VHDL
Basic Language Concepts: Simulation

EL 310
Erkay Savaş
Sabancı University

Motivation

- Construct VHDL models of digital systems for the purpose of simulating
- Quick start in building useful simulation models
- Core set of language constructs for describing attributes of digital systems
  - signals, events, propagation delays, concurrency, and waveforms

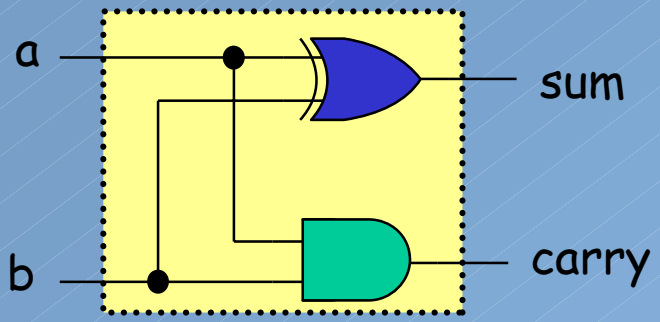
Signals 1

- Digital systems are about signals
- Signal is the basic part of any HDL
- The first VHDL object type: **signal**
- Signals may take values such as 1, 0, Z, X, L, H
- We can think of signal type as representative of the wires in a digital system in its simplest form
- Signals are different from variables
  - They have an associated time value.
  - A signal receives a value at a specific point in time.
  - It retains this value until it receives another value at a point in future.

Signals 2

- A sequence of values assigned to a signal over time is the waveform of the signal
- Signals may be declared to be of a specific type
  - Integer, real, or character
  - If this is the case, signal does not represent a single wire.
  - If we are simulating at a higher level of abstraction we wouldn't be concerned how many bit an integer signal should be assigned.

Entity-Architecture



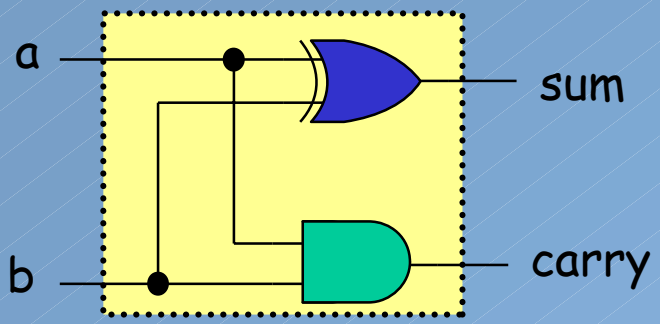
- Design entity: primary programming abstraction
  - Register, logic block, chip, or entire system
- What aspects of a digital system we want to describe?
  - Interface: how is it connected to other components? (inputs and outputs)
  - Function: what does it do for us? Internal behavior of the design (Boolean equations, truth tables, structural description).

Modeling a Digital System

- VHDL Language provides two distinct constructs to model these two aspects of a digital system:

1. Entity
2. Architecture

Entity Declaration



reserved key words

```
entity half_adder is
port(a, b: in bit;
     sum, carry: out bit);
end entity half_adder;
```

VHDL 1993

- Case insensitive
- Hyphen is not allowed in user supplied names first character must be a letter, last character cannot be underscore
- The interface is a collection of input and output ports.
- Ports are signals.
- They have types, e.g. **bit, integer, character, etc.**
- They have a mode of operation, e.g. in, out, inout (bi-directional)

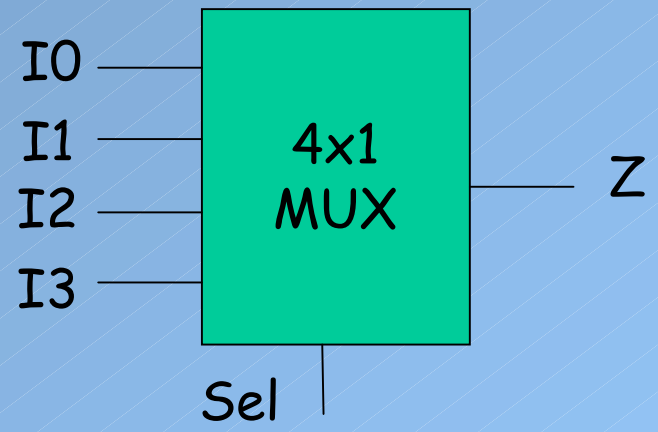
IEEE 1164 Standard Object Types

- Signals may be of different types
  - Vendors may define propriety types ➔ incompatibility problem.
  - SimVHDL defines a new type called `RealSignal` that can take on 12 values.
  - IEEE 1164 standard is developed to describe the logic systems more accurately.
  - The type **bit** is a part of VHDL.
  - But it is not sufficient to simulate the behavior of digital system since a wire can take values such as Z, X, L, H, etc.
  - IEEE 1164 Standard offers object type of `std_ulogic, std_ulogic_vector`

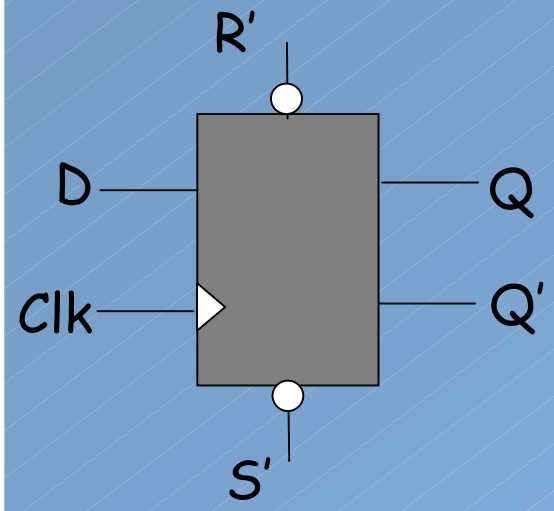
IEEE 1164

```
entity half_adder is

port(a, b: in std_ulogic;
     sum, carry: out std_ulogic);

end entity half_adder;
```

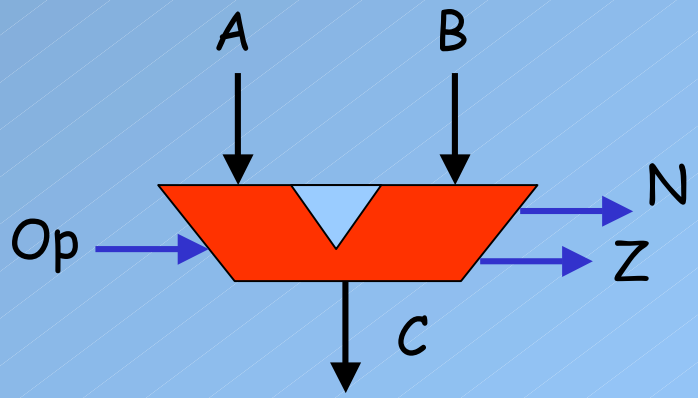
```
entity mux is

port(I0, I1: in std_ulogic_vector(7 downto 0);
     I2, I3: in std_ulogic_vector(7 downto 0);
     Sel   : in std_ulogic_vector(1 downto 0);
     Z     : out std_ulogic_vector(7 downto 0));
end entity mux;
```



Example Entity Descriptions

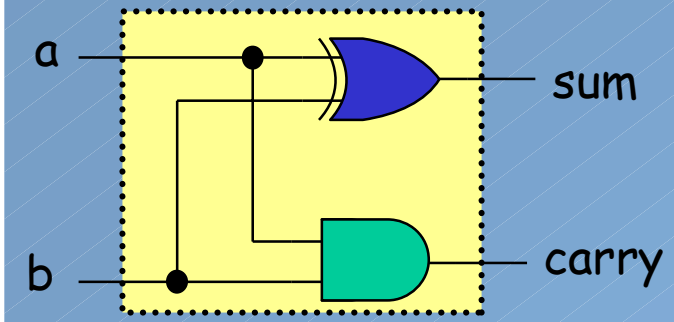



```
entity D_ff is
port(D, Clk, R, S: in std_ulogic;
     Q, Qbar     : out std_ulogic);
end entity D_ff;
```




```
entity ALU32 is
port(A, B: in  std_ulogic_vector(31 downto 0);
     C   : out std_ulogic_vector(31 downto 0);
     Op  : in  std_ulogic_vector(5 downto 0);
     N, Z: out std_ulogic);
end entity ALU32;
```

Describing Behavior: Architecture Construct




```
architecture behavioral of half_adder is
begin
sum <= (a xor b) after 5 ns;
carry <= (a and b) after 5 ns;
end architecture behavioral;
```

- The signal assignment statement: Description of events on the signals in left-hand-side (LHS) in terms of events on the signals in right-hand-side (RHS).
- Specification of propagation delays
- Recall: signal values are time-value pairs.

