day/date

1/2/23

# Digital System Design

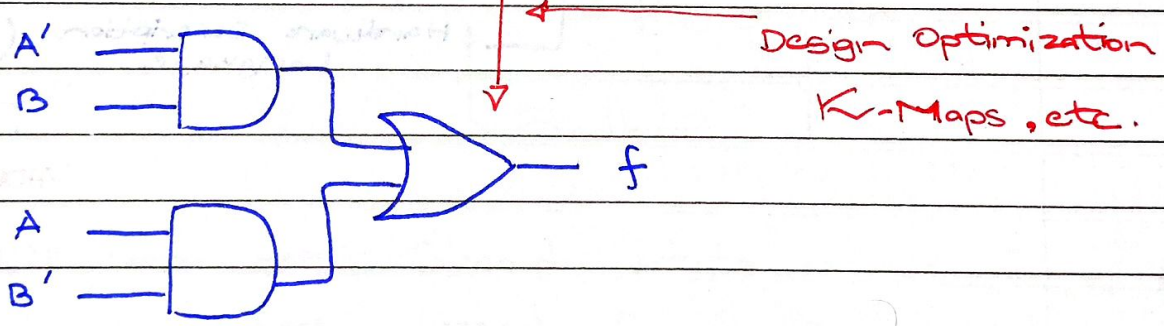
## Recap

### Half Adder

»» Truth Table

| A | B | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

»» $f = A'B + AB'$ (SoP)




### Full Adder

»» List out truth table

»» Write expression

»» Draw circuit

day/date

Problem Statement
↓
Algorithm
↓
Expressing Algorithm
(C / C++)
↓
Compiler
↓
Assembler

} Design Process

»» Textual (way of expressing problem)
└ Language
└ Hardware Description Language (HDL)
VHDL Verilog
↓
System-Verilog

»» Flow

Problem Desc. → FPGA CAD (Synthesis) »» Netlist → FPGA

- Simulation for verification
- Synthesis for netlist

/date

2/2/23

# Digital System Design

## Verilog

T.T => Expression => K Maps

Design Methodology

Netlist

Implementation

```
module example (a, b, c, y)
    input a;
    input b;
    input c;
    output y;

endmodule
```

## Behavioral HDL

→ Writing gate level code for large scale designs is tedious

→ "assign" : assign f = x1 & x2

ay/date

8/2/23

# Digital System Design

## Levels of Abstraction

- Gate Level
  - Gate primitives
  - Need to have netlist / circuit in mind

- Behaviour
  - Expression { LHS is being continuously driven ; assign LHS = RHS }
  - Behaviour { for (loops) , if...else , case }

→ assign : combinational circuits

→ Procedural assignment statements : [ if... else, for, case, etc. ]

→ always @( ... )
↑ sensitivity List / event control

Example: Full - Adder

sum = a ⊗ b ⊗ c

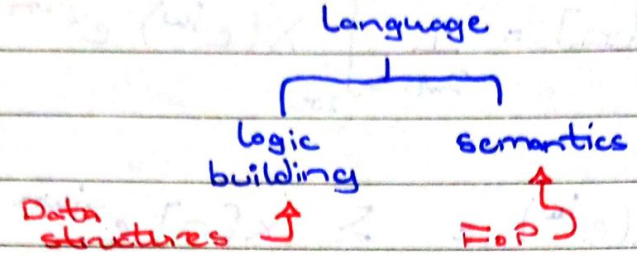
carry = a.b + b.c + c.a

| c | b | a | carry | sum |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

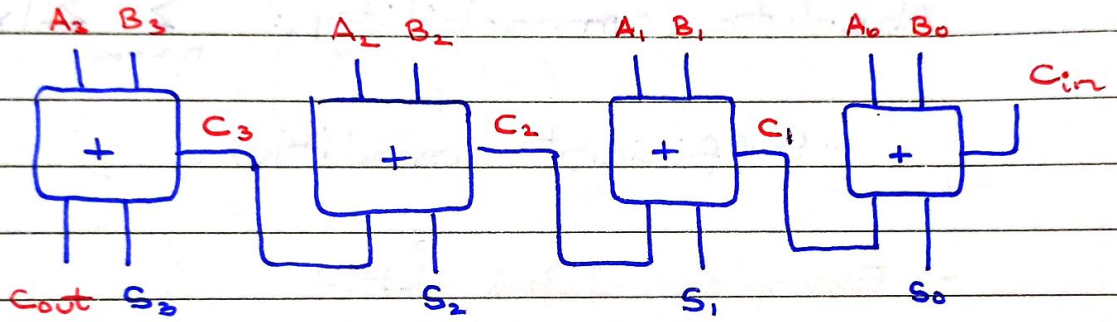
date

9/2/23

# Digital System Design




Adder (Extension by connecting one bit wide adders)




Leaf node

make a single fulladder module and instantiate it 4 times, with corresponding inputs/outputs

- Synthesizer looks at the whole structure and then outputs the circuit
- not sequential, unlike the compiler

→ loop should be in the always @() block
LHS variables should be of type reg

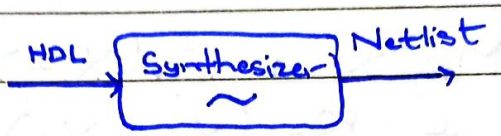
→ for .... loop index should be of type integer; cannot be anything else unless using generate

