where $Q = \{(s,e) | s \in S, e \in [0, ta(s)]\}$

In RPDEVS (see atomic definition in Equation (2)), the three state transition functions of PDEVS are merged to one single transition function δ which always is preceded by an evaluation of the output function λ. This evalution of λ though, happens iteratively. That is, λ is recalculated every time the input bag has changed due to a λ computation in an influencing component. As shown in [5], this λ *iteration* terminates as long as the model does not contain algebraic loops. Furthermore, it still may terminate if the algebraic loop can be resolved as we will see in Section 1.4. In contrast to PDEVS, λ also depends on the current input bag. Thus, Mealy behavior can be modelled without having to change the internal state. In fact, pure functional blocks can be modeled which do not need an internal state at all, e.g. a logic `NOT` gate just forwards input messages inverted to the output.

As already mentioned in the introduction, a PDEVS model can be compared to a Moore machine ($\lambda(s)$), wheras an RPDEVS can be compared to a Mealy machine because its output is a function of the input and the internal state ($\lambda(s, e, x^b)$).

1.3 `NOR` gate

A `NOR` gate with two inputs is constructed in PDEVS and RPDEVS to demonstrate the differences with an example in the context of logic gates.

$$NOR_{PDEVS} := < X, Y, S, \delta_{ext}, \delta_{int}, \delta_{conf}, \lambda, ta > \quad (3)$$

$$\begin{aligned}
X &= \{1,2\} \times \{0,1\} \\
Y &= \{0,1\} \\
S &= \{0,1\}^2 \times \{0,\infty\}, \quad s = (s_1, s_2, \sigma) \in S \\
\delta_{ext}(s,e,x) &= (a_1(s,x), a_2(s,x), 0) \\
\delta_{int}(s) &= (s_1, s_2, \infty) \\
\delta_{conf}(s,x) &= \delta_{ext}(s, ta(s), x) \\
\lambda(s) &= \neg(s_1 \vee s_2) \\
ta(s) &= \sigma \\
a_i(s, x^b) &= \begin{cases} x_i : & \text{if } (i, x_i) \in x^b \\ s_i : & \text{otherwise} \end{cases}
\end{aligned}$$

Equation (3) shows a PDEVS `NOR` gate. As an external event could update either both or only one input, the model has to keep the last seen values of its inputs in the internal state. A helper function a_i chooses the new input value from the input bag x^b if there is any, or otherwise the saved value from the internal state. The model also keeps the value σ for ta, which is set to 0 in the case of an external event. In this way, every external event is followed by a transitory state used to create an output event.

$$NOR_{RPDEVS} := < X, Y, S, \delta, \lambda, ta > \quad (4)$$

$$X = \{1,2\} \times \{0,1\}$$
$$Y = \{0,1\}$$
$$S = \{0,1\}^2 \ \ldots \ s = (s_1, s_2) \in S$$
$$ta(s) = \infty$$
$$\lambda(s,e,x^b) = \neg(a_1(s,x^b) \vee a_2(s,x^b))$$
$$\delta(s,e,x^b) = (a_1(s,x^b), a_2(s,x^b))$$

When looking at Equation (4) which shows the NOR gate in RPDEVS, one can see that, because λ can access the input bag directly, the transitory state is not needed and therefore, the state space is reduced.

1.4 Static RS Flip-Flop

Flip-flops are sequential logic elements. That means, their outputs not only depend on their current inputs but can also on historic input values [7]. This implies that they have an internal state to store the historic data. A static RS flip-flop is commonly built using two NOR gates connected with their outputs fed back to the input of the other (see Figure 1).

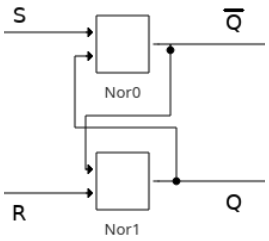

Figure 1: PowerRPDEVS model of the static RS flip-flop composed of two NOR gates.

Notably, the RS flip-flop is constructed from two *combinational* logic elements – the two NOR gates ideally have no state. Thus, deducing the mathematical model from the circuit of Figure 1 results in a system of two implicit equations:

$$Q = \neg(R \vee \overline{Q}) \quad (5)$$
$$\overline{Q} = \neg(S \vee Q) \quad (6)$$

Equations 5 and 6 can be solved as long as the inputs S and R are not both equal to 0 (see Table 1). The defined behavior for a RS flip-flop actually is to keep its previous output values in the case of $R = S = 0$. However, to know the previous value, the system needs to have a memory, i.e. an internal state. A real flip-flop is a continuous system and its signals are exposed to delays. These delays cause the system to still know its previous output, when the input changes.

S	R	Q	$\overline{Q}$
0	0	$\neg\overline{Q}$	$\neg Q$
0	1	0	1
1	0	1	0
1	1	0	0

Table 1: Solutions of Equations (5) and (6).