Architecture Construct

```
-- VHDL 1993
architecture behavioral of half_adder is
-- place declarations here
begin
-- place description of behavior here --
end architecture behavioral;
```

```
-- VHDL 1987
architecture behavioral of half_adder is
-- place declarations here
begin
-- place description of behavior here --
end behavioral;
```

Concurrent Assignment Statements

- Signal assignment operator: <=
- The operation of digital systems is inherently concurrent
  - Many components of a circuit can be simultaneously operating and concurrently driving distinct signals to new values.
  - Multiple signal assignment statements are executed concurrently in simulated time and referred to as concurrent signal assignment statements (CSA).

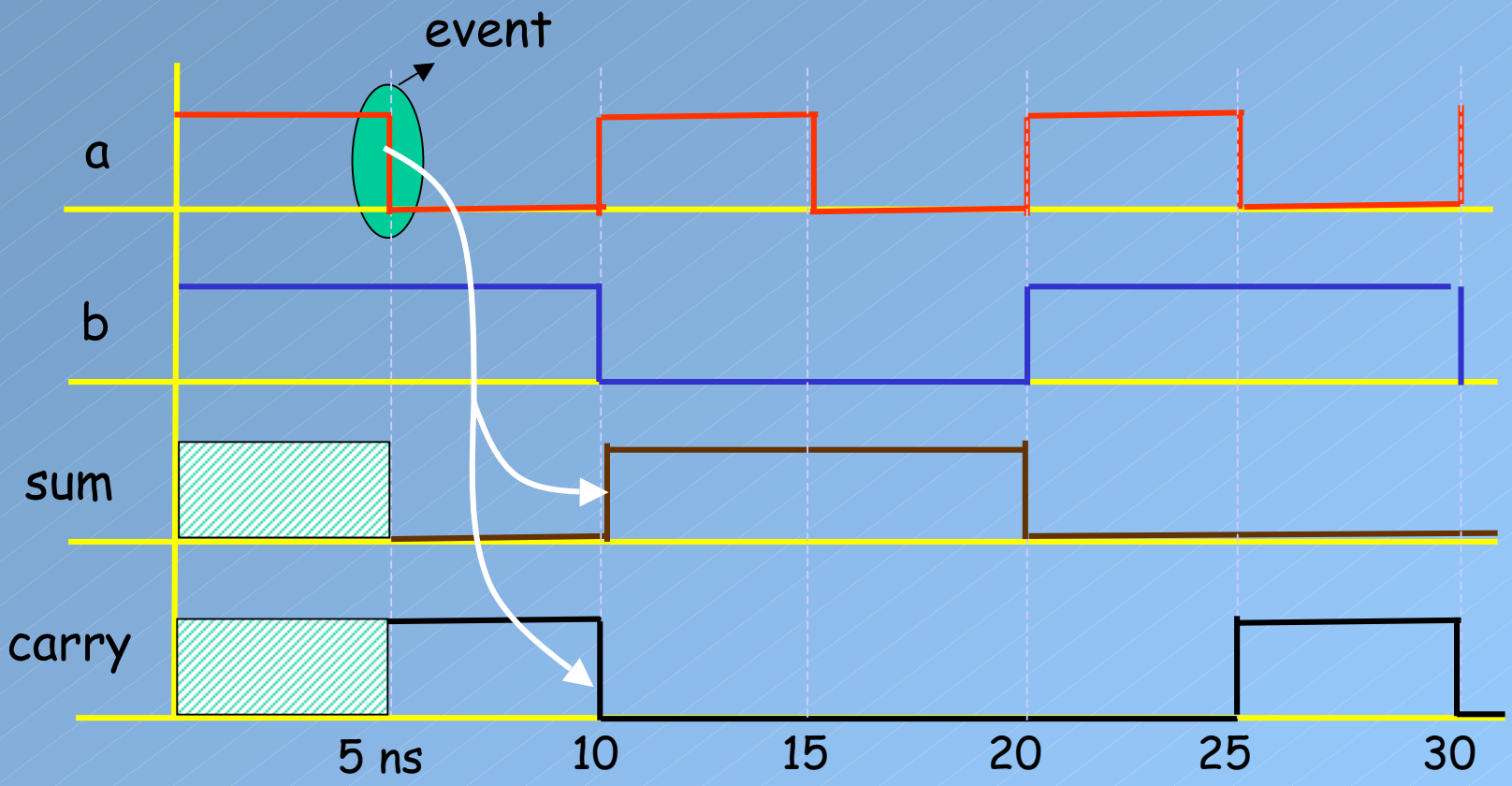
Simple CAS

```
library IEEE;
use IEEE.std_logic_1164.all;


entity half_adder is
port(x, y: in std_ulogic;
     sum, carry: out std_ulogic);
end entity half_adder;


architecture concurrent_behavioral of
half_adder is
begin
sum <= (a xor b) after 5 ns;
carry <= (a and b) after 5 ns;
end architecture concurrent_behavioral;
```

- Textual order is not important
- Flow of signals is important
- Simulation time does not proceed from one statement to the next

When a transition occurs at the right side of the assignment statement, the expression is evaluated and the assignment is scheduled to be performed at a future time determined by the time value after **after** keyword.