→ generate .... endgenerate ; index of type genvar

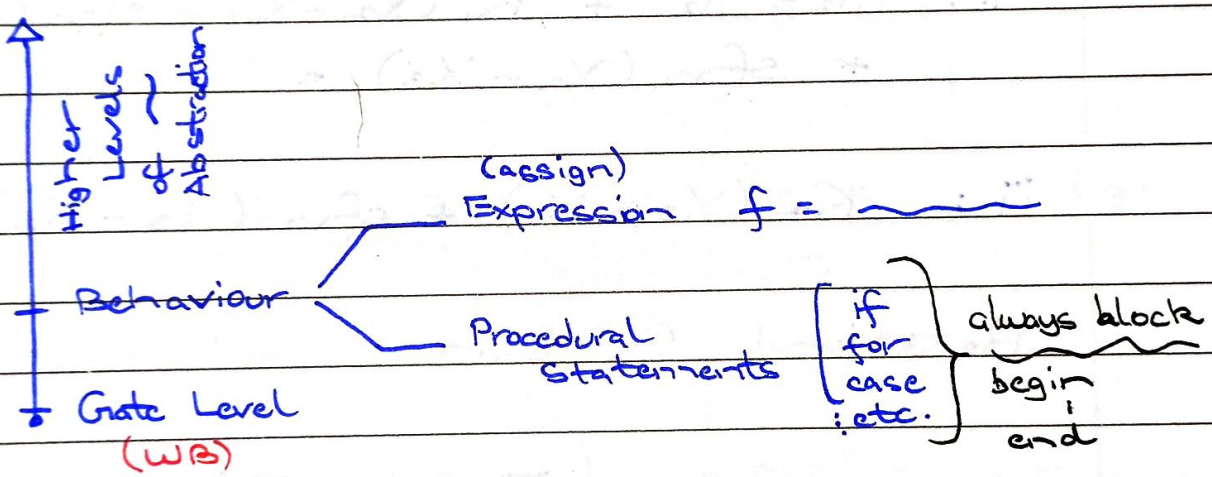
→ addbit is the stage name

→ generate masks the gates in RTL viewer ; no structural changes, only the way it is presented

→ keep type reg reserved for always @() blocks




## Levels of Abstraction




## Variables

- wire - intermediate connections
- reg - net output type
- genvar - loop index { generate block }
- integer - loop index { always block }

15/2/23

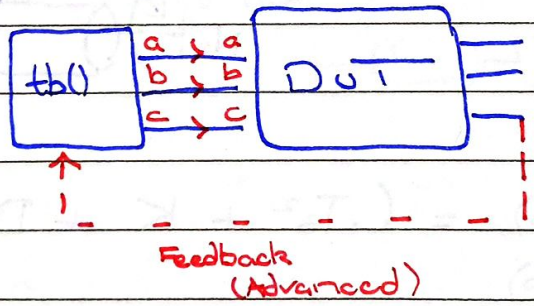
# Digital System Design

## Testbench

RTL ↔ Simulation ↔ UPF

- Does not typically have inputs / outputs.
- DUT must be instantiated in the testbench.

1. Declare module tb(); // No I/O
2. reg for input to DUT, wire for output
3. DUT instantiation
4. Initialization {intial begin ... end}




## Sequential Circuits

Combinational + Memory = Sequential

/date

16/2/23

# Digital System Design

- each always @(*) block has a separate hardware footprint inferred by the synthesizer
- order of statements inside always @(*) block matters
  - LHS = reg ; RHS in event control
  - last statement to assignments in an always @(*) block is executed, provided the reg on LHS is same

- Blocking Assignment (=)
  - Blocks progress ; Next waits for prev.
- Non-blocking Assignment (<=)
  - All assignments will be made at the same time

```
always @(*)                    Assume all inputs to be zero
    begin
        p   = a ^ b            // p = 0 , 1
        g   = a & b            // g = 0 , 0
        s   = p ^ cin          // s = 0 , 1
        cout = g | (p & cin)   // cout = 0 , 0
    end
```

- a goes to 1

With non-blocking, originally, the same outputs are yielded.

EXCELLENT

/date

When a goes to 1, non-blocking statements observe a state delay of one cycle.

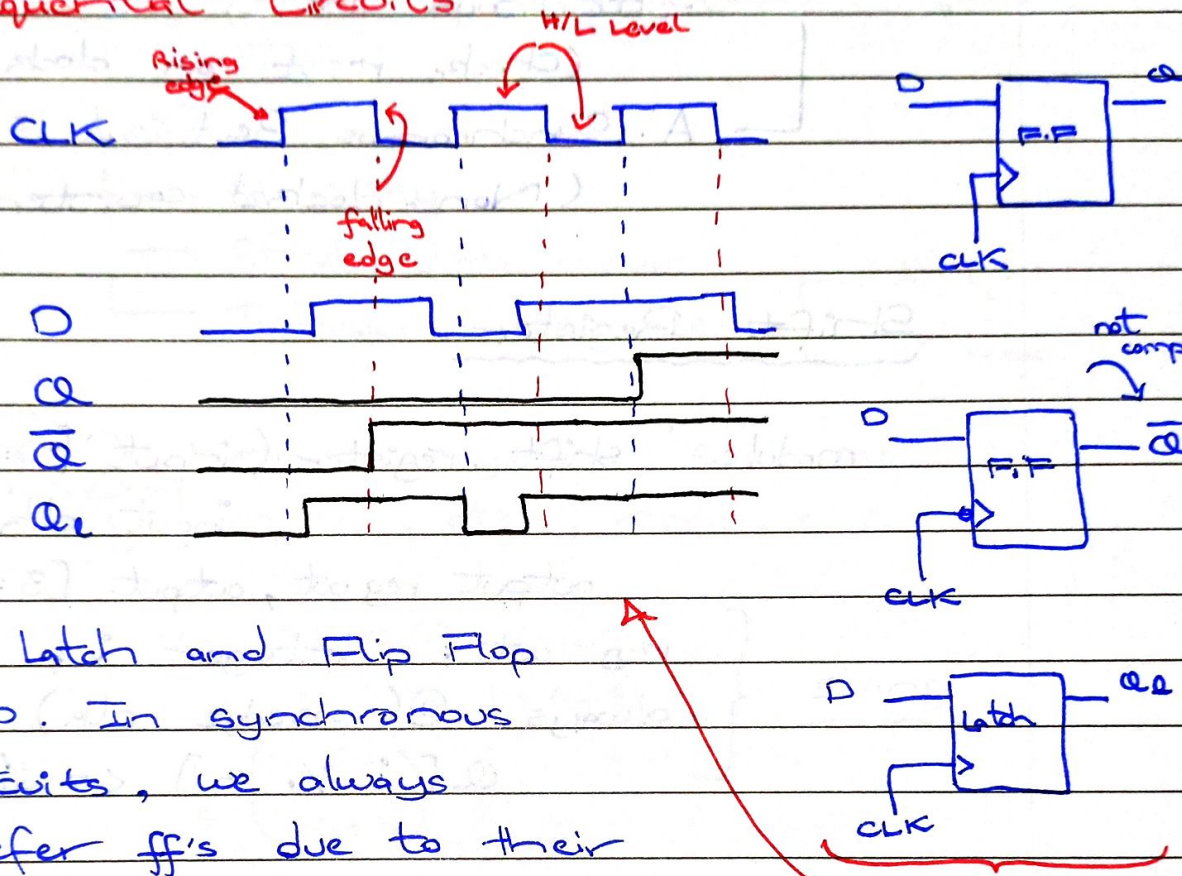
- Reserve blocking statements for combinational circuits.

