

System Modeling

Introduction

Rugby Meta-Model

Finite State Machines

Petri Nets

Untimed Model of Computation

Synchronous Model of Computation

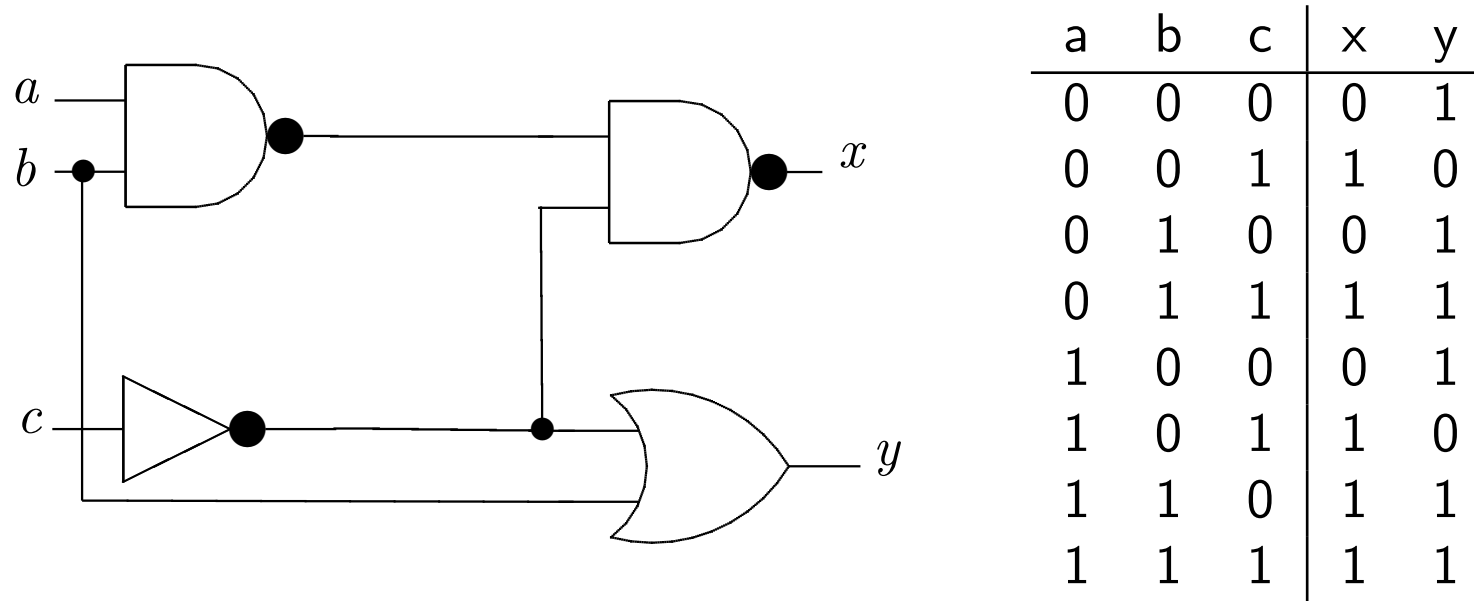
Timed Model of Computation

Integration of Computational Models

Tightly Coupled Process Networks

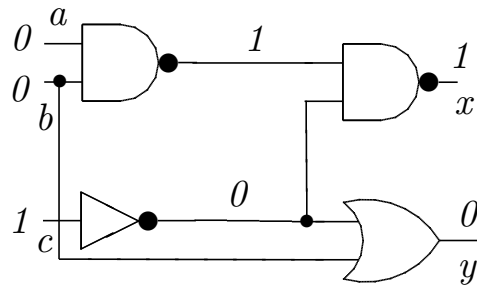


Timed Model of Computation

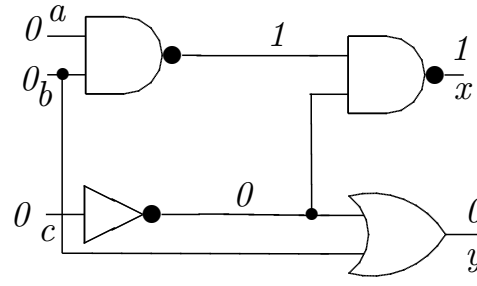


Gate	Delay
Inverter	1.5 ns
NAND gate	1.8 ns
OR gate	2.1 ns

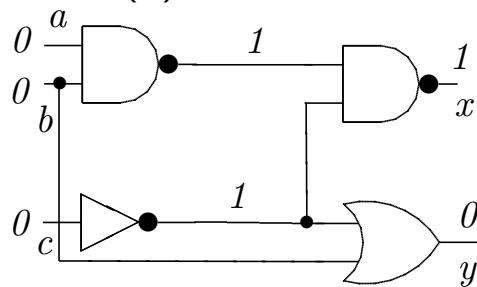
Propagation of Changes



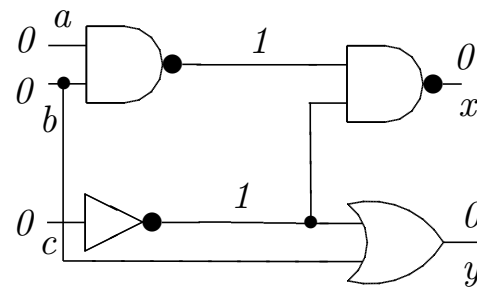
(a) at time t



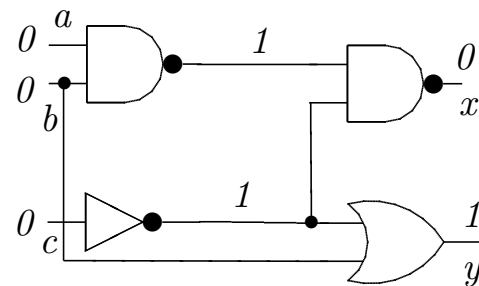
(b) at time t'



(c) at time $t' + 1.5\text{ns}$



(d) at time $t' + 3.3\text{ns}$



(e) at time $t' + 3.6\text{ns}$

Gate	Delay
Inverter	1.5 ns
NAND gate	1.8 ns
OR gate	2.1 ns

The Timed Model of Computation

- Uses also absent events to represent a global, synchronized time;
- The granularity of time is typically much smaller than for S-MoCs;
- Processes may consume and produce any number of events in each evaluation cycle;
- Absent events are inserted as needed to enforce causal behaviour;