Due to the discrete event nature, our PDEVS and RPDEVS models have to store the last seen input values and, thus, also possess an internal state. When one of the inputs S or R of the RS flip-flop changes, the affected NOR gate still has stored the previous output of the other NOR gate. Consequently, during the simulation of the PDEVS and RPDEVS models, not Equations (5) and (6) are solved, but a recurrence relation. How this recurrence relation looks like depends on the number of inputs that change concurrently and on whether the simulation algorithm works in parallel or sequentially.

Single input change. We now consider the cases in which only one input changes its value.

If input S changes, the upper NOR gate first calculates its output using the new value of S and the stored value for Q. Then, due to the change in $\overline{Q}$, the lower NOR gate calculates its output, already using the new value for $\overline{Q}$. Thus, the recurrence relation has the form:

$$Q_n = \neg(R \vee \overline{Q}_n) \quad (7)$$
$$\overline{Q}_n = \neg(S \vee Q_{n-1}) \quad (8)$$

In Table 3 the evolution of the recurrence relation in Equations (7) and (8) is depicted for all possible initial states Q_{n-1} and input values S and R. It can be seen, that in all cases a fix point is reached after at least 2 iterations ($Q_{n+1} = Q_n$). Nevertheless, the simulation of this model in PDEVS leads to an infinite loop. When processing the external event due to the change in one of the two inputs, the affected gate schedules an internal event with $ta = 0$. Then λ sets the output and triggers the other gate for an external event. Finally, δ_{int} sets $ta = \infty$. The other gate is activated though, and will do exactly the same afterwards. This again reactivates the first gate and, thus, the simula-

Q_{n-1}	S	R	Q_n	$\overline{Q_n}$	Q_{n+1}	$\overline{Q_{n+1}}$
0	0	0	0	1	0	1
0	0	1	0	1	0	1
0	1	0	1	0	1	0
0	1	1	0	0	0	0
1	0	0	1	0	1	0
1	0	1	0	0	0	1
1	1	0	1	0	1	0
1	1	1	0	0	0	0

Table 2: Solutions of Equations (7) and (8).

tion gets stuck in a loop (the model is *illegitimate*). To mitigate this issue, the PDEVS model must be extended, such that δ_{ext} of the NOR gates only enters a transitory state when the input bit x_i is different from the already stored bit s_i. It should be noted here, that this is an example for how reusability of PDEVS models is impaired due to the need of transitory states for modeling Mealy behavior.

In RPDEVS, the simulation terminates. The change in the input S directly leads to an execution of λ of the upper NOR gate. The produced output respectively input for the lower gate then triggers λ of the lower NOR. The output produced thereby triggers a recalculation of λ at the upper gate. However, if the newly produced output of the upper gate does not differ from its previous one, the lower gate is not triggered again. Consequently, as long as the recurrence relation reaches a fix point in a finite number of steps, the RPDEVS simulation algorithm will find that fix point and will be able to continue simulation.

The case in which the input R changes and S does not change, is completely analog and, thus, is not described.

Concurrent input change. If both inputs S and R change concurrently, it depends on the simulation algorithm, how the recurrence relation that is solved during simulation looks like. In the case of a sequential RPDEVS simulation algorithm, like implemented in PowerRPDEVS [9], first λ of the first block is calculated. Then λ of the second block is calculated, already using the new output of the first block. Thus, the recurrence relation to solve again is the one of first order discussed in the previous paragraph. Consequently, PowerRPDEVS can handle any concurrent change of both inputs S and R without getting stuck in an endless loop.

However, if the RPDEVS simulation engine works in parallel, that is, it calculates λ of both gates concurrently, the recurrence relation to be solved would be of second order (see Equations (9) and (10)).

$$Q_n = \neg(R \vee \overline{Q}_{n-1}) \tag{9}$$

$$\overline{Q}_n = \neg(S \vee Q_{n-1}) \tag{10}$$

This recurrence relation can become unstable though. When both inputs are 1 and then concurrently change to 0, the outputs start to alternate between 0 and 1.

Q_{n-1}	$\overline{Q_{n-1}}$	S	R	Q_n	$\overline{Q_n}$	Q_{n+1}	$\overline{Q_{n+1}}$
X	X	1	1	0	0	0	0
0	0	0	0	1	1	0	0

Table 3: The recurrence relation in Equations (7) and (8) becomes unstable if S and R change concurrently from 1 to 0.

In PDEVS, a concurrent change of both inputs first leads to an execution of δ_{ext} at both gates, storing the new input in the internal state and setting $\sigma = 0$ to enter a transitory state. The transitory state leads immediately to internal events and thus, to a execution of λ in both gates. It does not matter whether λ of the gates is executed in parallel, or consecutively, because both use the old output of the other one that is stored in the component's state. Therefore, for the PDEVS simulation algorithm the concurrent change of both inputs S and R always leads to the solution of the second order recurrence relation in Equations (7) and (8) regardless whether execution is parallel or sequential.

