Assignment 1

Submitted in partial fulfillment

of the requirements for the course:

SYSC 5104W - Methodologies of Discrete Event Modelling

to

Dr. Gabriel Wainer

by

Sasisekhar Mangalam Govind

101286270

Department of Systems and Computer Engineering

Carleton University, Ottawa

1125 Colonel By Dr

Ottawa, ON K1S 5B6, Canada

February 2024

Contents

List of Figures

Part 1

Conceptual Model of the DEVS-IAC Protocol

The Discrete Event System - Inter Atomic Communication (DEVS-IAC) protocol is a DEVS based communication protocol that facilitates communication between models running on distributed CADMIUM instances. DEVS-IAC was modelled with heavy inspiration from the Ethernet communication protocol. In the industry, real-time (RT) systems are gradually moving from protocols like Fieldbus or CAN towards Ethernet due to its low infrastructural cost and unanimity [2]. Hence, Ethernet was used as a foundation to create a real-time protocol that reliably trancieves data amongst DEVS models. DEVS-IAC, modelled using DEVS, allows a modeller to distribute their DEVS models while maintaining the formalism. The protocol can be generalised using the OSI communication model [1].

1.1 Coupled model

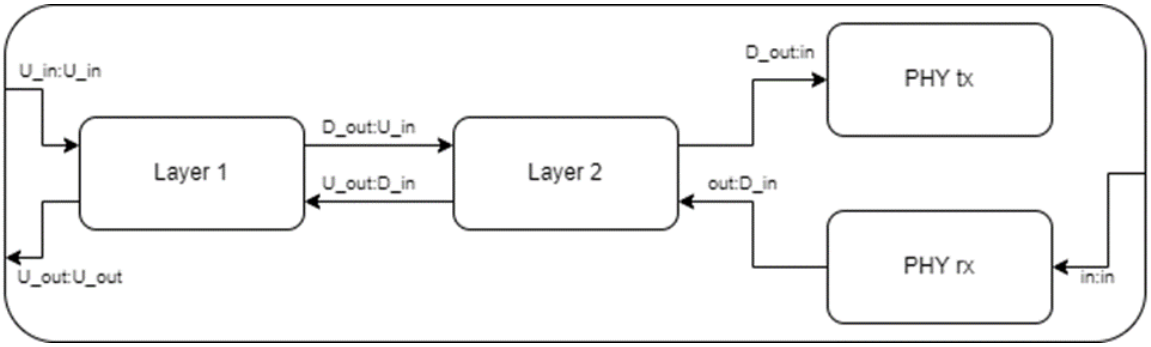

Figure 1.1: DEVS-IAC Coupled model

Figure 1.1 shows the structure of the DEVS-IAC coupled model. The model contains four atomic models:

1. **Layer 1**: Is responsible for converting incoming data into a byte stream or a bytestream into data expected by the upstream atomic model
2. **Layer 2**: Is responsible for converting the byte stream into a 64-bit wide *Frame* or a *Frame* into a byte stream
3. **PHY tx**: Responsible for transmitting input binary sequence as a series of pulses
4. **PHY rx**: Responsible for receiving voltage pulses and converting it into a bit stream

The Coupled model has two external inputs:

1. **in**: Receives Binary data from a different instance of the CADMIUM RT Engine
2. **Upstream_in (U_in)**: Receives model data from an atomic model

and one external output:

1. **Upstream_out (U_out)**: Produces the model data received from another instance of the CADMIUM RT Engine

Part 2

DEVS-IAC Protocol, Formalized

2.1 Formal Specification of the Coupled model

The formal specification of the DEVS model provided in Figure 1.1 is as follows:

$$C =< \{U_{in}, in\}, \{U_{out}\}, EIC, EOC, IC, SELECT > \tag{2.1}$$

Where,

- EIC = $\{(U_{in}, Layer_1.U_{in}), (in, PHY_{rx}.in)\}$
- EOC = $\{Layer_1.U_{out}, U_{out}\}$
- IC = $\{(Layer_1.D_{out}, Layer_2.U_{in}), (Layer_2.D_{out}, PHY_{tx}.in), (PHY_{rx}.out, Layer_2.D_{in}), (Layer_2.U_{out}, Layer_1.D_{in})\}$
- SELECT = $\{PHY_{rx} > PHY_{tx} > Layer_2 > Layer_1\}$

The sextuple 2.2.4 formally defines the Inputs, Outputs, Internal Connections, External Input Connection, External Output Connections and Tie breaking logic for a DEVS coupled model. Now, Further subsections will formally define the atomic models inside this coupled model.