A Timed MoC Process Constructor

$$\text{mealyT}(\gamma, g, f, w_0) = p$$

$$\text{where } p(\hat{s}) = \hat{s}'$$

$$\pi(\nu, \hat{s}) = \langle \hat{a}_i \rangle$$

$$\pi(\nu', \hat{s}') = \langle \hat{b}_i \rangle$$

$$\nu(i) = \gamma(w_i)$$

$$\nu'(i) = \gamma(w_i) + K_i$$

$$\hat{b}_i = \langle \sqcup \rangle^{K_i} \oplus \hat{c}_i$$

$$\hat{c}_i = f(w_{i-1}, \hat{a}_{i-1}) \quad \text{for } i > 0$$

$$w_i = g(w_{i-1}, \hat{a}_{i-1}) \quad \text{for } i > 0$$

$$\text{for } i \in \mathbb{N}_0, \hat{s}, \hat{s}', \hat{a}, \hat{b} \in \hat{S}, w_i \in V$$

The Delay to Enforce Causality

$$K_i = \begin{cases} \max(0, T_i(i) - T_o(i-1) - 1) & \text{if } i > 0 \\ \max(0, T_i(i) - 1) & \text{if } i = 0 \end{cases}$$

$$T_i(i) = \sum_{j=0}^i \text{length}(\hat{a}_j)$$

$$T_o(i) = \sum_{j=0}^i \text{length}(\hat{b}_j)$$

A Timed Process Example

$$p = \text{mealyT}(\gamma, g, f, 0)$$

$$\gamma(x) = 2$$

$$g(w, x) = 0$$

$$f(w, \langle x_1, x_2 \rangle) = \begin{cases} \langle x_2, x_1 \rangle & \text{if } x_2 > x_1 \\ \langle x_1 \rangle & \text{if } x_2 = x_1 \\ \langle x_2, x_1, x_2 - x_1 \rangle & \text{if } x_2 < x_1 \end{cases}$$