/date

22/03/23

# Digital System Design

## Sequential Circuits




D-Latch and Flip Flop O/P. In synchronous circuits, we always prefer ff's due to their ability to mitigate glitches.

→ always @(posedge clk
negedge ...) to make a clocked always block.
upon detecting either keyword, synth. will make LHS variables outputs of a flip flop

→ Latches can be inferred using an if statement, however, without else (selection)

Reset (Drive o/p to zero regardless of i/p)

- Synchronous Reset
  (Check reset on clock state)
- A. Synchronous Rest
  (Non-clocked counterpart)

## Shift Register

```
module shift_register (input [0:0] in,
                       input clk, rst,
                       output reg out, output [3:0] Q );
```

ignore

```
reg out; integer i;
always @(posedge clk) begin
    Q[i % 3] <= Q[i % 3 ~]
```

```
reg out; reg Q;
always @(posedge clk) begin
    Q[1] <= in;
    Q[2] <= Q[1]
    Q[3] <= Q[2]
    out  <= Q[3]
end
endmodule
```

1/03/23

# Digital System Design

Summary
- GATE structure — and(), or() ...
- Expression — assign ^ & | ...
- Behavioral — always @()
  - Sequential
  - Combinational
  - if .. else
  - for
  - case

output must be of type reg

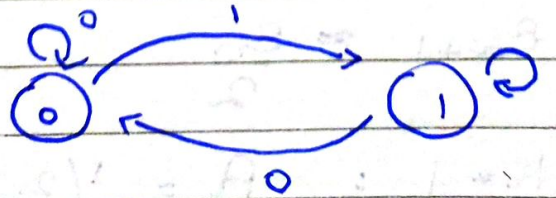
-> Combinational circuit : Event control can be replaced by *

-> Sequential circuit : Explicitly state posedge / negedge in event control

## Finite State Machines

- next state logic
- state register
- output logic

Example: w : speed indication z : control




Mealy ~~Moore~~ Machine (Woops)

ignore
revise
FSMs

| Present A=z | Input w | Next A=z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Moore: If output α present state only

Mealy: If output α present state + input

---

initial is reserved for testbench / sim

- exception is for loading bits into memory using $readmemb("file.txt", rom);
- use templates for boiler-plate code
  : RoM (clocked read template)
- use IP catalog for blackbox models of complex blocks

/date

8/03/23

# Digital System Design

In always @(posedge ....)
As many flops are inferred as unique number of LHS assigments

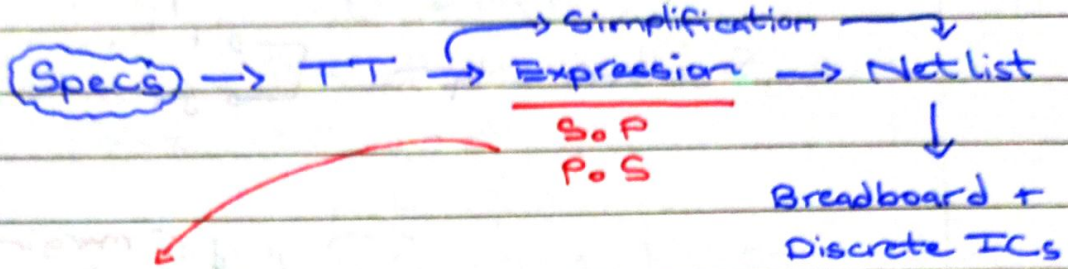
A lotta quizzes today mayne
It really do be like that when it be when it does. ;-;

date

9/03/23

# Digital System Design

## Programmable Logic Devices




| a | b | c | sum |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| ⋮ | | | ⋮ |

$a'b'c + a'bc' \ldots\ldots$

SoP

- requires lesser gates
- the way gates are built using transistors support SoP form

FPGA: offers reusability over discrete patching

- Design is about trade offs
  - ~ More flexible but less performance & vice versa

- JTAG: Joint Test Access Group

15/03/23

# Digital System Design

**CPLD** Contains several PAL-like blocks (PLA with fixed OR plane)

→ Interconnection : Architecture ensuring connection b/w PAL-like blocks

→ PAL-like blocks are connected / placed in 2D fashion

**FPGA**

Logic switch Connection

→ LB, SB and CB in 2D fashion

→ Two LB's are connected using CB's

→ Two CB's are connected using SB's

→ Logic Block is fundamentally a truth table (look-up table) in silicon form

I butchered the spellings, mb

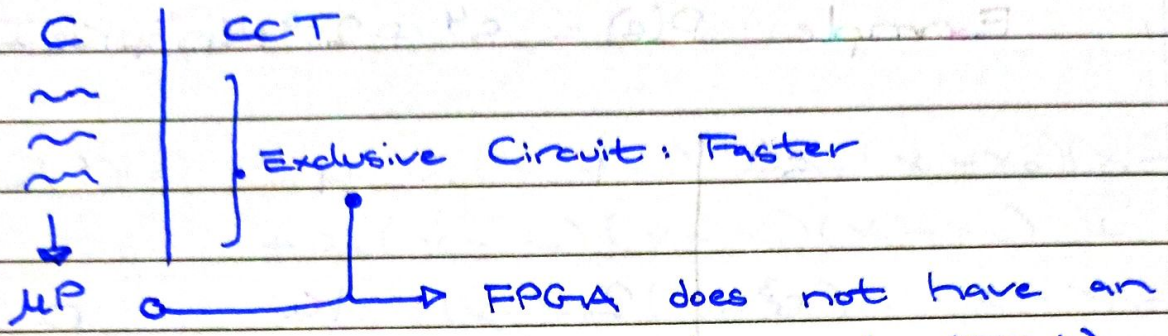
- FPGA
  - LB
  - Dedicated Resources
    - µProcessor

```
main () { ...
        ---   HOTSPOT : To be accelerated
        ---   (Exclusive circuit)
        ....
}
```