2.2 Formal Specification of the Atomic models

2.2.1 PHY_{rx}

The formal specification of the PHY_{rx} model is given in two parts.

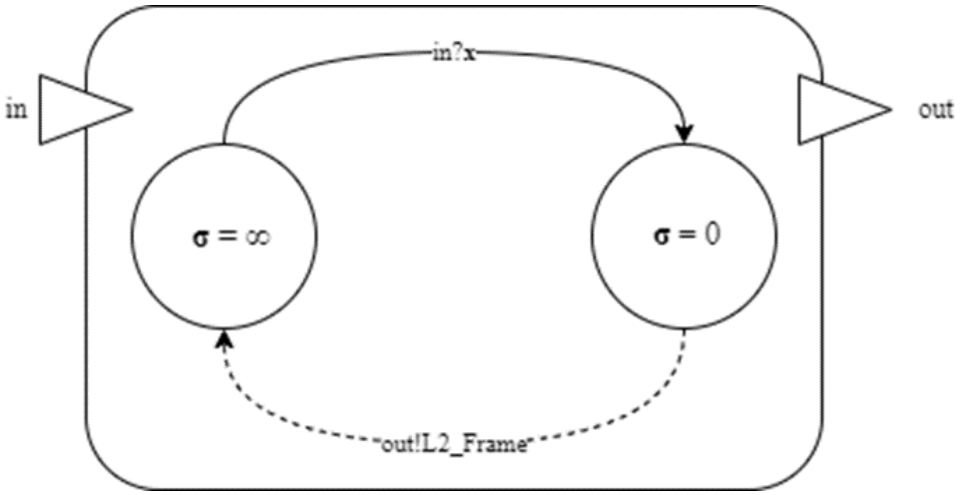

FIGURE 2.1: PHY_{rx} DEVS Graph

The DEVS graph given in 2.1 can be rewritten as the following:

$$PHY_{rx} =< X, Y, S, s_0, \delta_int, \delta_ext, \lambda >$$

Where,

- X = $\{in\ \epsilon\ \{\bar{x}\ \epsilon\ \{0,1\}^{64}\}\}$
- Y = $\{out\ \epsilon\ \{data,\ frame\_num,\ total\_frames,\ select,\ checksum\}\}$
- S = $\{\sigma,\ out\}$; $s_0\ =\ \{\sigma = \infty,\ out\ =\ \phi\}$
- $\delta_{int}(\sigma\ =\ 0)\ \{\sigma\ =\ \infty\}$
- $\delta_{ext}(\sigma\ =\ \infty,\ e,\ x)\ \{$

  $out\ =\ decode(\bar{x})$;

  $\sigma\ =\ 0$;

  $\}$
- $\lambda(\sigma\ =\ 0)\ \{send\ out\}$

2.2.2 PHY_{tx}

$$PHY_{tx} =< X, Y, S, s_0, \delta_int, \delta_ext, \lambda >$$

Where,

- X = $\{in\ \epsilon\ \{data,\ frame\_num,\ total\_frames,\ select,\ checksum\}\}$
- Y = $\{\phi\}$
- S = $\{\sigma,\ driver\}$; $s_0\ =\ \{\sigma = \infty,\ driver\ =\ \phi\}$
- $\delta_{int}(\sigma\ =\ 0)\ \{\sigma\ =\ \infty\}$
- $\delta_{ext}(\sigma\ =\ \infty,\ e,\ x)\ \{$

  $driver\ =\ serialize(in);$

  $\sigma\ =\ 0;$

  $\}$
- $\lambda(\sigma)\ \{\phi\}$

Here, the atomic model has a state variable *driver* that points to a transmit method in the hardware. As soon as this variable is set, the hardware takes this value and pushes it out using the suitable physical layer protocol (In this case, Manchester Encoding).