$$p(\langle 1, 2 \rangle) = \langle \sqcup, 2, 1 \rangle$$

A Timed Process Example - cont'd

$$p(\langle 1, 2, 3, 4, 5, 5, 5, 4, 3, 2, 1, 1, 1, 2, 2, 3, 3, 4, 4 \rangle)$$

$$= \langle \sqcup, 2, 1, 4, 3, 5, \sqcup, 5, \sqcup, 3, 4, -1, 1, 2, -1, 1, 2, 3, \sqcup, 4 \rangle$$

i	$T_i(i)$	$T_o(i)$	K_i	\hat{a}_i	\hat{b}_i	\hat{c}_i
0	2	3	1	$\langle 1, 2 \rangle$	$\langle \sqcup, 2, 1 \rangle$	$\langle 2, 1 \rangle$
1	4	5	0	$\langle 3, 4 \rangle$	$\langle 4, 3 \rangle$	$\langle 4, 3 \rangle$
2	6	6	0	$\langle 5, 5 \rangle$	$\langle 5 \rangle$	$\langle 5 \rangle$
3	8	8	1	$\langle 5, 5 \rangle$	$\langle \sqcup, 5 \rangle$	$\langle 5 \rangle$
4	10	12	1	$\langle 4, 3 \rangle$	$\langle \sqcup, 3, 4, -1 \rangle$	$\langle 3, 4, -1 \rangle$
5	12	15	0	$\langle 2, 1 \rangle$	$\langle 1, 2, -1 \rangle$	$\langle 1, 2, -1 \rangle$
6	14	16	0	$\langle 1, 1 \rangle$	$\langle 1 \rangle$	$\langle 1 \rangle$
7	16	17	0	$\langle 2, 2 \rangle$	$\langle 2 \rangle$	$\langle 2 \rangle$
8	18	18	0	$\langle 3, 3 \rangle$	$\langle 3 \rangle$	$\langle 3 \rangle$
9	20	19	1	$\langle 4, 4 \rangle$	$\langle \sqcup, 4 \rangle$	$\langle 4 \rangle$

Zip Constructor

$$\text{zipT}(\gamma) = p$$

$$\text{where } p(\hat{s}_a, \hat{s}_b, \hat{s}_c) = \hat{s}'$$

$$\dot{e}'_{i+1} = \langle \hat{a}_i, \hat{b}_i \rangle$$

$$\dot{e}'_0 = \sqcup$$

$$\pi(\nu_a, \hat{s}_a) = \langle \hat{a}_i \rangle, \nu_a(i) = \gamma(k_i)$$

$$\pi(\nu_b, \hat{s}_b) = \langle \hat{b}_i \rangle, \nu_b(i) = \gamma(k_i)$$

$$\pi(\nu_c, \hat{s}_c) = \langle \dot{c}_i \rangle, \nu_c(i) = \gamma(k_i)$$

$$\pi(\nu', \hat{s}') = \langle \sqcup \rangle^{\gamma(k_i)-1} \oplus \langle \dot{e}'_i \rangle, \nu'(i) = \gamma(k_i)$$

$$k_{i+1} = \hat{c}_i[1]$$

$$k_0 = 0$$

$$\text{for } \hat{s}_a, \hat{s}_b, \hat{s}_c, \hat{s}', \hat{a}_i, \hat{b}_i, \dot{c}_i \in \hat{S}, i \in \mathbb{N}_0, k_i \in \hat{E}$$