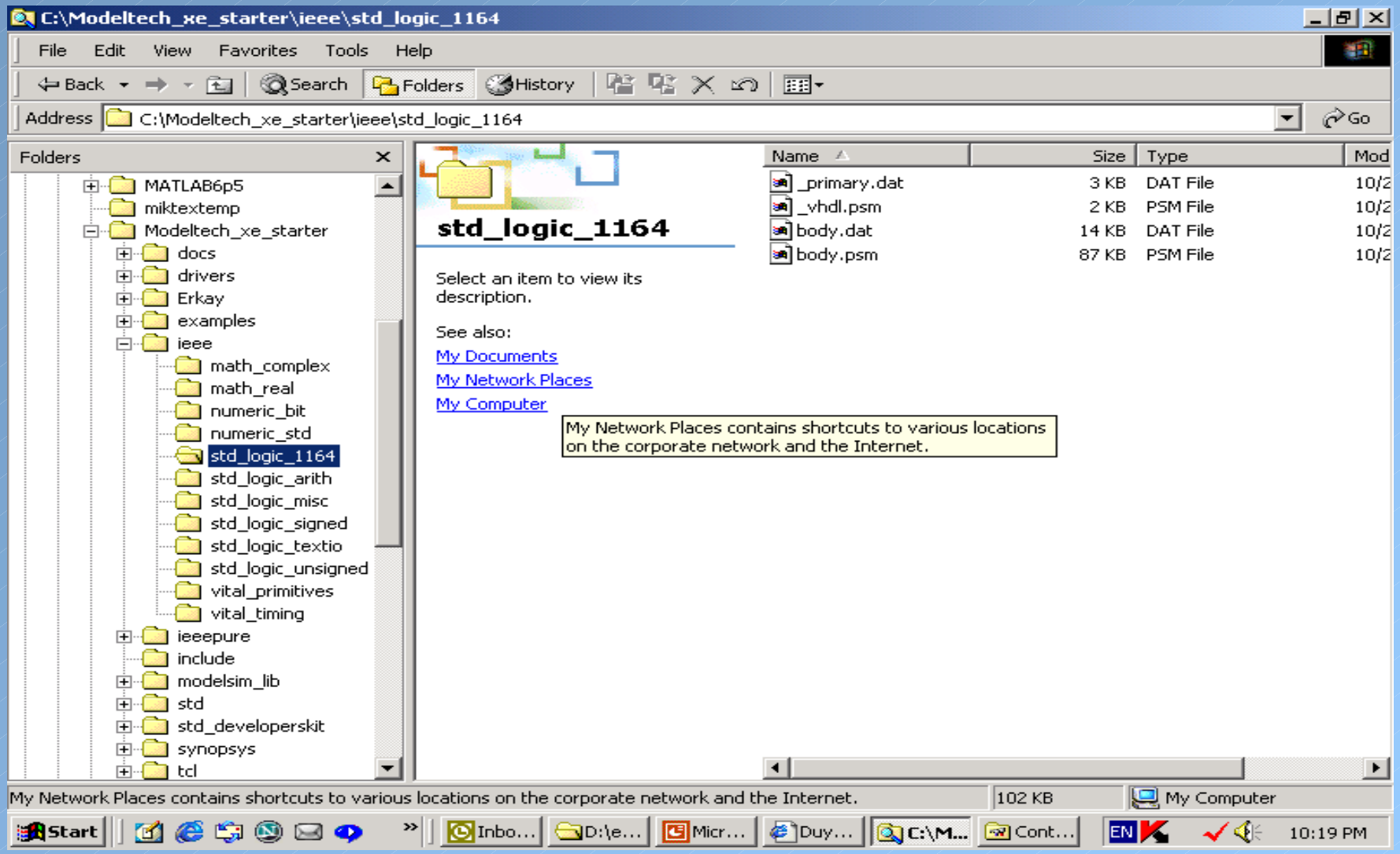
Simple CAS



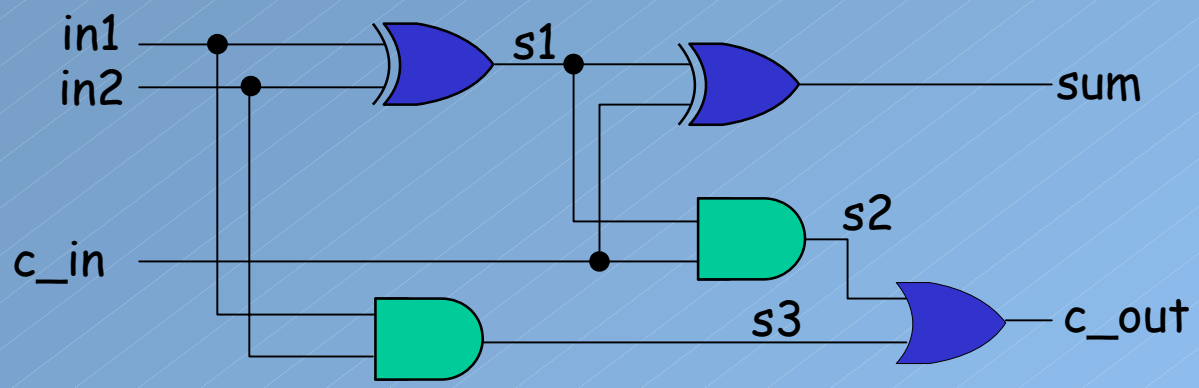
VHDL Libraries

- Libraries are repositories for frequently used design entities
  - The **library** clause identifies a library we want to access
  - The logical name (IEEE in the example) usually map to a directory in the system
  - The directory contains various design units that have been previously compiled.
  - A **package** is one such design unit
  - It may contain definition of types, functions, or procedures
  - The **use** clause determines which packages will be accessed in the library.

IEEE Library



VHDL Model of a Full Adder




```
-- VHDL 1993
architecture dataflow of full_adder is
-- place declarations here
begin
-- place description of behavior here --
end architecture dataflow;
```

VHDL Model of a Full Adder

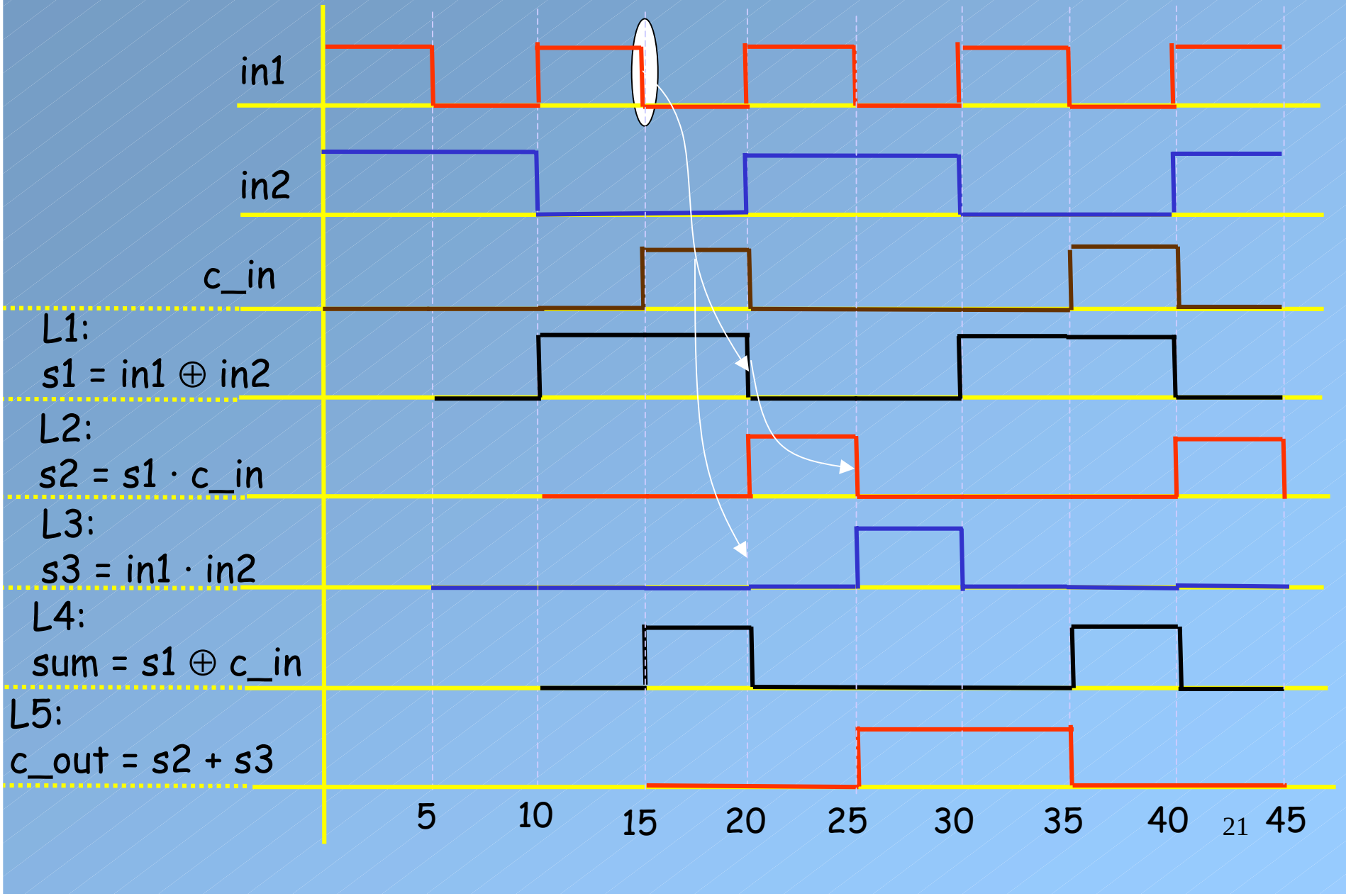
```
library IEEE;
use IEEE.std_logic_1164.all;

entity full_adder is
port(in1, in2, c_in: in std_ulogic;
     sum, c_out    : out std_ulogic);
end entity full_adder;

architecture dataflow of full_adder is
-- declarations
signal s1, s2, s3: std_ulogic;
constant gate_delay: Time:= 5 ns;

begin

L1: s1 <= (in1 xor in2) after gate_delay;
L2: s2 <= (c_in and s1) after gate_delay;
L3: s3 <= (in1 and in2) after gate_delay;
L4: sum <= (s1 xor c_in) after gate_delay;
L5: carry <= (s2 or s3) after gate_delay;

end architecture dataflow;
```