2.2.3 $Layer_2$

$$Layer_2 =< X, Y, S, s_0, \delta_int, \delta_ext, \lambda >$$

Where,

- X = $\{Upstream_{in},\ Downstream_{in}\}$
- Y = $\{Upstream_{out},\ Downstream_{out}\}$
- S = $\{\sigma,\ U_{out},\ D_{out}\}$
- $s_0\ =\ \{\sigma = \infty,\ U_{out}\ =\ \phi,\ D_{out}\ =\ \phi\}$
- $\delta_{int}(\sigma\ =\ 0)\ \{\sigma\ =\ \infty\}$

- $\delta_{ext}(\sigma\ =\ \infty,\ e,\ x)\ \{$

  $if(Upstream_{in})\{$
  $D_{out}\ =\ serialize(Upstream_{in});$
  $else\{$
  $U_{out}\ =\ de\_serialize(Downstream_{in});$
  $\}$
  $\sigma\ =\ 0;$

  $\}$

- $\lambda(\sigma)\ \{send\ (D_{out})?\ D_{out}\ :\ U_{out}\}$

2.2.4 $Layer_1$

$$Layer_1 =< X, Y, S, s_0, \delta_int, \delta_ext, \lambda >$$

Where,

- X = $\{Upstream_{in},\ Downstream_{in}\}$
- Y = $\{Upstream_{out},\ Downstream_{out}\}$
- S = $\{\sigma,\ U_{out},\ D_{out}\}$
- $s_0\ =\ \{\sigma = \infty,\ U_{out}\ =\ \phi,\ D_{out}\ =\ \phi\}$
- $\delta_{int}(\sigma\ =\ 0)\ \{\sigma\ =\ \infty\}$
- $\delta_{ext}(\sigma\ =\ \infty,\ e,\ x)\ \{$

  $if(Upstream_{in})\{$
  $D_{out}\ =\ serialize(Upstream_{in});$
  $else\{$
  $U_{out}\ =\ de\_serialize(Downstream_{in});$
  $\}$
  $\sigma\ =\ 0;$

  $\}$

- $\lambda(\sigma)\ \{send\ (D_{out})?\ D_{out}\ :\ U_{out}\}$

Part 3

DEVS-IAC Protocol, Tested

3.1 Model Under Test (Simulation)

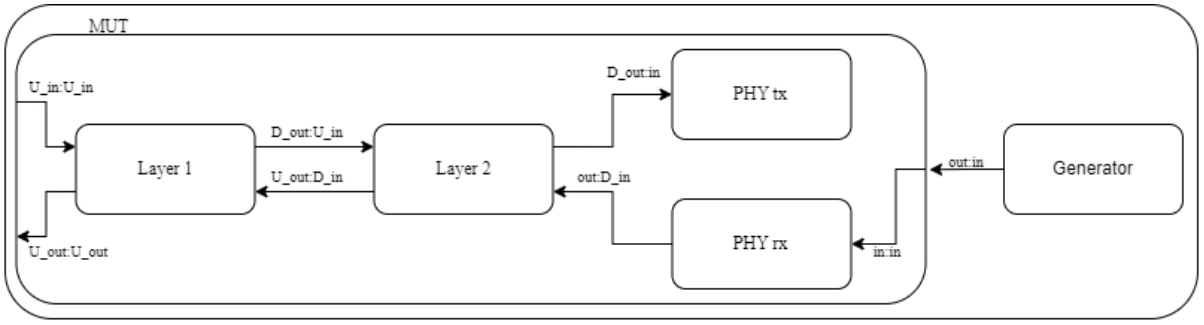

FIGURE 3.1: Simulation Environment of the DEVS-IAC model

In Figure 3.1 we can observe the simulation environment of the DEVS-IAC communication protocol. The Generator Atomic model is connected to the Model Under Test (MUT). The generator produces random values and injects it into the model to verify the functionality of the model.