2 Simulation

Simulation was done in *PowerRPDEVS* which is the proof-of-concept implementation of an RPDEVS modelling environment that includes the simulation engine and a graphical model editor. It is open source and available in [9].

2.1 Static RS Flip-Flop

The model of the RS flip-flop in Figure 1 was simulated with the initial values $Q = 0$ and $\overline{Q} = 1$. The input sequence and results are shown in Figure 2. Contrary to

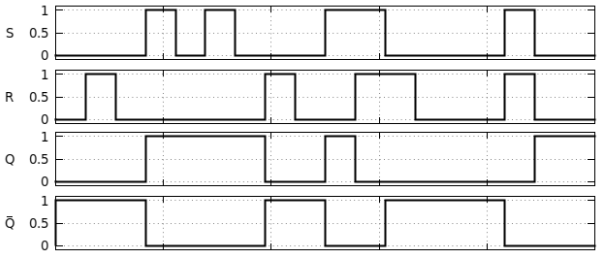

Figure 2: static RS flip-flop – Simulation results

simulation in Simulink [8], no work-arounds are needed and the outputs are not delayed.

A transition to the "forbidden" state $S = R = 1$ and back to $S = R = 0$ is included. As long as $S = R = 1$, the behaviour is actually well-defined as $Q = \overline{Q} = 0$. When S and R change to 0 simultaneously, the behaviour depends on the ordering of the λ function executions. As mentioned in Section 1.4, parallel execution of the `NOR` gates' λ functions would cause an infinite loop (oscillation) in the simulation, but it works in *PowerRPDEVS* because the execution is serialized.

2.2 D Flip-Flop

A (clock triggered) D flip-flop can be constructed from a triggered RS flip-flop and additional wiring at its inputs. The atomic `LogicTriggeredSampling` (`LTS`) was implemented for this example. It can detect edges on its second (lower) input, either triggering for rising edges, falling edges or both, and it either forwards the left ($y(t) = \lim_{\tau \nearrow t} x(\tau)$) or the right limit ($y(t) = \lim_{\tau \searrow t} x(\tau)$) of its first (upper) input when triggered by an edge. This block was placed before the inputs of a static RS flip-flop (see Figure 3). When the `LTS` blocks are set to take the right limit a change in the input that occurs at the same time as the clock edge is accepted by the flip-flop and it is not accepted otherwise.

It was first tried to use a different trigger detection mechanism: a *falling* block as in [8] in the Modelica model of the triggered RS flip-flop. This did not work out well in RPDEVS though, as the block has to send a `1` for an infinitesimal time frame and then switch back to `0` whenever it detects an edge. This means that because of the event-based nature of RPDEVS, the λ output at the time of the edge would be `1` and the block would need to schedule an internal event to set the output `0`. If ta is set to 0 for this purpose a transitory state would be introduced which we are trying to avoid. On the other hand, if it was set to $ta = x \in \mathbb{R}^+$ this would open a time frame during which the flip-flop would accept changes in its inputs, although the clock edge occured in the past. Thus, in the end the triggered RS flip-flop was modeled as depicted in Figure 3.

Figure 4 shows the D flip-flop consisting of the triggered RS flip-flop and a `NOT`. The results of the simulation of the D flip-flop are shown in Figure 5. $\overline{Q}$ is omitted as it always carries exactly the opposite logic level of Q. The triggering clock edge is set to be the falling edge.

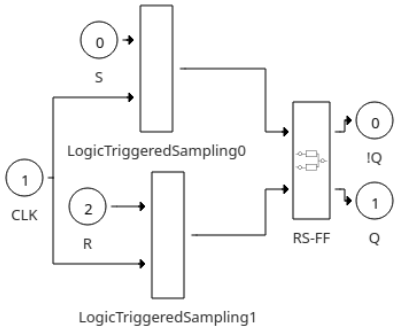

Figure 3: triggered RS flip-flop model

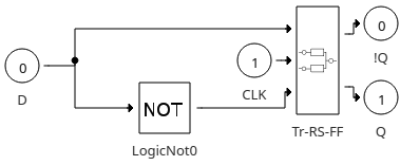

Figure 4: D flip-flop model

During the design of the `LTS` block we recognized that it is actually a D flip-flop in its own right. The block accepts its first input (corresponding to D) as its output only when there is an edge on its second input (corresponding to CLK) which is exactly the behaviour of a D flip-flop.

Replacing the D flip-flop with an `LTS` block yields exactly the same results as in Figure 5.

2.3 Shift register

A shift register is a series of D flip-flops where the input signal is shifted through one flip-flop at a time whenever the clock input triggers.