VHDL Model of a Full Adder

- Constant: can be declared to be of a specific type. Its value is set at the start of the simulation and cannot be changed during simulation
- Time is the only predefined physical type of language.

```
type TIME is range implementation_defined
   units
      fs;                    -- femtosecond
      ps  = 1000 fs;         -- picosecond
      ns  = 1000 ps;         -- nanosecond
      us  = 1000 ns;         -- microsecond
      ms  = 1000 fu;         -- millisecond
      sec = 1000 ms;         -- second
      min = 60 sec;          -- minutes
      hr  = 60 min;          -- hours
   end units;
subtype DELAY_LENGTH is TIME range 0 fs to TIME'HIGH;
```

Waveform of Full Adder



Activities at 15 ns

- Event:
  `in1` (1 → 0) [`in2` 0→ 0 (no event)]
- Expressions evaluated because of this event:

  `L1: s1 = in1 ⊕ in2` (1 →0) and
  `L3: s3 = in1 · in2` (0→0)
- `s1` and `s3` does not take the evaluated values immediately. Instead, they are scheduled to take the evaluated values at 20 ns.
- Therefore, expressions
  `L2: s2 = s1 · c_in` and `L4: sum = s1 ⊕ c_in`
  use the current value of `s1` which is 1.

- The scheduled event on `s1` is executed at 20 ns, which is itself an event and triggers other events.

Concurrent Assignment Rules

- model of simulation time:
  1. All statements with event occurring at the current time on signal in the RHS of the signal assignment are evaluated.
  2. All future events that are generated from the execution of these statements are then scheduled
  3. Simulation time is advanced to the time of next event.
  - Process repeats
  - User can specify events, delays, and concurrency
  - The order of execution of the statements depends on the flow of values.

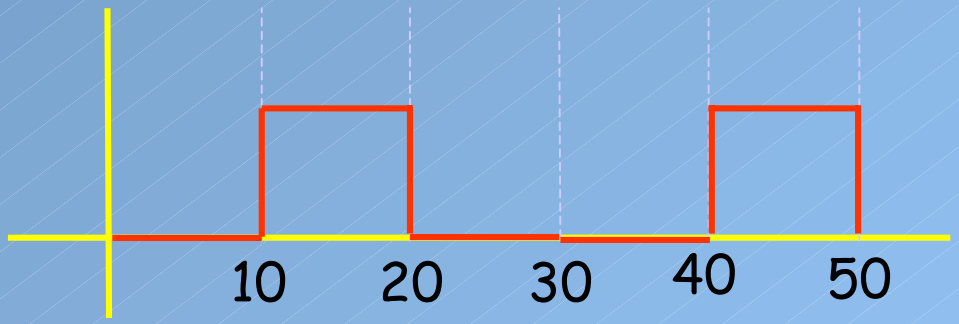
Implementation of Signals 1

- Initialized declaration:
  `signal s1 : std_ulogic:='0';`
  (initialization is not necessary)
- General form of concurrent signal assignment
  `signal <= value expression after time expression;`
- A signal has a history of values over time: waveform.
- RHS is referred to as waveform element.
- Time-value pair is referred to as a transaction.
- Can we specify multiple waveform elements?
  ```
  s1 <= (x xor y) after 5 ns, (x or y) after 10 ns,
  (not x) after 15 ns;
  ```
- Three transactions will be generated.
- The list of all the current transactions pending on a signal is called driver for the signal.
- Driver is essentially a waveform on the signal

Implementation of Signals 2

- The transactions that have not been occurred in simulation is called projected output waveform.
- Specifying waveforms:
- Example: a single waveform element

```
s1 <= '0', '1' after 10 ns, '0' after 20 ns, '1'
after 40 ns;
```



All waveforms must be ordered in increasing time.

```
s1 <= '0', '1' after 10 ns, '1' after 5 ns,
'0' after 20 ns, '1' after 40 ns
```
➔ Invalid

Resolved Signals 1

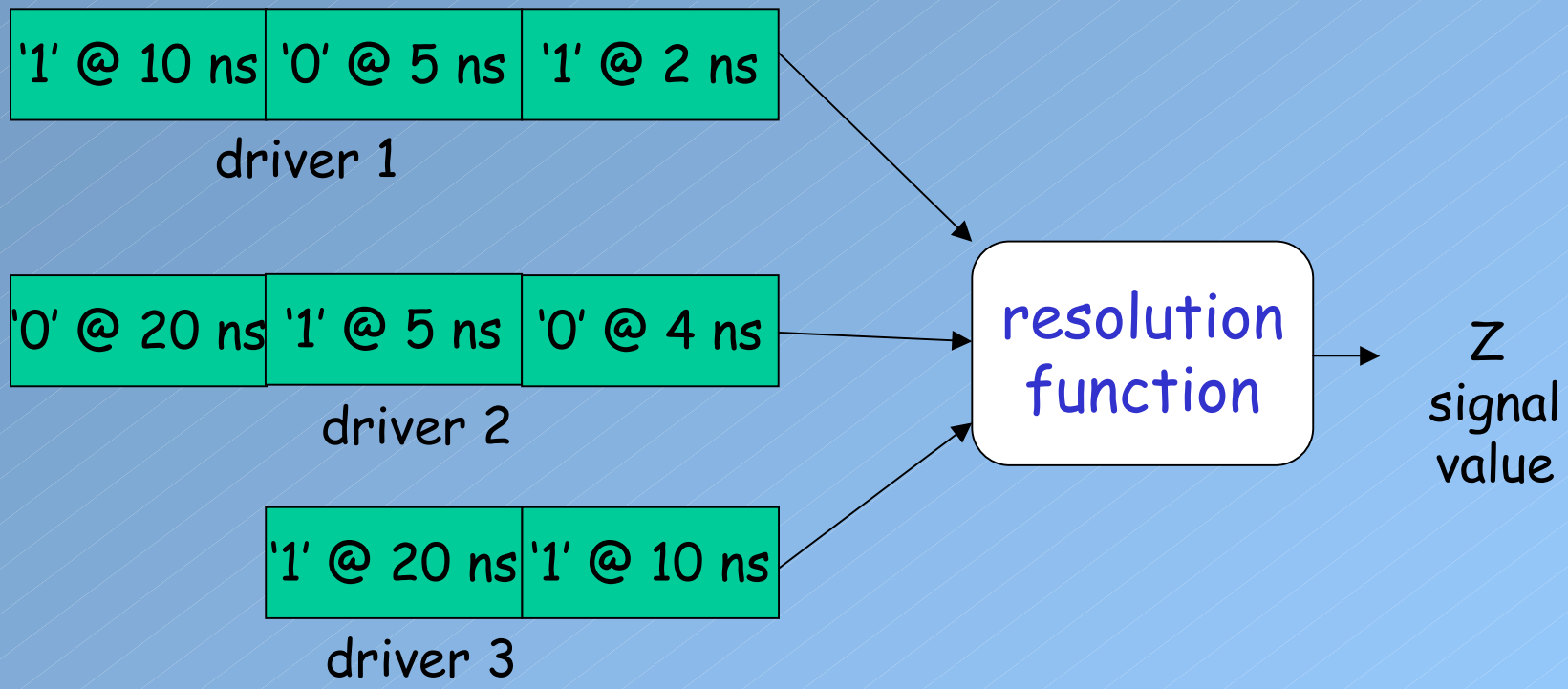
- One assumption
  - There is only one driver for a signal.
  - In real devices, there may be multiple drivers for a signal, e.g. busses, wired logic.
  - `s1 <= '0','1' after 10 ns,'0' after 20 ns,'1' after 40 ns;`
  - `s1 <= '1' after 10 ns,'1' after 20 ns,'0' after 40 ns;`
- Resolution function
  - A shared signal must be of special type: resolved
  - VHDL uses resolution function to resolve driver conflicts
  - `std_logic` and `std_logic_vector` are resolved versions of `std_ulogic` and `std_ulogic_vector`
  - User may define new resolved types, but he must also provide an implementation of resolution function.