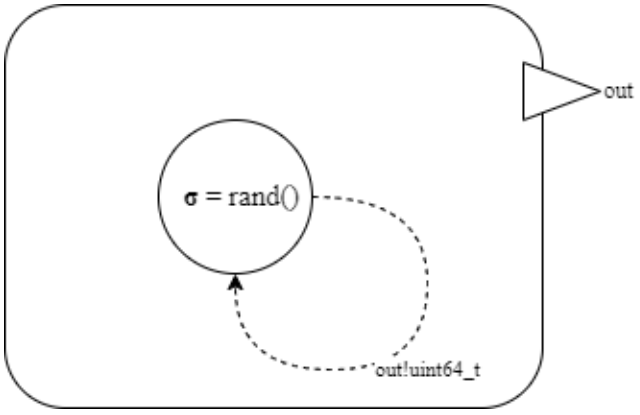

FIGURE 3.2: DEVS graph of the generator atomic model

Figure 3.2 shows the DEVS graph of the atomic model Generator. Every random time instance, the Generator injects a 64-bit unsigned integer into the MUT.

The Simulation output was as such:

TABLE 3.1: Simulation output of model given in 3.1

Time	Model ID	Model Name	Port Name	Data
...	...	...	...	...
1.4851	1	generator	out	57599806414848
1.4851	1	generator		57599806414848
1.4851	3	phy_rx		in_data: 0x346300002000
1.4851	3	phy_rx	out	$Layer_2$ *Frame*
1.4851	3	phy_rx		in_data: 0x346300002000
1.4851	4	Layer_2		$Layer_2$ *Frame*
1.4851	4	Layer_2	upstream_out	{0, 20, 0, }
1.4851	4	Layer_2		$Layer_2$ *Frame*
1.4851	5	Layer_1		downstream_in_data: { 0, 20, 0, }
1.4851	5	Layer_1	upstream_out	{ RGB: 0, 32, 0 }
1.4851	5	Layer_1		downstream_in_data: { 0, 20, 0, }
...	...	...	...	...

Upon observing the simulation output in Table 3.1, one can observe that each atomic model produces three log outputs. The first instance of each model shows the execution of δ_{ext}, then comes the execution of λ finally followed by the execution of δ_{int} We see that at a random time instance ($1.4851s$) the generator produces an output of a 64 bit integer. This integer propagates through the communication protocol to produce the output of {`0x00, 0x20, 0x00`} which represents the color Green.

3.2 Model Under Test (Execution)

Now that the model has been tested and verified, a simple case study is used to validate the functionality of the model. The model functionality is tested on a *Traffic Light model*.

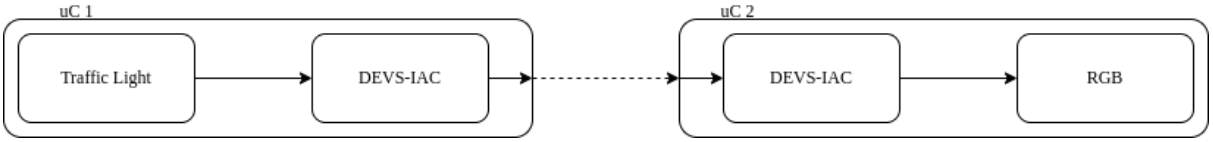

FIGURE 3.3: Execution Environment

From Figure 3.3 we can observe how the environment is set up. We see two microcontrollers, namely, $\mu C1$ and $\mu C2$. $\mu C1$ is the *Transmitter* that transmits the traffic light values as DEVS-IAC packets. $\mu C2$ is the *Receiver* that receives the DEVS-IAC messages and changes the colours of the RGB-LED (labelled as RGB).