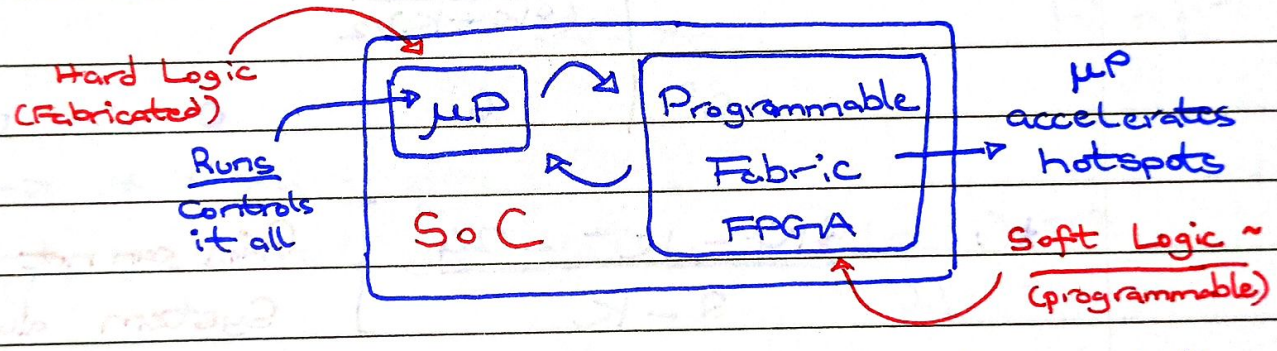
EXCELLENT

16/03/23

# Digital System Design




- ISA
  - captured by HDL
    - mapped to → FPGA, ASIC ☺☺




→ Hard logic (dedicated HW) will always (?) beat soft logic performance

ate

22/03/23

# Digital System Design

Gotcha!

## if ... else vs. case

```
-> always @(*)
   begin
      if s == 0
         a = b
      else
         a = c
   end
```

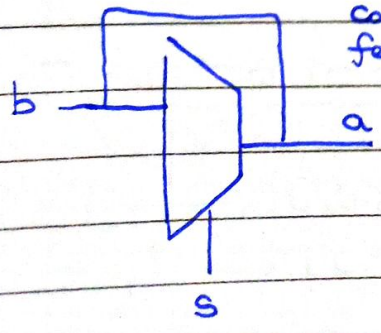
```
-> always @(*)
   begin case (s)
   1'b0: a = b ;
   1'b1: a = c ;
   default: a = xx ;
   endcase
   end
```

case → case x, case z — Don't use these lil' guys they bad ~

if there are more than two ~ i.e presence of if ... else if ... else : a priority encoder is inferred

| in | | | | out | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | x | x |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | x | 0 | 1 |
| 0 | 1 | x | x | 1 | 0 |
| 1 | x | x | x | 1 | 1 |

else absence : { LATCH }




combinatorial feedback path

incomplete if else or missing default

EXCELLENT

/date

## Circuit Timing

- Delay { 90% - 10% , 50% - 50%
  - $t_r$ : time it takes to reach 10% → to 90%

day/date

5/4/23

# Digital System Design

## Path Delay & Performance Analysis

- Critical Path
- Metastability
- Fmax
- Circuit Timing

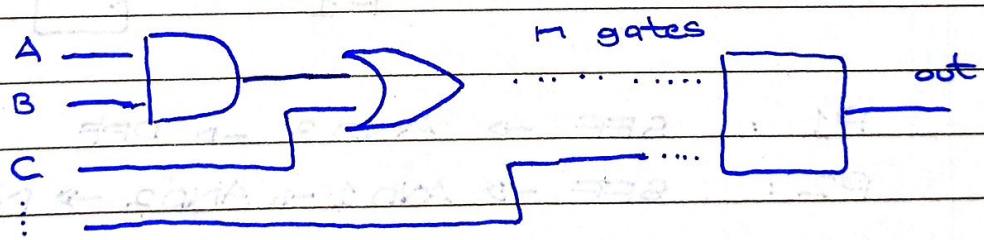
**Operators:**

{ + , - , × , / } → Arithmetic Circuits

root is addition of all operators

Gate Delay occurs due to stray capacitances made by overlap of two materials (n-p junctes.)

Longest propagation delay in the circuit {attributed to output}




delay = n × fundamental delay

To synchronize the circuit, we insert flip-flops clocked by a clock with period that of the longest path (critical path) {also fixes race conditions}

EXCELLENT

y/date

Critical Path -

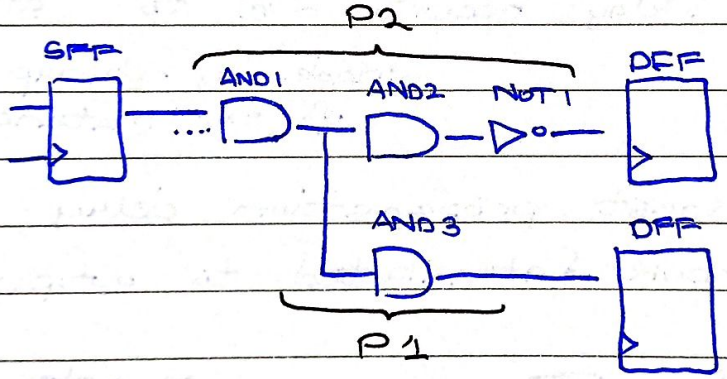
- Longest Path b/w the source and destination flops
- $F_{max} = 1/\text{critical Path}$
- Global Fmax is the largest Fmax of individual blocks in the design

$CP_{Global}$ { max ( $CP_1$, $CP_2$, .... $CP_n$ ) }

$F_{max\ Global}$ { min ( $F_{max_1}$, $F_{max_2}$ .... $F_{max_n}$ ) }

$T_{min}$ = minimum clock period

→ Short path delay } misinterp.




