23-11-2024

→ Vijay

Top → Down → rather than opp.

State Machine →

phrasing (finite operator) in compiler.

$A \times B \neq B \times A$

quintiple

Σ - input alphabet → set of indivisible unit

Γ - output alphabet

S - set of state → history of i/ps - that are relevant

δ - function that maps. $\Sigma \times S \to S$ → cartesian product → cross product** → set of ordered pair → maps i/p set to o/p set

ω - function that maps $\underbrace{\Sigma \times S \to \Gamma}_{\text{Mely}}$ / $\underbrace{S \to \Gamma}_{\text{Moore}}$

Language.

BNF - Backus Naur Form.

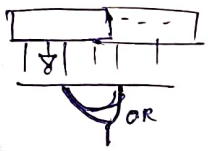
grep → global regex parser.

{ fullname :: == firstname, middlename }

↓

token

Shift Reduce Parsing



tokens are shifted in

↓

Rule match } → Syntax match, Semantic match

↓

Reduce

→ shift reduce conflict
reduce reduce conflict } context free grammar

Sanskrit

child eat

child eat an apple

LALR(1)

↳ Look ahead

• Abstract Syntax tree

Homeset - sound or letter.

Error - correction code RS (544, 514)

Cache
- LRU
- MFU
- FIFO

Galois Fields.

Set → Group → Ring → Field → Polynomial → Vector space

with, within

5

07-12-2024

Software for automating tasks. → Synth
→ Simulation
→ GDS error check

Algebra → a set with operation & rules.

Language

- Symbols, not-Symbols (empty space)
- Strings, ~~[illegible]~~ → meaningful string
- Lexicon – set of meaningful words
- ~~[illegible]~~ Sentence –
- Stanza –
- Grammer → set of rules in a lang
- Token – read till space
- pharsing → extracting meaning from it.
  - → Scanner - lexical analyzer.
  - → Lookup engine → Symbol table (CAM - Content addecable memory)
  - → Syntatic analyzer → checks with perdefined rules.
  - → Sematic analyzer → checks for valid constructs
  - → AST → ~~[illegible]~~ parses stores in ←

Program

AT&T → lex → flex (GNU)

→ yacc → bison

↳ LALR1 → Look ahead 1 parser

Bad grammer

→ ambiguity in grammer.

Compiler

→ translation from one to other

C → assembly

.V → .C

Language.

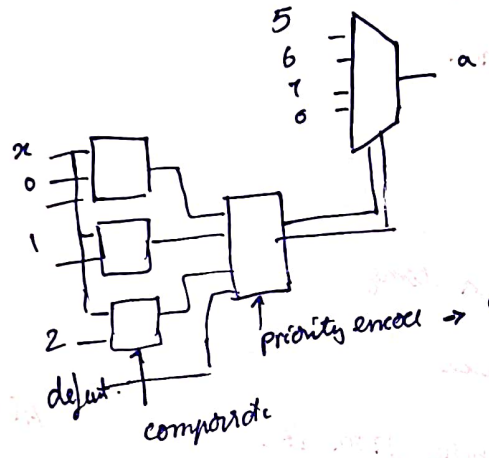
Lisp → Emacs Lisp.

↳ unambiguous grammar

Verilog constructs Combinational circuit

assign Y = (a & b) | c;


```
if (x==0)
  a<=5;
if (x==1)
  a<=6;
if (x==2)
  a<=7;
else
  a<=a;
```



→ compare with all lest priority before match,

```
case (x)
  ...
  ...
  ...
endcase
```

⇒ Infers parallel mux (regular)

Sequential circuit

always @ (posedge clk)

↓ sync / async reset

↓ ASIC (prof) ↓ FPGA's

use of begin end everywhere → edit code → best case syntax error; worst case error in Silicon.