Unzip Constructor

$$\text{unzipT}() = p$$

$$\text{where } p(\hat{s}) = \langle \hat{s}', \hat{s}'' \rangle$$

$$\dot{e}_{i+1} = \langle \hat{a}'_i, \hat{a}''_i \rangle$$

$$\dot{e}_0 = \langle \sqcup, \sqcup \rangle$$

$$\pi(\nu, \hat{s}) = \langle \dot{e}_i \rangle, \nu(i) = 1$$

$$\pi(\nu', \hat{s}') = \langle \langle \hat{a}'_i \rangle \rangle, \nu'(i) = 1$$

$$\pi(\nu'', \hat{s}'') = \langle \langle \hat{a}''_i \rangle \rangle, \nu''(i) = 1$$

$$\text{for } \hat{s}, \hat{s}', \hat{s}'', \hat{a}'_i, \hat{a}''_i \in \hat{S}, \dot{e}_i \in \hat{E}, i \in \mathbb{N}_0$$

Source, Sink and Initialisation Constructors

$$\begin{aligned}
 \mathit{sourceT}(g, w_0) &= p \\
 \text{where} \quad p() &= \hat{s}' \\
 w_i &= \hat{e}'_i \\
 g(w_i) &= w_{i+1} \\
 \pi(\nu', \hat{s}') &= \langle \langle \hat{e}'_i \rangle \rangle, \nu'(i) == 1
 \end{aligned}$$

$$\begin{aligned}
 \mathit{sinkT}(g, w_0) &= p \\
 \text{where} \quad p(\hat{s}) &= \langle \rangle \\
 g(w_i) &= w_{i+1} \\
 \pi(\nu, \hat{s}) &= \langle \hat{a}_i \rangle, \nu(i) = 1
 \end{aligned}$$

$$\begin{aligned}
 \mathit{initT}(\hat{r}) &= p \\
 \text{where} \quad p(\hat{s}) &= \hat{r} \oplus \hat{s} \\
 \nu &= \nu' = 1 \\
 \hat{r}, \hat{s} &\in \hat{S}
 \end{aligned}$$

The Timed Model of Computation

Definition: The **Timed Model of Computation (Timed MoC)** is defined as $\text{Timed MoC} = (C, O)$, where

$$C = \{mealyT, zipT, unzipT, sourceT, sinkT, initT\}$$

$$O = \{\parallel, \circ, \mathbf{FB}_S\}$$

In other words, a process or a process network belongs to the **Timed MoC Domain** iff all its processes and process compositions are constructed either by one of the named process constructors or by one of the composition operators. We call such processes **T-MoC processes**.

Timed MoC Variants

Timer based process invocation (*mealyPT*)

- Functions f and g never see or return absent events;
- The process interfaces remove and insert absent events as needed;
- The function γ defines how many events are consumed. Essentially, it represents a timer.

Event count based process invocation (*mealyST*)

- Functions f and g never see or return absent events;
- The process interfaces remove and insert absent events as needed;
- The function γ defines how many **non-absent** events are consumed.
- f , g and γ have no influence on when they are invoked;

Event count with time-out (*mealyTT*)

- Functions f and g never see or return absent events;
- The process interfaces remove and insert absent events as needed;
- The function γ defines how many **non-absent** events are consumed and a maximum total number of input events.
- f , g and γ have little influence on when they are invoked, but can set a timer which, when expiring, triggers the process;

Handling of Global Time - Conflicting Requirements -

Independence property Processes obtain information only via their inputs;

Central maintenance Global time must be centrally maintained and distributed to all processes;

Handling of Global Time

- Different Approaches: **Local Timer** -

- Every process has local knowledge of the global time;
- Requires that the global time is efficiently communicated;
 - ★ Reasonable assumption in single processor systems;
 - ★ Questionable in distributed systems;
- Global state (time) can be used for synchronization and communication;
- Violates the independence property;

Handling of Global Time

- Different Approaches: **Time Tags** -

- All events are annotated with a global time stamp;
- Reconciles the independence property with central maintenance;
- Difficulties to properly relate events on different input signals of a process;
- Does not blend well with blocking read semantics;
- Generalizes naturally to continuous time;