P1 : SFF → AND 3 → DFF

P2 : SFF → AND 1 → AND2 → NOT 1 → DFF

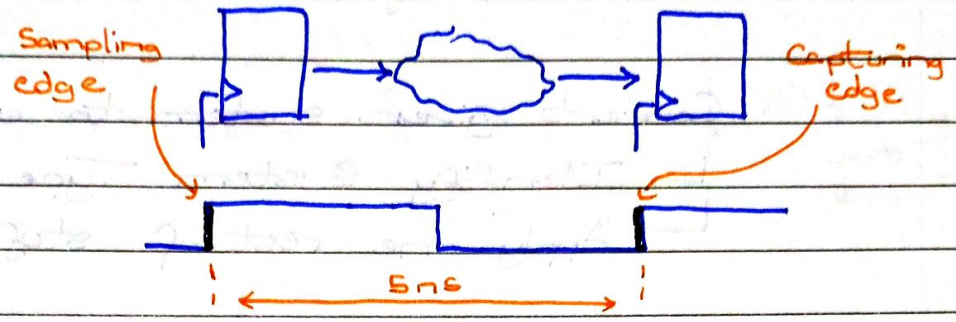
SFF's Posedge : Sampling edge

DFF's Posedge : Capturing edge

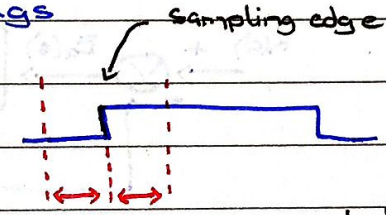
date

6/04/23

# Digital System Design




FF : 3 timings

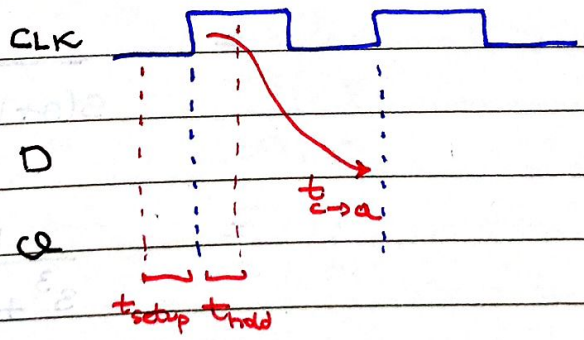



related to input

D (input) to FF should be stable in $t_{setup}$ / $t_{hold}$

Delay of FF itself

$t_{c \to q}$ ] related to output




If D changes within $t_{setup} - t_{hold}$, the system becomes metastable.

$t_{c \to q}$ : delay it takes for Q to appear after the sampling edge

EXCELLENT

day/date

15/04/23

# Digital System Design

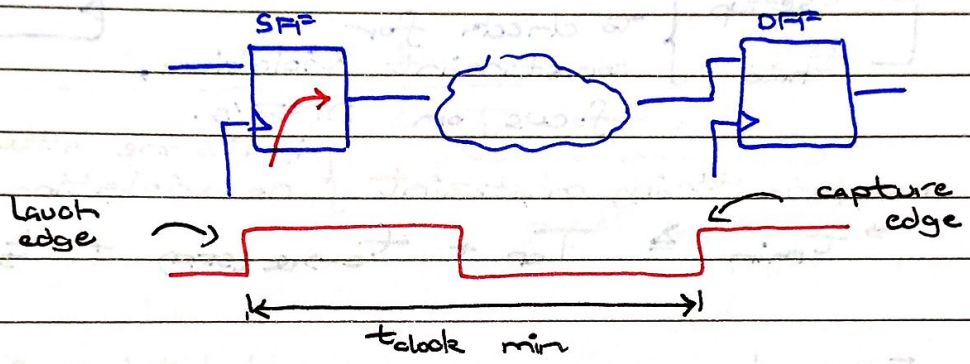
$t_{setup}$ & $t_{hold}$ : timing constraints

$t_{c \to q}$ : delay

→ $t_{clock\ min} \geq T_{critical\ path}$

$t_{clock\ min}$ : clock period

$T_{critical\ path}$ : delay of comb. circuit




**SFF**

- $t_{setup}$
- $t_{hold}$
- $t_{c \to q}$

× we're only interested in o/p of SFF

i/p has been sampled already

**DFF**

- $t_{setup}$
- $t_{hold}$ ×
- $t_{c \to q}$ ×

## • Updated condition

→ $t_{clock\ min} \geq T_{critical\ path} + t_{setup(dest)} + t_{c \to q\ (source)_{max}}$

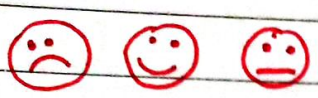
$t_{c \to q}$ : o
- min
- max

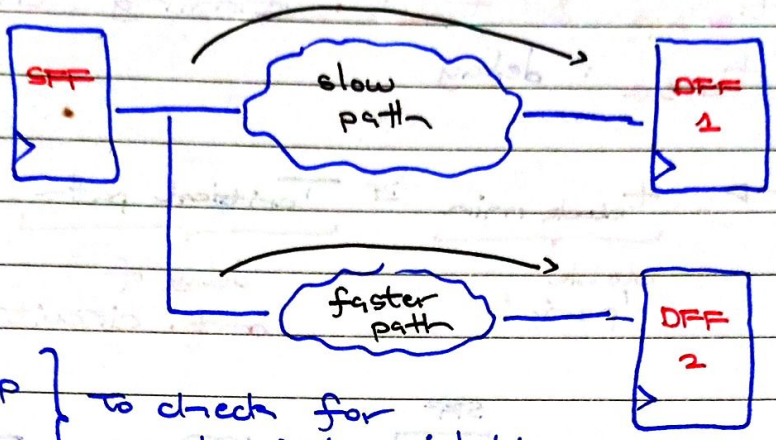
as
- 0 → 1
- 1 → 0

transitions do not take the same time ~

EXCELLENT

day/date


} my mixed feelings about choosing DSD as elective ; it do be like that




constraints
- $t_{setup}$
- $t_{hold}$

} to check for constraint violation, focus on DFF's.

max path timing constraint } if it's the actual max path, no violations can take place

$$t_{min} \geq T_{cp} + t_{c \to q\,(SFF)} + t_{setup\,(DFF)}$$

Example

$t_{min}$ : 5ns
$T_{cp}$ : 3ns
$t_{c \to q\,(max)}$ : 1ns

} $t_{setup\,(DFF)} = 1ns$

If combination of $T_{cp}$ and $t_{c \to q}$ is greater than $t_{setup}$ ; no setup violation occurs
- else wise system is metastable
  - $t_{setup}$ : max path constraint
  - $t_{hold}$ : min path constraint

» check setup violation on the max path } if setup ≠ violated ; hold ≠ violated as well

EXCELLENT

day/date

» faster paths have a higher odd of violating setup and hold constraints.