STD_ULOGIC

```
type std_ulogic is(
      'U',  -- Uninitialized
      'X',  -- Forcing unknown
      '0',  -- Forcing 0
      '1',  -- Forcing 1
      'Z',  -- High impedance
      'W',  -- Weak unknown
      'L',  -- Weak 0
      'H',  -- Weak 1
      '-',  -- don't care
);

subtype std_logic is RESOLVED std_ulogic;
```

RESOLVED is the name of resolution function.

Resolved Signals 2



Resolved Signals 3

```
function WIRED_OR(inputs: bit_vector) return bit is
begin
    for j in inputs'range loop
        if(inputs(j)= '1' then
            return '1';
        end if;
    end loop;
    return '0';
end WIRED_OR;



subtype resolved_bit is WIRED_OR bit;
-- resolved_bit signal is associated with resolution
-- function WIRED_OR
```

Conditional Signal Assignment

- 4-to-1, 8-bit MUX

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux4 is
port(in0, in1, in2, in3: in std_logic_vector(7 downto 0);
     S0, S1: in std_logic;
     Z    : out std_logic_vector(7 downto 0));
end entity mux4;

architecture behavioral of mux4 is

begin

Z <= in0 after 5 ns when S0 = '0' and S1 = '0' else
     in1 after 5 ns when S0 = '0' and S1 = '1' else
     in2 after 5 ns when S0 = '1' and S1 = '0' else
     in3 after 5 ns when S0 = '1' and S1 = '1' else
     "00000000" after 5 ns;

end architecture behavioral;
```

- Question: What does the last statement cover?

4-to-1, 8-bit MUX

- Any event on input signals `in0-in3` or control signals `S0, S1` may cause a change in the value of the output signal
  - When this happens, all four conditions are checked and appropriate one is evaluated.
  - Expressions in the RHS are evaluated in the order they appear.
  - This is what is called priority order.
  - In the previous example, only one condition holds; thus the order does not matter.
  - In some models, priority is important

Priority Behavior

- 4-to-2 priority encoder

```
library IEEE;
use IEEE.std_logic_1164.all;

entity priority_encoder is
port(S0, S1, S2, S3: in std_logic;
     Z    : out std_logic_vector(1 downto 0));
end entity priority_encoder;

architecture behavioral of priority_encoder is

begin

Z <= "00" after 5 ns when S0 = '1' else
     "01" after 5 ns when S1 = '1' else
     "10" after 5 ns when S2 = '1' else
     "11" after 5 ns when S3 = '1' else
     "00" after 5 ns;

end architecture behavioral;
```

- What if S0 and S1 are '1' at the same time?
- This model is a combinational logic.

Unaffected Keyword

```
Z <= "00" after 5 ns when S0 = '1' else
     "01" after 5 ns when S1 = '1' else
     unaffected when S2 = '1' else
     "11" after 5 ns when S3 = '1' else
     "00" after 5 ns;
```

- Semantics:
  - When S2 = '1' and both S0 and S1 have the value '0', then the value of the output signal does not change.
  - This model represents a sequential circuit.
  - unaffected keyword is only supported in VHDL 1993.

Selected Signal Assignment 1

- Register file design
  - Read-only register file with two reading ports

```
library IEEE;
use IEEE.std_logic_1164.all;

entity register_file is
port(
  addr1, addr2: in std_logic_vector(2 downto 0);
  reg_out_1, reg_out_2 : out std_logic_vector(31 downto 0));
end entity register_file;

architecture behavioral of register_file is

signal reg0,reg2:
       std_logic_vector(31 downto 0):=x"12345678";
signal reg1,reg3:
       std_logic_vector(31 downto 0):=x"abcdef00";

begin

...

end architecture behavioral;
```

Selected Signal Assignment 2

```
...
begin
with addr1 select
reg_out_1 <= reg0 after 5 ns when “000”;
             reg1 after 5 ns when “001”;
             reg2 after 5 ns when “010”;
             reg3 after 5 ns when “011”;
             reg3 after 5 ns when others;
with addr2(1 downto 0) select
reg_out_2 <= reg0 after 5 ns when “00”;
             reg1 after 5 ns when “01”;
             reg2 after 5 ns when “10”;
             reg3 after 5 ns when “11”;
             reg3 after 5 ns when others;
end architecture behavioral;
```

- **select** is similar to case statement in C.
- All choices are evaluated; only one must be true.
- Specified choices must cover all the possibilities
- Question: Why do we need others in the last statement

A Note in Portability

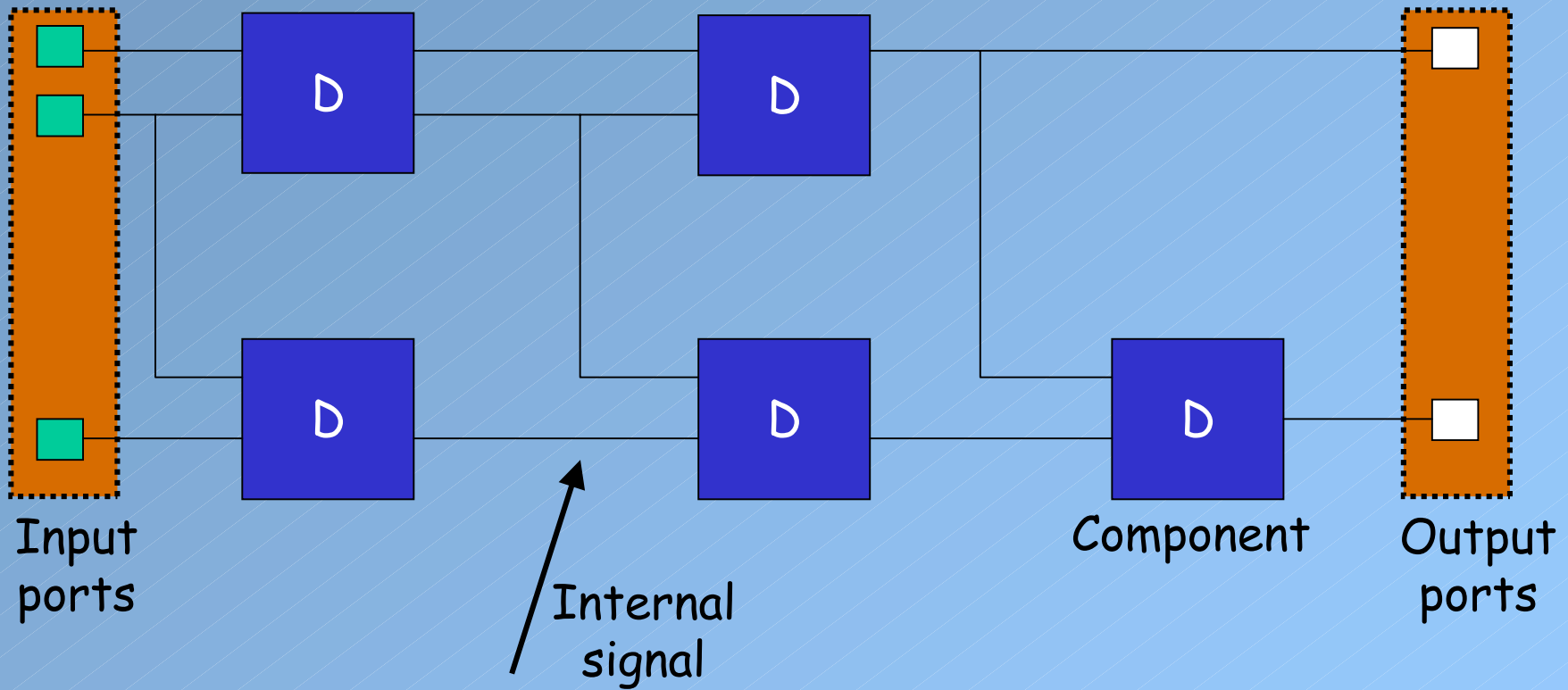
- `signal reg0,reg2:`
  `std_logic_vector(31 downto 0):=x"12345678";`
- In some old simulators, hexadecimal values must be converted to the type `std_logic_vector`.
- The function `to_stdlogicvector()` is available in std_logic_1164 package.
- `signal reg0,reg2: std_logic_vector(31 downto 0):=`
  `to_stdlogicvector(x"12345678");`
- CAD vendors offer many type conversion functions with different names in packages
- Those kinds of packages are usually in IEEE library.
- Check IEEE library.

VHDL Models with CSAs

- Construct Schematic
  1. Model each component (e.g. gate) as a delay element.
  2. Draw a schematic interconnecting all the components. Uniquely label each component.
  3. Identify the input signals as input ports.
  4. Identify the output signals as output ports.
  5. All remaining signals are internal signals
  6. Associate a type with each input, output, and internal signals (e.g. std_logic)
  7. Make sure that each input port, output port, and internal signal are labeled with a unique name

An Example Schematic

- Delay element model of a digital system



A Template for VHDL with CSA