// comments → why? part to explain
→ don't comment the operation.

function, task
loop → for, generate
class X

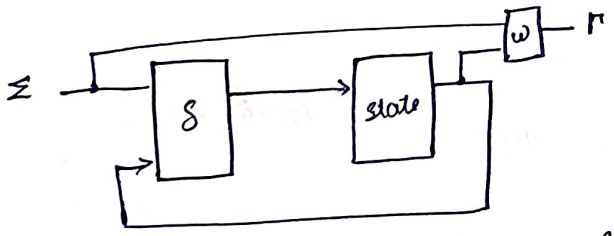
} → using don't provide 1→1 mapping with netlist infered, create a layer of abstraction

ECO → Engineering change order (change made to silicon dye after implementation done)
↳ using spare gate, wire

→ Final netlist usually single bit instead of bus[n-1:0]

28-12-2024

State machine building from case, if, always @(posedge clk)



δ - same
ω - different

Style 1 - 3 process

always @(*)
[δ]

always @*
[ω]

always @(posedge clk)
transition [state]

=

Style 2 2 process

always @*
[δ], [ω]

always @(posedge clk)
[state]

≠

Style 3 1 process.

always @(posedge clk)
[δ, ω, state]

Difference in style 2 & style 3 ⇒ cannot be such that it should be made not violate the protocol timing diagram to be met in both cases.

Style 2

1.- infer latch ⇒ mismatch synth simulation, mismatch
synth → differ
simulation → times.

2. - can miss, harder to debug.

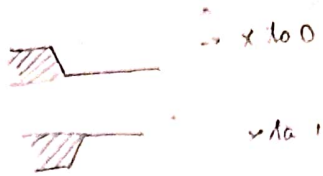
3. O/p should be REGESTERED
⇒ directly adding Flop violates protocol
⇒ O/p should be present in prev state so that O/p available at 'flop' that makes 1 clk delay.

Style 3

1. Output from ~~latch~~ flop
** ⇒ Timing path met
t_{sh}, t_h
⇒ Registering input & o/p

UVM -> Reusability, adds layer of abstraction from hardware.

Hardware vs Software



Concurrency

- single thread, cannot simulate parallelism without considerable complications.
- not just linearly dependent, circular dependency.

event queue → Sensitivity list →

called when that condition met

- list of sensitivity

↳ Event queue

to simulate hardware

RHS

(b, c), @(*)

⇒ infered sensitivity from control

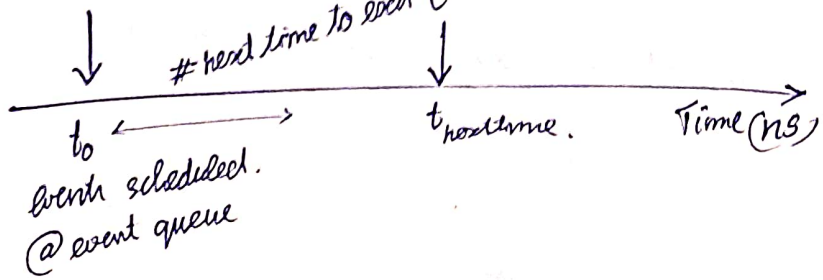
assign a = b & c

gets triggerd only when change in sensitivity list

Event queue

- to wait for the other changes to occurr that inputs depends on can change
- assign Y = a || b ||⇒ a, b RHS change can occur in other assign to be holded till its clear to go

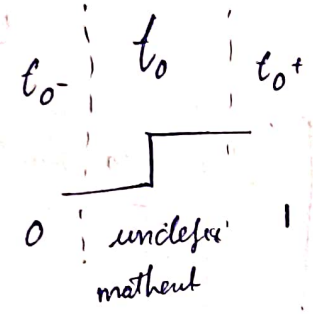
(delay given to simulator to know when to simulation at next time $t_{nextime}$)



Event queue

Simulation δ → instance - seq
→ instance - seq
→ instance - seq.

Sampling → when sensitivity @ posedge & value has posedge ↑ negedge ↓ (or) changes at sensitivity.