Example

— From slides ☺

$t_{min} \geq T_{cp} + t_{su} + t_{ca\,max}$

$t_{min} \geq 2.7\ ns$

Hold time violation: (no violation condition)

$t_h < T_{cp} + t_{ca\,min}$

shortest possible delay

## Fixing Violations

Setup Violation: Slow down CLK ; move registers around ~

Hold Violation: Add BUFFERS

Class Exercise : (Same timing as in previous example)

$t_{gate} = 1.2\ ns \quad t_{su} = 0.6\ ns \quad t_h = 0.4\ ns$

$t_{ca}$ : 0.8 ns → 1 ns

CP = $Q_0$ » $AND_1$ » $AND_2$ » $AND_3$ » $XOR_4$ » $Q_3$

$t_{min} \geq T_{cp} + t_{ca\,max} + t_{su}$

$t_{min} \geq 6.4\ ns$

$F_{max} = 156.25\ MHz$

No violation whatsoever

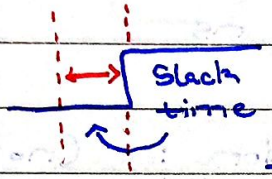
EXCELLENT

/date

## Hold Time Violation

$$t_h < T_{cp} + t_{comin}$$
$$t_h < 1.2 + 0.8$$
$$t_h < 2ns$$

Since, $t_h = 0.4ns$ ; no hold time violation




Time b/w settling and capture

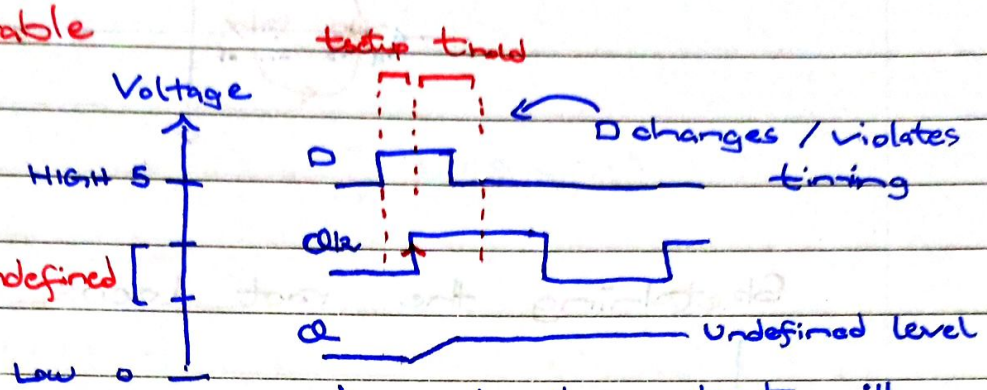
Thumbs up if slack time > 0

date ☺

13/04/23

# Digital System Design

## Metastable




↳ only the output will go meta

## Timing Violations

↳ Common Situations → Async Input Signals (↱ cause of metastability)

→ Slower convergence when moving near the metastable state.
Higher deviations when moving away from stable states.

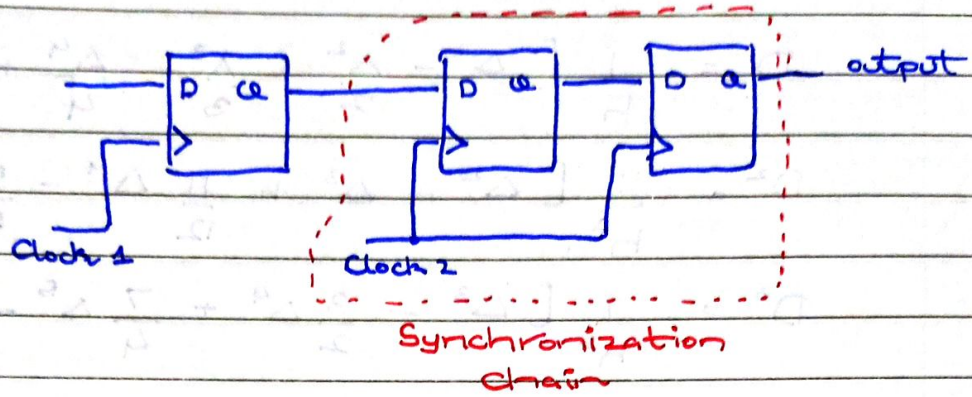
Meta stable level may resolve to 0 or 1.

→ If a meta level is output from the source ff, everything's well if value settles correctly before the capture edge.

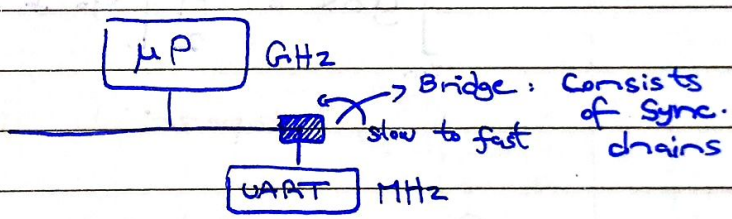
Slack time = Capture edge - Settling time

day/date

# Register Chain (Synchronization)




- Common technique to -
  - Add synchronizers to all async interfaces
  - Heuristically adopted approch to use two - three ff's as it increases MTBF exponentially