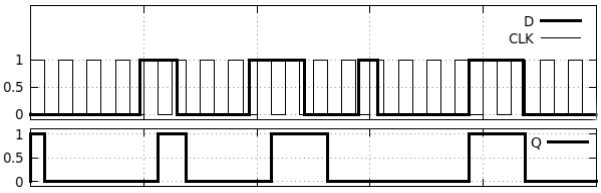

Figure 5: D flip-flop – Simulation results

The shift register in Figure 6 was modelled by using three of the D flip-flops designed above. Note that for the first flip-flop the `LTS` blocks (that are part of the triggered RS flip-flop, see Figure 3) is set to use the right limit and for the other flip-flops it is set to use the left limit of the input.

The reason is that the input signal would otherwise travel through all the flip-flops when the first clock edge arrives, because all D flip-flops are triggered by the same clock and none of them delays the signal.

The results of the simulation are shown in Figure 7. They show the individual D flip-flops' output and how the input signal (first a `1`, then a `0`) propagates through the shift register one stage per clock cycle.

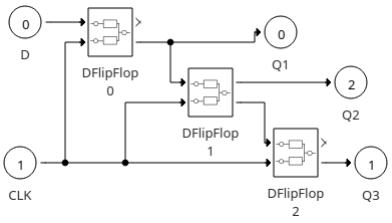

Figure 6: Shift register model

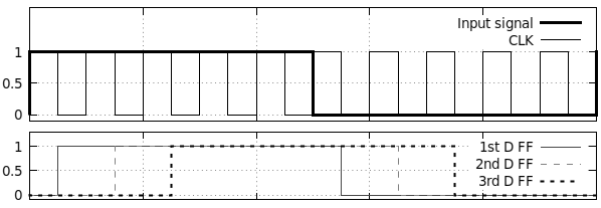

Figure 7: Shift register – Simulation results

In Junglas' tests [8], the Simulink model worked correctly without intervention, but the Modelica model seemed to show a peculiar issue that he mitigated by placing *Pre* blocks between the flip-flops which introduces an infinitesimal delay to break algebraic loops.

3 Conclusion

The *Revised Parallel DEVS* formalism offers new ways to deal with immediate outputs (Mealy behaviour of models) and algebraic loops. Specifically, we discussed a purely functional `NOR` gate in detail, showing that a model of it in RPDEVS can be realized with a smaller state space than in PDEVS, thus, reducing the complexity of the model. The static RS flip-flop was presented to show the behaviour of RPDEVS models with a feedback loop with no delay. The result was that a primitive coupling of `NOR` gates to form an RS latch would almost in any case lead to the expected behaviour of a physical `NOR` gate, but a transition to the "forbidden" state can lead to oscillation if the simulation engine utilizes parallelism.

The simulation of the RS flip-flop with PowerRPDEVS shows the behaviour that is expected from an RS flip-flop, without the need to introduce delay blocks or arrange it in a special way, which is necessary in other simulators.

The triggered D flip-flop model required the creation of the `LTS` block that was capable of forwarding an input event exactly when a clock edge occured which turned out to be a D flip-flop on its own. When they were put together to form a shift register, we needed to take into account the delays that actually make a shift register work and introduce them in our model as infinitesimal delays in the `LTS` block.

Acknowledgement

I want to thank my colleagues Franz Preyser and Wolfgang Kastner for the opportunity to publish and present this paper at the ASIM Workshop in Braunschweig this year. I am also very grateful for its acceptance in this journal propelled by Felix Breitenecker.

References

[1] Sakarovitch J, Thomas R. *Elements of Automata Theory*. 2011.

[2] Zeigler BP, Praehofer H, Kim TG. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press. 2000.

[3] Chow ACH, Zeigler BP. Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In: *Proc. of WSC'94*. 1994; pp. 716–722.

[4] Joslyn C. The process theoretical approach to qualitative DEVS. 1996; .

[5] Preyser FJ, Heinzl B, Kastner W. RPDEVS: Revising the Parallel Discrete Event System Specification. In: *Proc. of MATHMOD 2018*; pp. 269–274.

[6] Preyser FJ, Heinzl B, Kastner W, Breitenecker F. RPDEVS Abstract Simulator. In: *Proc. of ASIM-Workshop Simulation technischer Systeme/Grundlagen und Methoden in Modellbildung und Simulation*. 2019; p. 6.

[7] Cavanagh J. *Sequential Logic*. CRC Press. 2006.

[8] Junglas P. Pitfalls using discrete event blocks in Simulink and Modelica. In: *Tagungsband des Workshops der ASIM/GI-Fachgruppen STS und GMMS 2016*. 2016; pp. 90–97.

[9] PowerRPDEVS on SourceForge. `https://sourceforge.net/projects/powerrpdevs/`. [Online; accessed 27-Dec-2018].