Handling of Global Time

- Different Approaches: **Absent Events** -

- All signals are periodic containing “absent events”;
- Reconciles the independence property with central maintenance;
- Works well with blocking read semantics;
- Cannot handle well non-integral time structures;
- Does not easily generalize to continuous time;

Handling of Global Time - Implementation Issues -

- Global time can be realized as a shared timer resource; suitable for single processor implementations;
- Distributed systems require local clocks and synchronization protocols
- Different MoCs map differently well on different implementations;

For example:

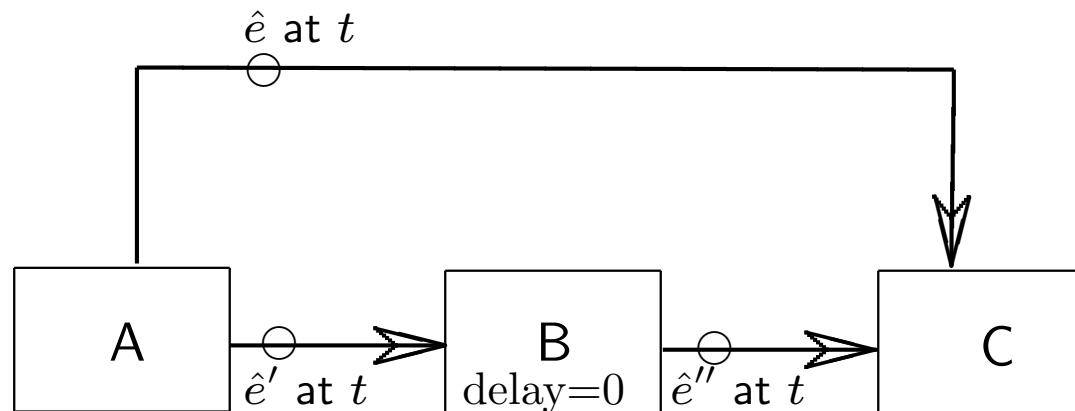
- ★ *mealyPT* (timer based process invocation) fits well to a shared timer resource;
- ★ *mealyST* (event count based process invocation) fits well to a distributed implementation;

δ -delay based Timed MoC

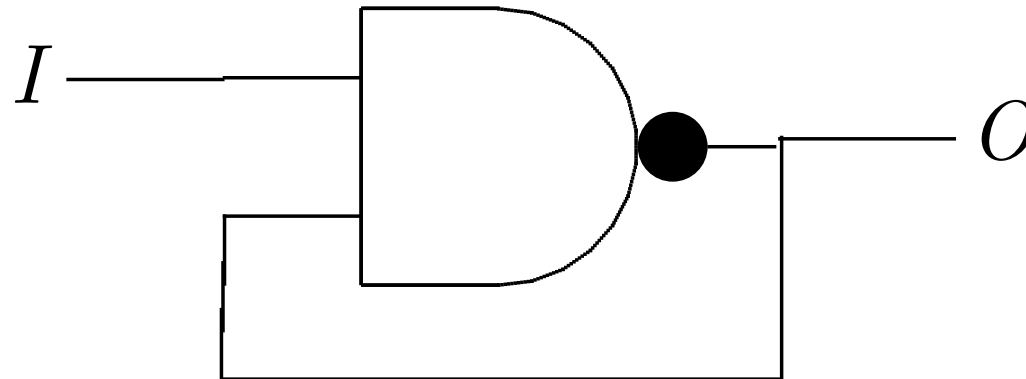
- Realizes global time
- Violates the independence property
- Prominent design languages are VHDL and Verilog

A Nondeterministic Discrete Event Model

A naïve discrete event model is nondeterministic due to simultaneous events. E.g.:

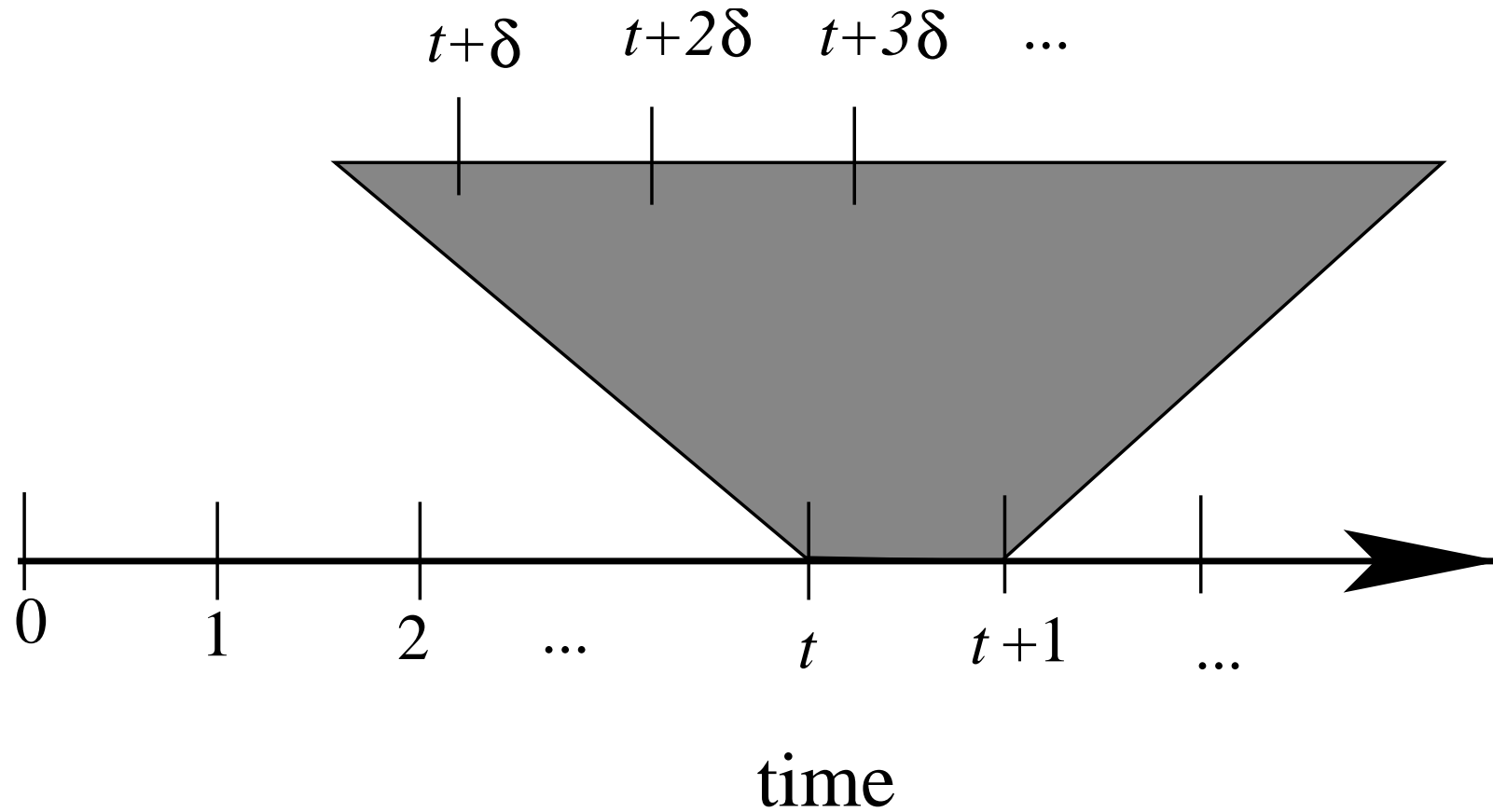


Feed-back loops in Níve Discrete Event Models