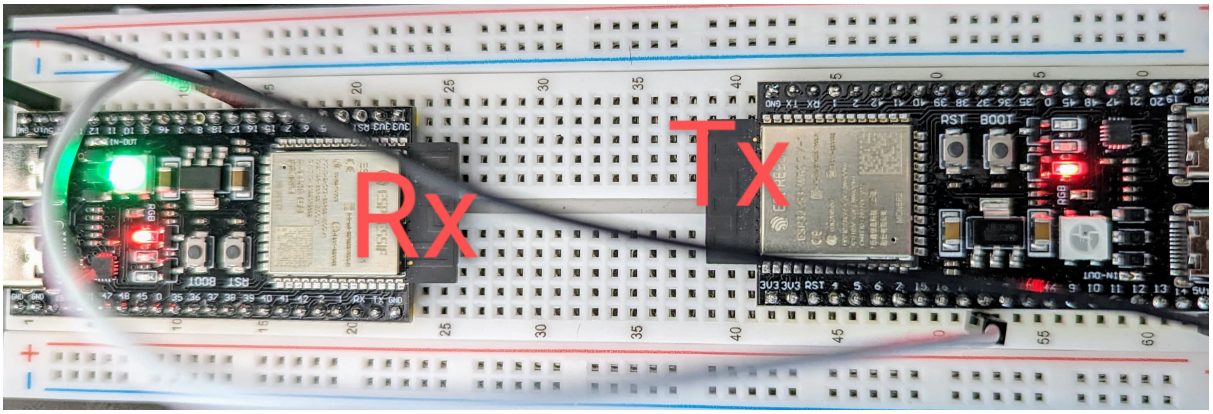

FIGURE 3.4: Physical Environment

The physical setup is shown in Figure 3.4. We can observe the two microcontrollers (ESP32s) with the left μC being the receiver and the μC on the right being the transmitter (labelled as Rx and Tx respectively).

TABLE 3.2: Execution logs of the Transmitter

Time	Model ID	Model Name	Port Name	Data
...	...	...	...	...
10	4	layer1		upstream_in_data: 0x002000
10	5	traffic_light	out	0x002000
10	5	traffic_light		Next State = 0x201000
10.1	3	layer2		upstream_in_data: {0, 20, 0, }
10.1	4	layer1	downstream_out	{0, 20, 0, }
10.1	4	layer1		upstream_in_data: 0x002000
10.2	2	phy_tx		HW_out: 0x346300002000
10.2	3	layer2	downstream_out	$Layer_2\ Frame$
10.2	3	layer2		upstream_in_data: {0, 20, 0, }
10.3	2	phy_tx		HW_out: 0x346300002000
...	...	...	...	...

TABLE 3.3: Execution logs of the Receiver

Time	Model ID	Model Name	Port Name	Data
...	...	...	...	...
10.3166	3	phy_rx		in_data: 0x346300002000
10.3166	3	phy_rx	out	$Layer_2\ Frame$
10.3166	3	phy_rx		in_data: 0x346300002000
10.3166	4	Layer_2		$Layer_2\ Frame$
10.3166	4	Layer_2	upstream_out	{0, 20, 0, }
10.3166	4	Layer_2		$Layer_2\ Frame$
10.3166	5	Layer_1		downstream_in_data: { 0, 20, 0, }
10.3166	5	Layer_1	upstream_out	{ RGB: 0, 32, 0 }
10.3166	5	Layer_1		downstream_in_data: { 0, 20, 0, }
10.3166	1	led_output		colour: { RGB: 0, 32, 0 }
...	...	...	...	...

Observing the execution logs of the Transmitter and Receiver in Table 3.2 and 3.3 respectively, we can observe the flow of information from one atomic model to the next. Observing Table 3.2 first, we see that the *traffic_light* atomic model generates a traffic light colour at the 10^{th} second of value `0x002000`. This value flows through the communication protocol and at the 10.3^{th} second, is transmitted as `0x346300002000`. Now, observing Table 3.3, we see that at the 10.3166^{th} second, the *phy_rx* model receives the value of `0x346300002000`. From this we see that the information is transmitted at the 10.3^{th} second and received at the 10.3166^{th} second. From this, we can infer that the Δt between transmission from $\mu C1$ to $\mu C2$ including the time it takes to encode and decode the data in the hardware level, is $16.6ms$. Further, we see the state of the RGB LED being set per the value of received from the transmitter.

References

[1] Mohammed M Alani. "Guide to OSI and TCP/IP models". In: (2014).

[2] Peter Neumann. "Communication in Industrial Automation-What is Going On?" In: *IFAC Proceedings Volumes* 37.4 (2004), pp. 63–71.