```
library library-name-1, library-name-2;
use library-name-1.package-name.all;
use library-name-2.package-name.all;

entity entity_name is
port (input signals: in type;
      output signals: out type);
end entity entity_name;

architecture arch_name of entity_name is
-- declare internal signals
-- you may have multiple signals of different types
signal internal-signal-1: type := initialization;
signal internal-signal-2: type := initialization;

begin
-- specify value of each signal as a function other signals

internal-signal-1 <= simple, conditional, or selected CSA;
internal-signal-2 <= simple, conditional, or selected CSA;

output-signal-1 <= simple, conditional, or selected CSA;
output-signal-2 <= simple, conditional, or selected CSA;

end architecture behavioral;
```

Block Statement

- It disables signal drivers by using guards

```
block-label: block[(guard-expression)][is]
        [block-header]
        [block-declarations]
begin
        concurrent-statements; -- any number or none
end block;
```

- If a *guard-expression* appears in the block statement, there is a signal called *GUARD* of type Boolean within the block.

```
B1: block(STROBE='1')
begin
        Z <= guarded not A;
end block B1;
```

- CSA statement is executed if the implicit signal *GUARD* is TRUE. In the example, Z gets the value of *A'* when STROBE = '1'.

Block Statement

- The signal GUARD can be used explicitly.

```
B2: block(CLEAR='0' and PRESET='1')
begin
        Q <= '1' when not GUARD else '0';
end block B2;
```

- Rising-edge triggered D flip-flop.

```
entity d_flip-flop is
port(d, clk: in std_logic; q, qbar: out std_logic);
end entity d_flip_flop;

architecture behavioral of d_flip_flop is
begin

L1: block(clk='1' and not clk'STABLE)
signal temp: std_logic;
begin
   temp <= guarded d;
   q <= temp;
   q <= not temp;
end block L1;
end architecture behavioral;
```

Delay Models

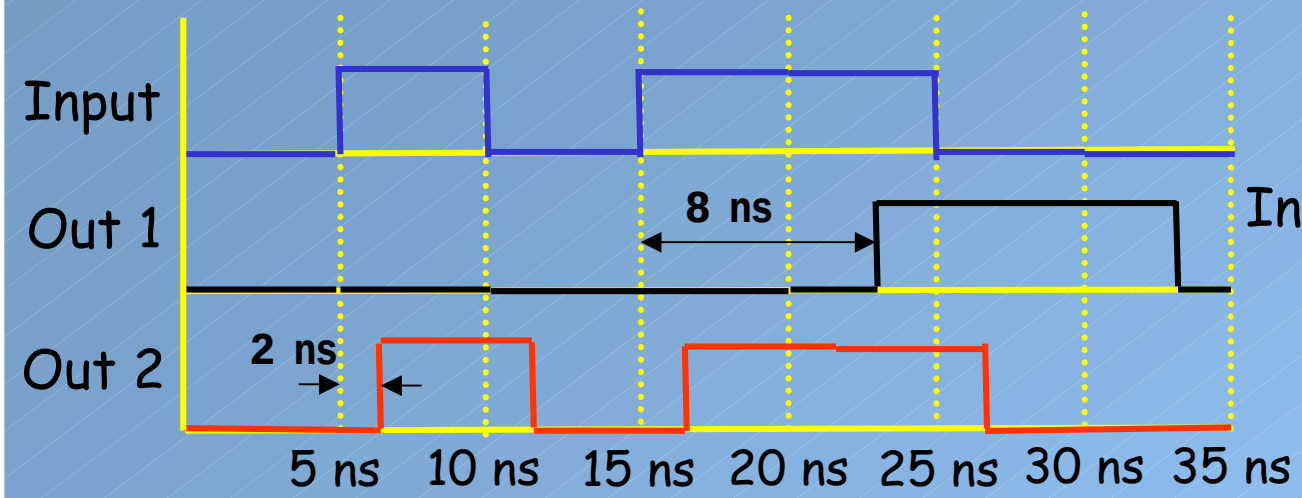
- Propagation delay is an important aspect of a component
- We need various delay models to accurately represent the behavior of digital circuits.
- These are

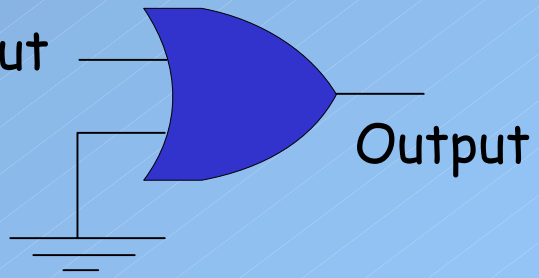
1. Inertial
2. Transport
3. Delta

Inertial Delay Model

- Digital circuits have inertia.
  - It takes a finite amount of time and energy for the output of circuit to respond to any change on the input.
  - The change on the input (event) has to persist for certain period of time in order the output to respond.
  - Otherwise, there will be no change at the output corresponding to the event.
  - This inertial delay model is the default delay model for VHDL programs.

Inertial Delay Model: Example





- `Output <= Input + 0` **`after`** *`propagation-delay`*`;`
- Out1 is the output waveform for delay = 8 ns
- Out2 is the output waveform for delay = 2 ns

- Any pulse with a width of less than the propagation delay through the gate is rejected.

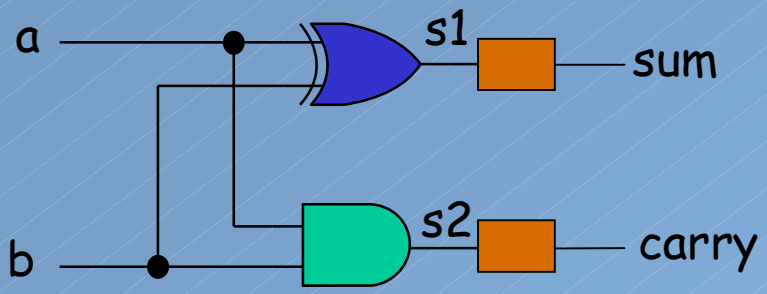
Inertial Delay Model

- If the event does not persist for the duration of pulse rejection width the output does not respond
  - The VHDL language uses the propagation delay through the component as the default pulse rejection width.
- If we know the pulse rejection width of a component, we can use it
- `sum <= reject 2 ns inertial (x xor y) after 5 ns.`
- General form in VHDL'93
  `signal <= reject` *`time-expression`* `inertial` *`value-expression`* `after` *`time-expression`*`.`
- We cannot specify pulse rejection width in VHDL'87. Delay value is used for this.

Transport Delay Model

- We use transport model for components with no inertia.
  - e.g. accurate modeling of wire delays
  - With small feature size in modern day circuits, it is important to model the propagation delay of wires.
  - Any event is propagated to the output. The change does not have to persist.
  - Even very short pulses are propagated in this model.
  - `sum <= transport (x xor y) after 5 ns;`
  - Transport model is not commonly used.
  - Default delay model is inertial.

Transport Delay Model: Example

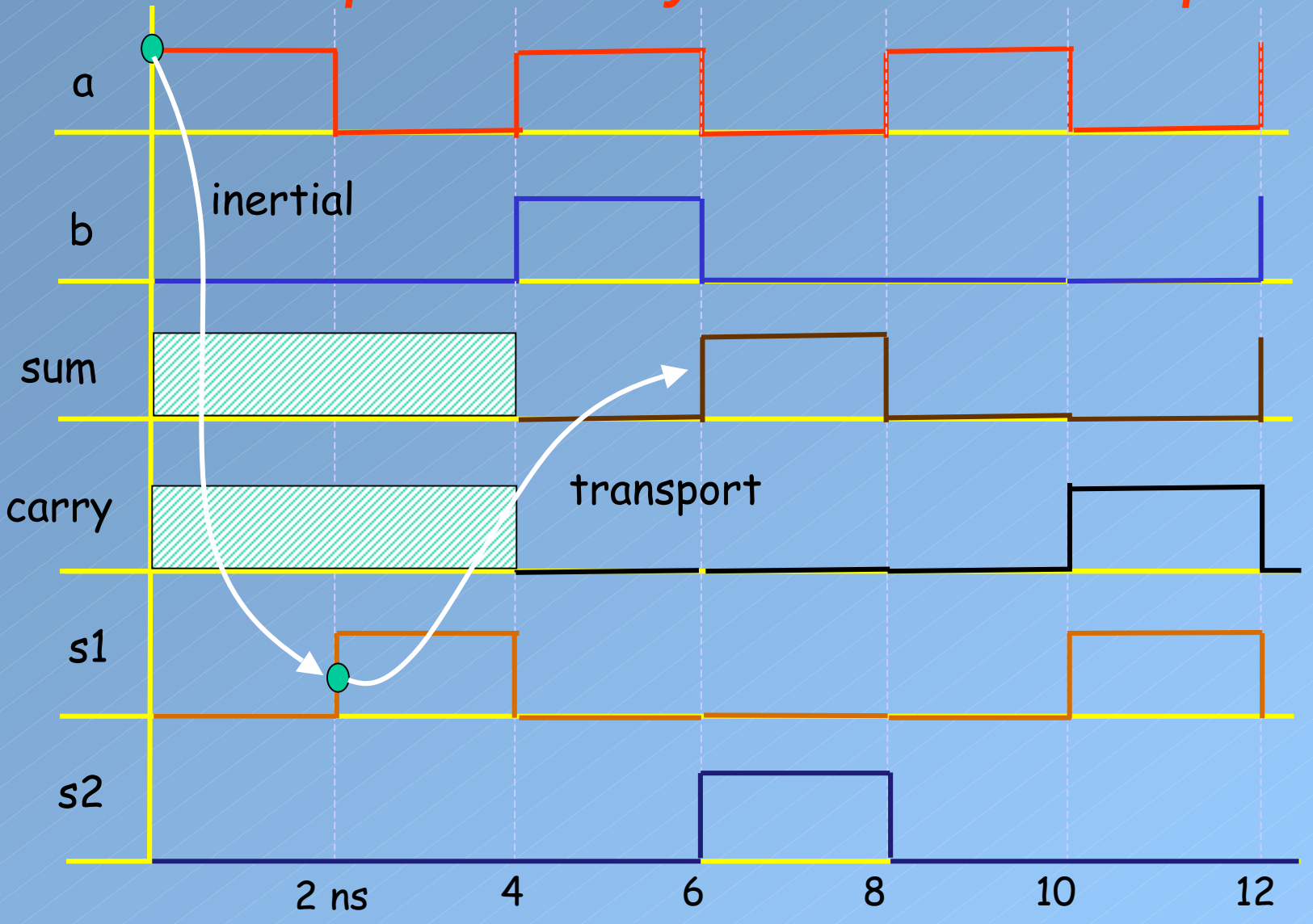


- Modeling wires as delay elements
- delay type is **transport**