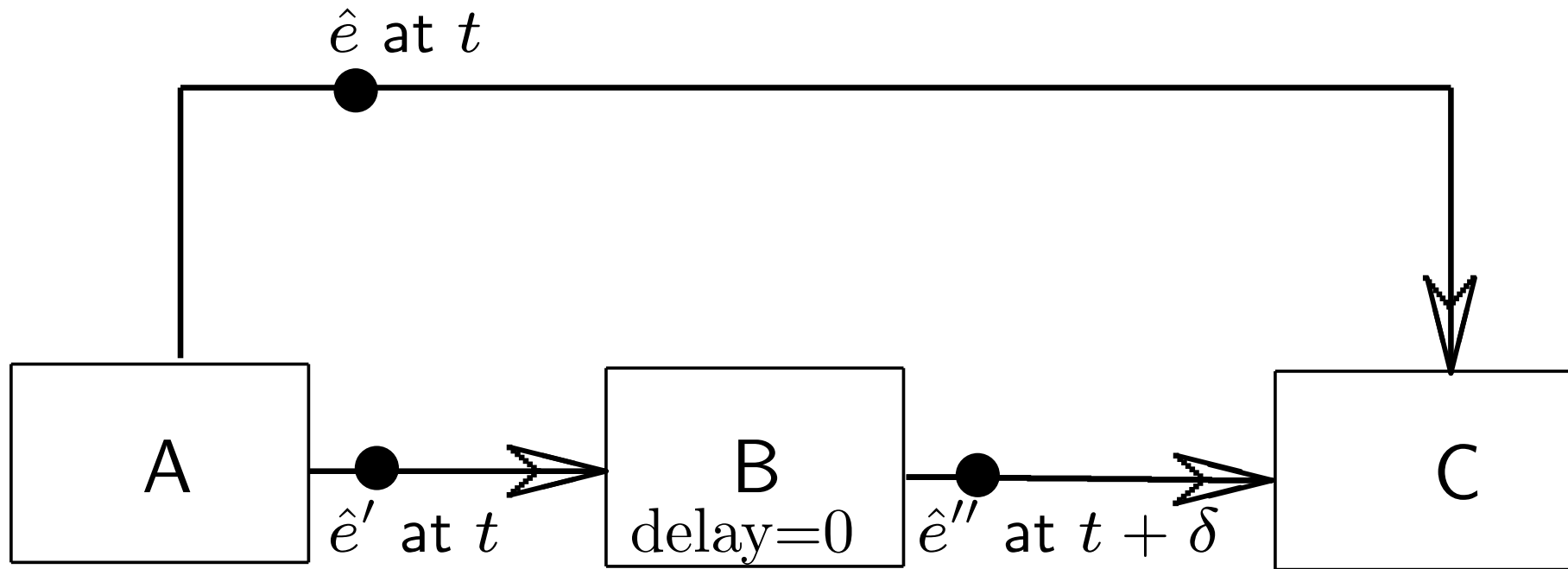


Inconsistency when $I = 1$.