Addr : [n-1 : 0]
Data : [n-1 : 0]
Write : 1'b
Read : 1'b

→ Sync chains work fine for single bit signals

↳ Added complexity / Hardware inefficient to implement sync chain for every bit
↳ Use FIFO (first - in - first - out) ∴ in Lying

day/date

19/04/23

# Digital System Design

## Arithmetic Circuits

- Addition
- Subtraction
- Multiplication
- Division

→ Combinational Blocks
  - Results available in one clock cycle

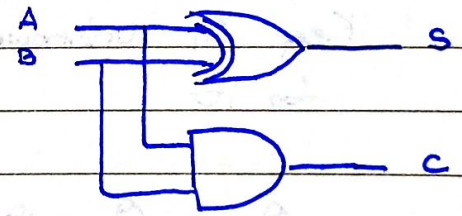
→ Multi-Cycle
  - Compute result over multiple clock cycles

→ 1-bit Half Adder

operands →

| A | B | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

op-width

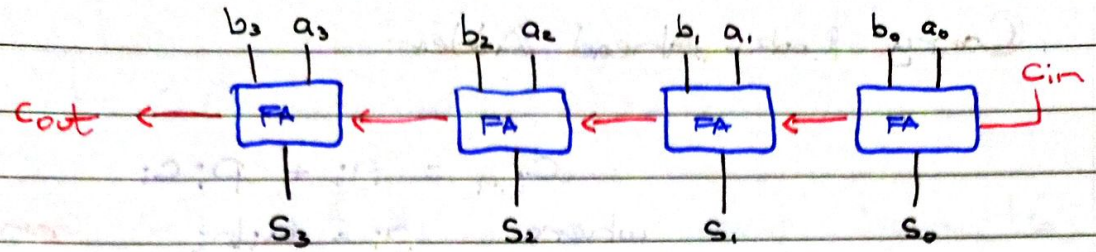



Does not accomodate for Cout of previous stage

→ Full Adder — Still 2-operands, however, with carry accomodated for

EXCELLENT

ay/date




Ripple

To perform A - B ; set $C_{in} = 1$ and negate b's

To perform both ; use XOR with b's and $C_{in}$ as control

→ Delay for N-bit Carry Propagate Adder

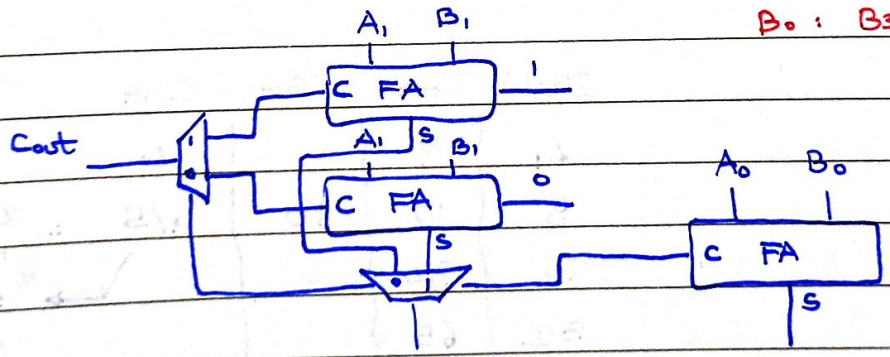
$2(N-1)\,t_{PD} + 3\,t_{PD}$

↳ proportional to N

Carry - Select Adder

$A_0 : A_{3-0}$ $A_1 : A_{7-4}$

$B_0 : B_{3-0}$ $B_1 : B_{7-4}$




Increased performance at the expense of Area

41% ↓ $t_{PD}$ Area ↑ 50 %.

EXCELLENT

/date

# Carry Look-Ahead Adder

$$c_{i+1} = g_i + p_i c_i$$

where $g_i = a_i b_i$ Generate

$p_i = (a_i + b_i)$ Propagate

- If $g_i$ is 1, $c_{i+1}$ is predicted to be 1
- If $g_i \neq 1$, and either A or B is one, $c_{i+1}$ is the same as previous $c_i$.

$g_i$ & $p_i$ are independent of carry terms only the inputs

{ All outputs ready after $4 t_{PD}$.
Delay remains same after even considering n-bit adder. }

| Bit W | CPA | CSA | CLA |
|---|---|---|---|
| 4 | 9 | | 4 |
| 8 | 17 | 10 | 4/5 |
| 16 | 33 | | |
| 32 | 65 | | |
| 64 | 129 | | |

∴ 4 if assuming no fan in problem
→ 5 if arranging CLA as CSA

## Multiplication

→ Multiplication is an AND operation.

date

N bit x N bit
↳ 2N output bits

P = A * B ⇝ P must have twice the bits of A and B

→ Dedicated Hardware Mapping: {Hard Logic}
↳ Default Fmax = 1 GHz

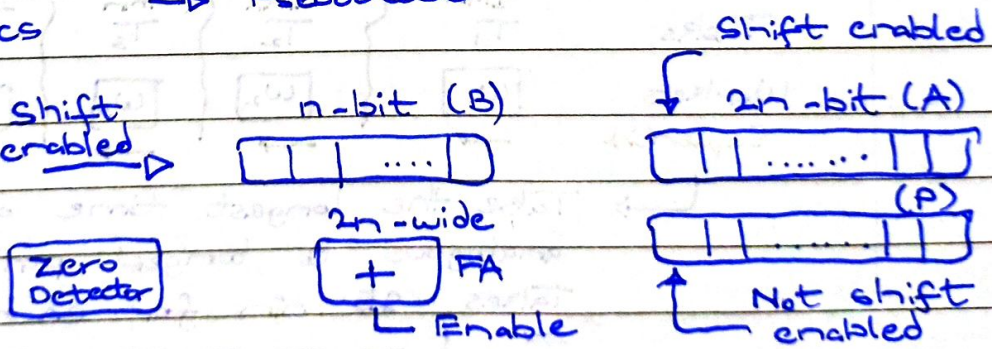
{ DSP Multiplier : Fmax = 250 MHz }

Combinational implementation's relation with Fmax ; ~ in Lecture

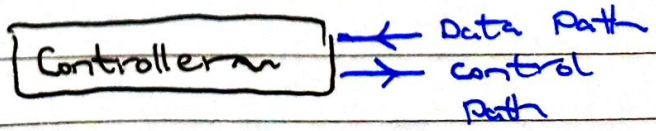
Serial Multiplier (Multi - Cycle)