```
library IEEE;
use IEEE.std_logic_1164.all;

entity half_adder is
port (a, b: in std_logic;
      sum, carry: out std_logic);
end entity half_adder;

architecture transport_delay of half_adder is
signal s1, s2: std_logic := '0';
begin
s1 <= (a xor b) after 2 ns;
s2 <= (a and b) after 2 ns;
sum <= transport s1 after 4 ns;
carry <= transport s2 after 4 ns;

end architecture transport_delay;
```

wire delay is 4 ns

Transport Delay Model: Example



What if we used inertial delay for wires?

Inertial vs. Transport

- Which model is being used depends on the component
  - For example, assume that we have a model of board level design, and VHDL models for chips on the board.
  - We may use transport model for the delay experienced by signals connecting chips on the board

Delta Delays

- What if we do not specify any delay?

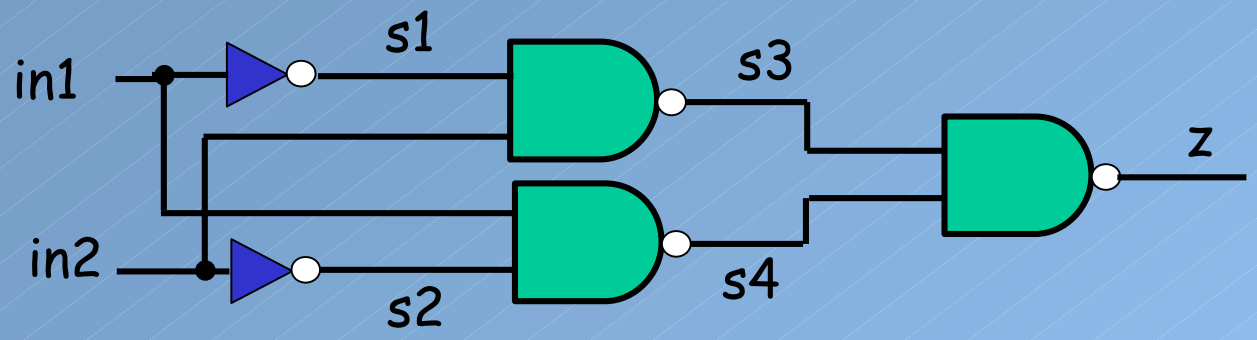
  `sum <= (x xor y);` ≡ `sum <= (x xor y) after 0 ns;`

- we may choose to ignore delays when
  - we do not have realistic delay for the components
  - we are not interested in physical timing behavior
- How can we preserve the concurrency in case of no propagation delays for components
  - For functional correctness, we must maintain the correct ordering of events
  - VHDL language provides a infinitesimally small delay called delta delay.

Delta Delays

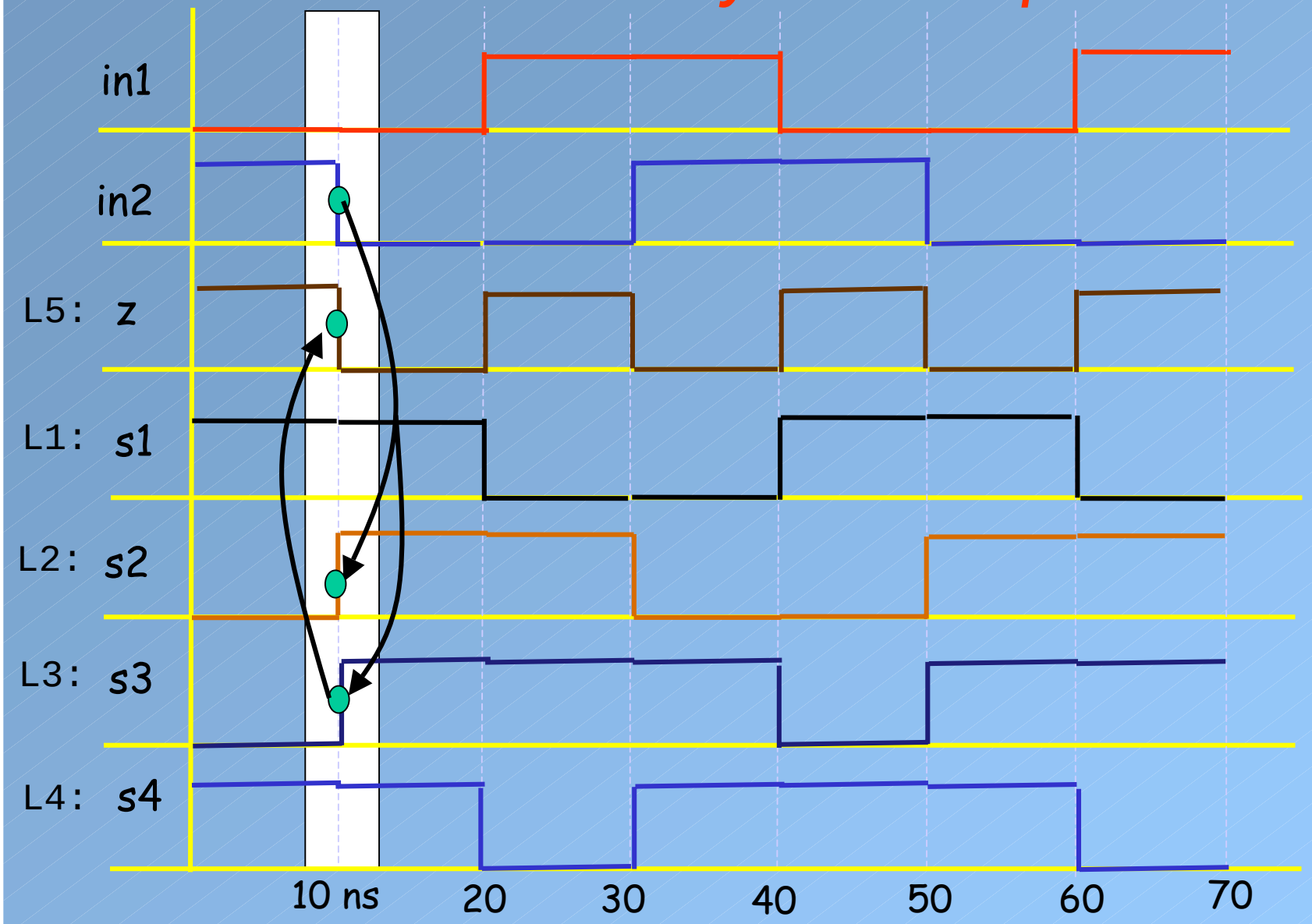
- Assignment takes place a delay of Δ after the RHS is evaluated.
  - If events with zero delay are produced at timestep T, the simulator simply organizes and processes events in time order of occurrence:
  - Events at $T+\Delta$ are processed first,
  - then events at $T+2\Delta$
  - followed by the events occurring 3Δ seconds later.