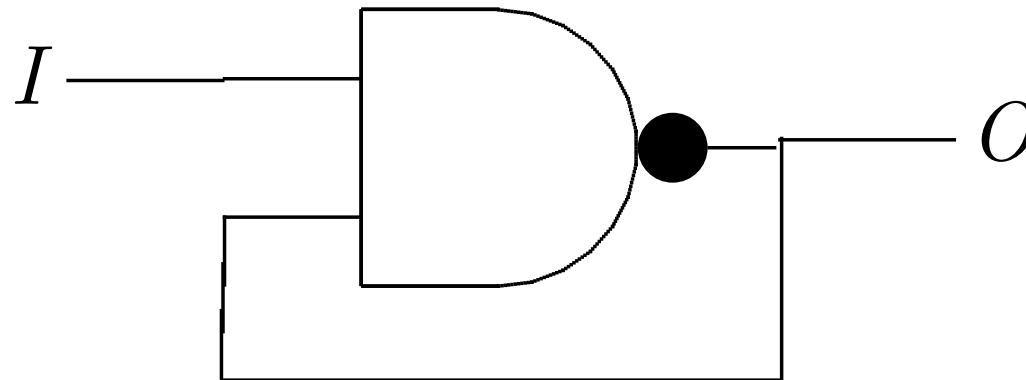
A Two Level Time Structure



A Deterministic δ -delay based Discrete Event Model

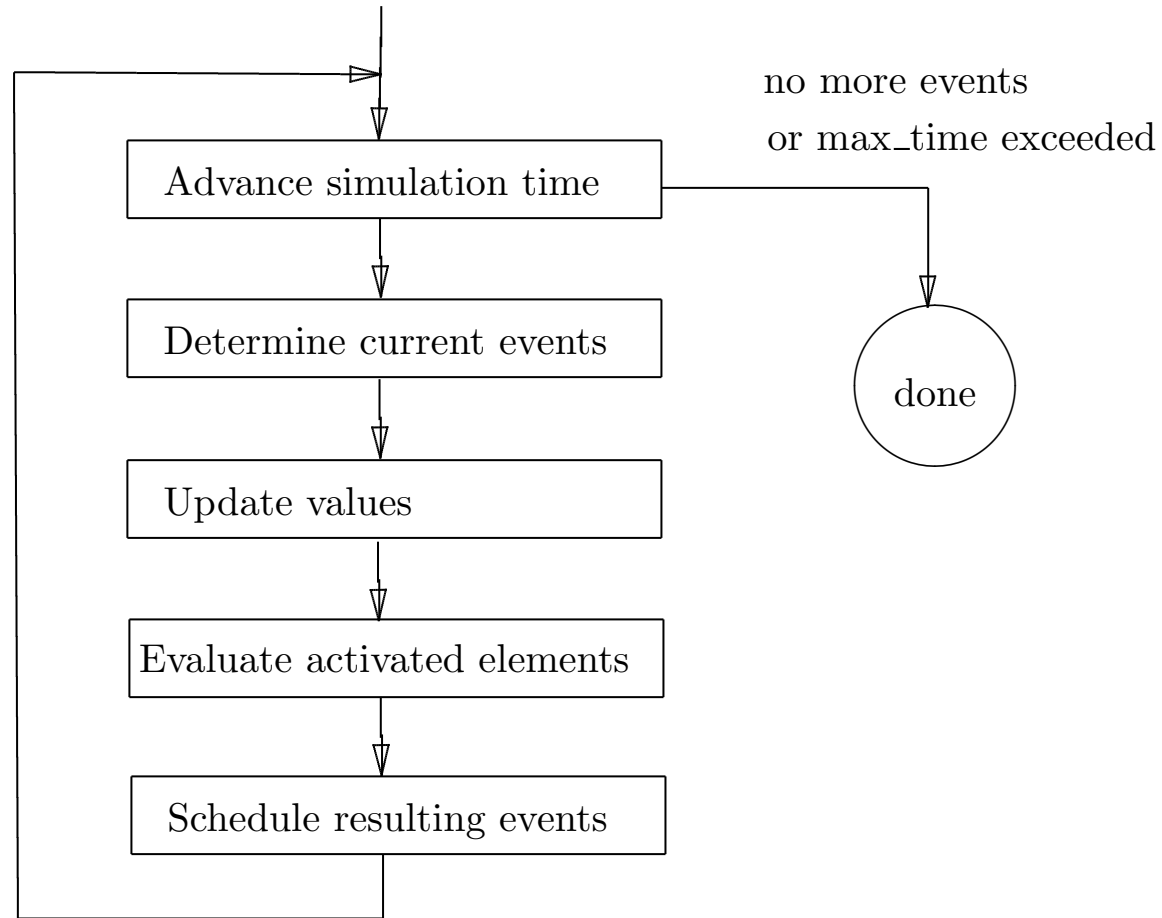


Feed-back Loops



No inconsistency because the output appears always 1δ after the corresponding inputs. But the circuit will oscillate indefinitely for $I = 1$ without ever advancing the real time.

Event Driven Simulation Cycle



Summary

- A timed model allows detailed timing analysis
- Maintenance of a global time is a challenge for distributed systems
- Different approaches to model global time:
 - ★ Absent events
 - ★ Time tag based
 - ★ Local timer which is synchronized with the global time
- Many design languages as timed models
- VHDL uses a local timer + time tag approach
- VHDL uses a δ delay model to avoid non-determinism and ambiguity
- Event queue becomes a bottleneck for VHDL simulations