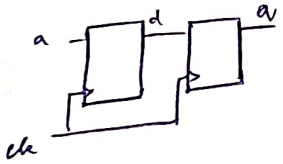
$t_0 - t_{0^-} = 0$

$t_{0^+} - t_0 = 0$

↳ depends on sensitivity @ pos/negedge ... ⇒ clk

when it happens

Causality



⇒ d previous value will be sampled.

if clk has no effect (independent)

t_{0^+} is sampled ⇒ Verilog construct

mathematical convention

cause can never see its effect

** in verilog test bench

→ # delay events that are independent, of clk create ambiguity

→ @ posedge clk ⇒ clears that confusion and create proper sampled ⇒ clears synth - sim mismatch earlier

In real world t_{setup} t_{hold}, t_{clock} → At, the signal new value will be never known.

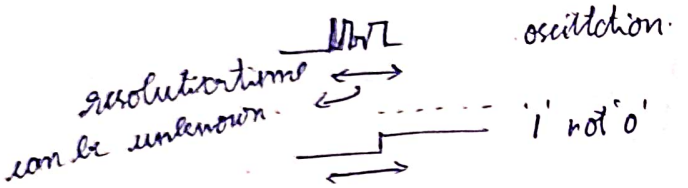
→ Also in simulation if the change in an effect of it t_0^- is taken

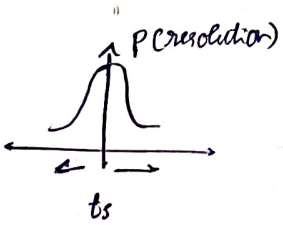
causality & effect

<u>Resolution time</u> → to settle to known state

↳ when violated, when o/p will be resolved in unknown

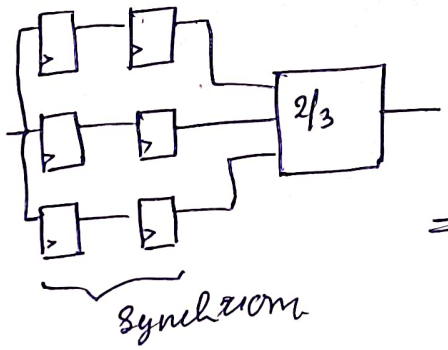
weather resolvable (or) not is not sure.





$2\tau_{cq}$ ⇒ will be resolved

→ parallel metastable stabilizer.



⇒ redundancy to resolve metastability

12-01-2025

→ Error handeling ⇒ & reporting.

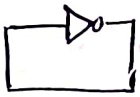
→ logic to catch multiple error. → using other branch in AST

→ Error recovery. → (Not always possible)

⇒ comments, indentation,

→ Cycle based simulator.
→ faster, doesn't consider. δ state

Event queue



Arbitration, Priority Encoder, Data with conversion ... 9 topics.
↓
harder with num that are not multiple.

M×N

PISO → +

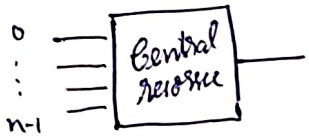
SIPO → + part select.

→ M & N are variable

↓ JESD 204B Implementation.

Arbitration

→ history (time)



- cache, network switch scheduler.
- & scaled up version  multiple resources, some participants.

- metered traffic, priority, data size, seq. time
  ↓
  VoIP (QoS)

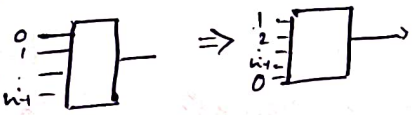
Based on
Priority, Round Robin, Least recently used, Least frequently used
(time)

Priority encoder

→ Deterministic way.

↳ Random → Non-deterministic

Priority
↳ give unfair advantage to lesser bits



→ mitigated by rotating priority
↓
round robin

All have commonality
⇒ counter, measure age.