  - While the time advances with steps of Δ seconds, the simulation time does not actually proceed.

Delta Delays: Example



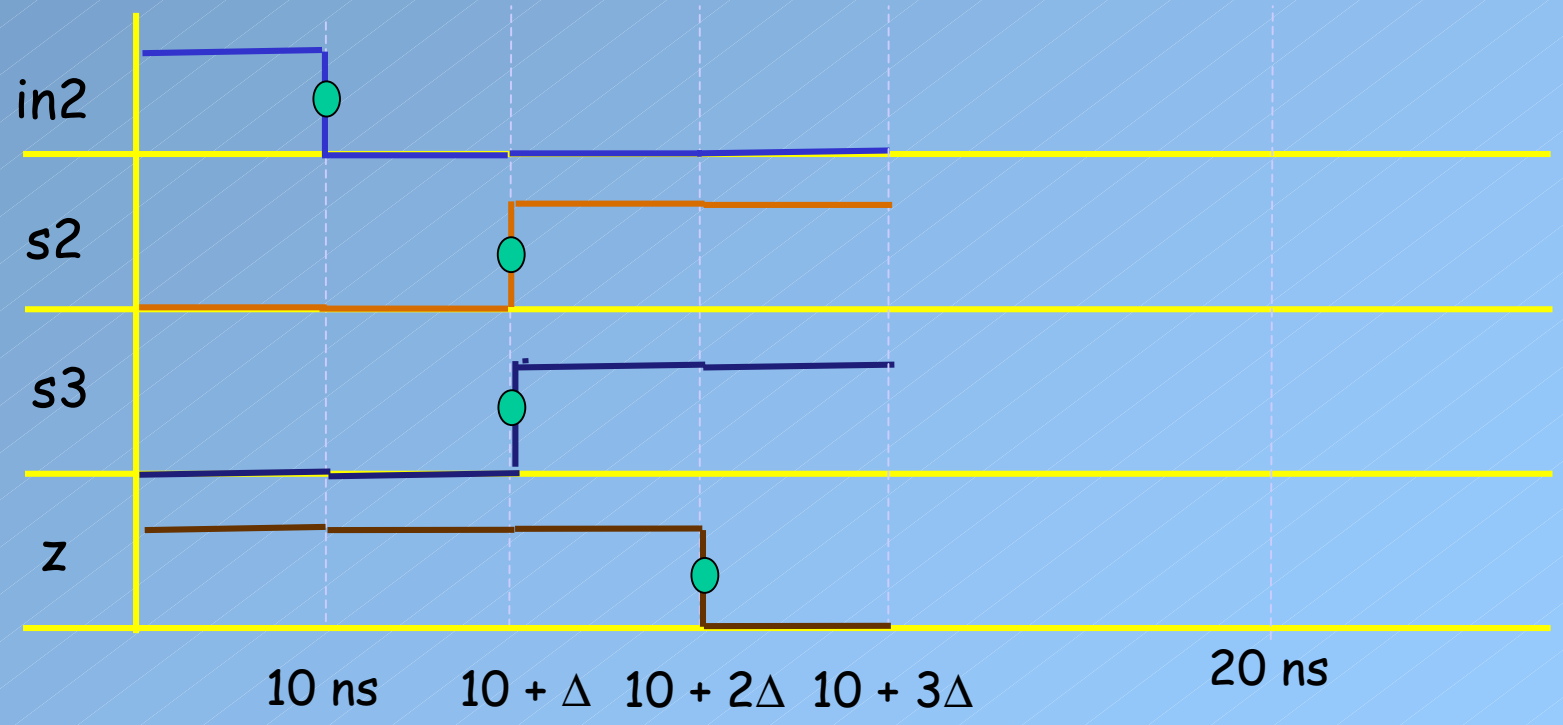

```
library IEEE;
use IEEE.std_logic_1164.all;

entity combinational is
port (in1, in2: in std_logic;
      z: out std_logic);
end entity combinational;

architecture behavior of combinational is
signal s1, s2, s3, s4: std_logic := '0';
begin
L1: s1 <= not in1;
L2: s2 <= not in2;
L3: s3 <= not (s1 and in2);
L4: s4 <= not (s2 and in1);
L5: z  <= not (s3 and s4);
end architecture behavior;
```

Delta Delays: Example



Delta Delays: Closer Look

- At time 10 ns, in2: 1 → 0
- This event leads to another events triggering an event on Z.
- event on Z occurs at the same time the event on in2
- Simulator does not show the actual trace as below.



Delta Delays

```
library IEEE;
use IEEE.std_logic_1164.all;

entity combinational is
port (in1, in2: in std_logic;
      z: out std_logic);
end entity combinational;

architecture behavior of combinational is
signal s1, s2, s3, s4: std_logic := '0';
begin
L5: z  <= not (s3 and s4);
L2: s2 <= not in2;
L4: s4 <= not (s2 and in1);
L3: s3 <= not (s1 and in2);
L1: s1 <= not in1;
end architecture behavior;
```

- Textual order is not important
- Flow of signals counts

Summary

- Entity and architecture constructs
- Concurrent signal assignments (CSA)
  - simple concurrent assignments
  - conditional concurrent assignments
  - selected concurrent assignments
- Constructing models using CSA
  - modeling events, propagation delays, and concurrency
- Modeling delays
  - inertial
  - transport
  - delta

Summary (cont)

- Signal drivers and waveforms
- Shared signals, resolved types, and resolution functions
- generating waveforms using waveform elements
- events and transactions