Problem Statement → Architecture Block Diagram → μ - Architecture White Blocks

C/C++ / Python / MATLAB } Executable specs → Pseudocode




Registers also need load { P=0 ; A = ~ B = ~ }




EXCELLENT

/date

Data path : { z : zero detect ;
$b_0$ : B at $0^{th}$ index }

Control path : { shift A , load A , shift B
load B , load P , sel P }

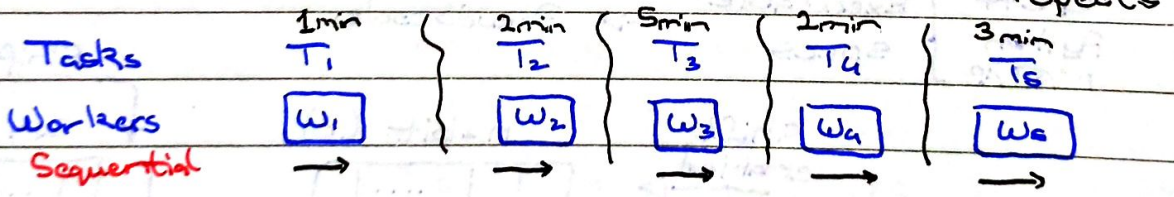
Architecture ⟷ μ - Architecture
↕ Analogy ↕
ISA ⟷ μ - Processor

~ Fixed Point
~ Floating Point } Arithmetic (skippo? or just general idea)

~ μP vs. FPGA

parallelism

needs task to be parallelized be same / repeats




↳ Take the longest time of workers ;
analogous to longest path in FFs
Takes 25 min for each big task
$\overline{T} = \{ T_1, T_2, T_3, T_4, T_5 \}$

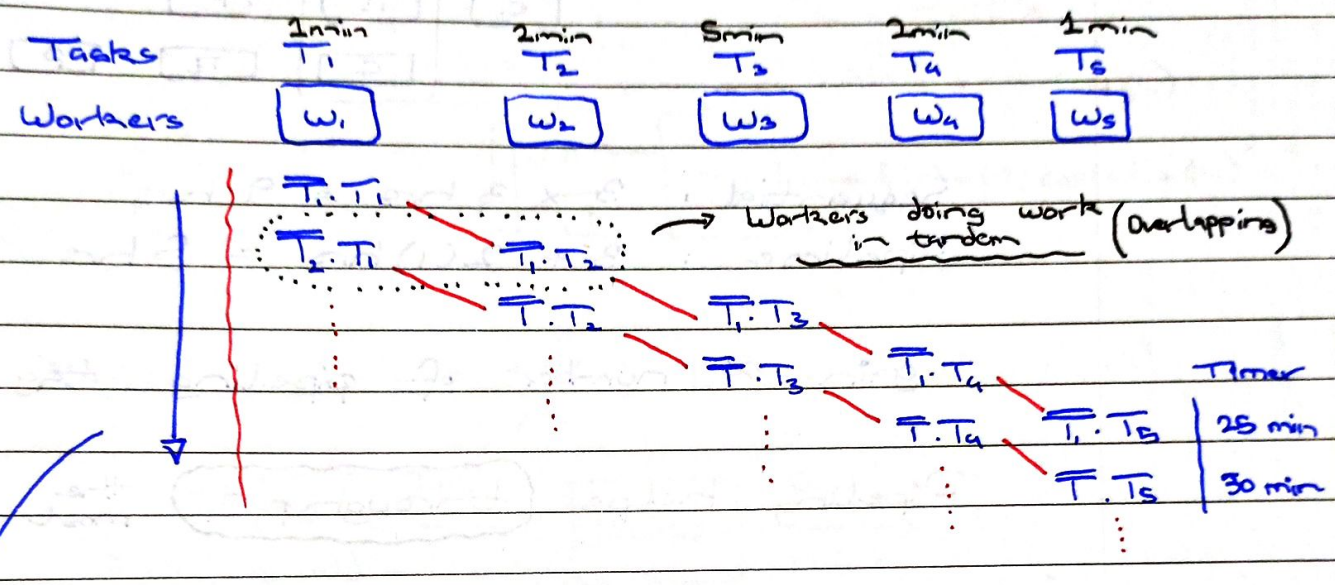
Continued on next DSD page

date

17/05/23

# Digital System Design

## Continued




→ Workers doing work in tandem (Overlapping)

$$\overline{T}_{0,1,\ldots 73} = \{T_1, T_2, T_3, T_4, T_5\}_{0,1,2,\ldots 73}$$

Sequential Time Taken = 25 min × 73

Pipelined " " = 5 min × 72 + 25 min

Latency: Time taken ↑ Latency calculations

Throughput: Amount of work done per unit time

one paper being marked every [5 min] } after pipeline is full
i.e. ignore initial task

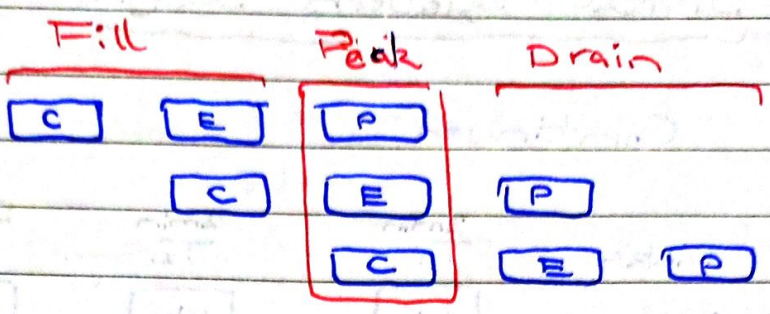
for individual tasks $\overline{T}$ [25 min]
total Latency [25 + 5 × 72]

└ overall pipelining gain is ≈ to the number of stages

EXCELLENT

date

Example




Sequential : $3 \times 3$ hrs $= 9$ hrs

Pipeline : $3 + 2(1)$ hrs $= 5$ hrs

gain $\approx$ number of pipeling stages

Pipeling helps throughput the most