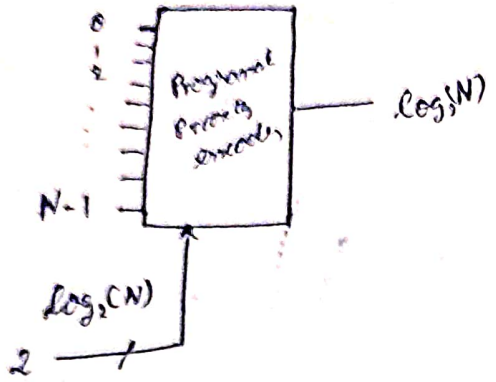
→ search & sort ⇒ traverse the list, select that match
↓ priority encoder. ↓ moving. addre..

Recursion in hardware → possible but with fixed depth.

to have 100% utilization validation / 1ns → billionth of sec.
calc

(programable priority encoder)

→ update lists to priority encoder before & select @ execution time.



- priority encoder - tree of selection delay problem
- programable priority encoder
  → compondi complex
  ↳ to implement <u>Round Robin</u>

- Priority encoder in hardware → CAM (Content addreable memory)

<u>Content addreable memory</u>
→ with multiple matcher (pick one according to logic)

Big(O)

Classical priority encoder $O(n)$

Programable priority encoder $n(O(n))$
→ un-optimized → n No of priority encoder. with enable lines.
un winding the loop.

<u>Iteration</u> vs <u>Recursion</u>

$a_0\ a_1\ a_2\ a_3 \ldots$

sum = 0

$\underbrace{\overbrace{a_0\ a_1}^{S_{00}}\ a_2}_{S_0}\ \underbrace{\overbrace{a_3\ a_4}^{S_{01}}\ a_5}_{S_1}$

$S = S_0 + S_1$

```
arr_sum (arr, N)

if (N <= 0)
    return 0
else
    arr_sum (arr, N-1)
```

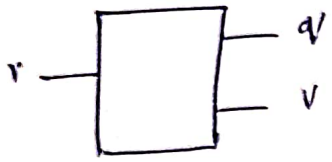
Tower of hanoi → recursive solution easy, iteration tree not possible.

Tolerate in humans, remove in device

hair in hen,

1 i/p priority encoder

requestor, output (q), valid



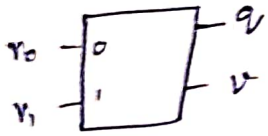
q : 0

v = 1 when r = 1

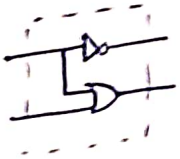
2 outputs.

2 i/p priority encoder.

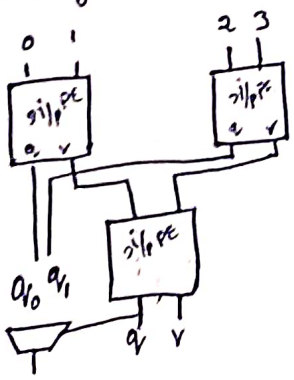
r_1	r_0	q	v
0	0	1	0
0	1	0	1
1	0	1	1
1	1	0	1



2 outputs



Used as building blocks to make priority encoder



MSB	LSB
L/R	L/R 0
	L/R 1

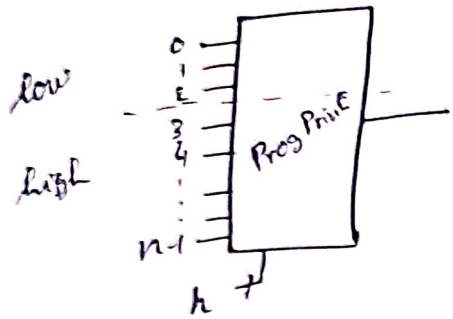
This changes it to $n(a(w)) \Longrightarrow O(\log_2(N))$

in programable priority encoder conflict occur when program

↓

similar conflict in CAM

2 matches.



One flaw

⇒ Implementation of binary tree makes

0	1	2
25%	25%	50%

instead

33.3%	33.3%	33.3%

(+) random shuffler in i/p & o/p makes it fair. (Propability)