CE1005 Digital Logic
Revision Notes

Date No.

Quantisation

→ Range of analog values lumped together and assigned a representative digital value

Binary represented in Voltage *Memory Work

	0	1
TTL (Transistor-Transistor Logic)	0 – 0.8V	2 – 5V
CMOS (Complementary metal-oxide semiconductor)	0 – 1.5V	3.5 – 5V

Series vs. Parallel

Simplicity/Cost vs Speed

Base N to Base 10 Conversion

→ $ab.c_N = (a \times N^1) + (b \times N^0) + (c \times N^{-1})_{10}$

$10.1_2 = (1 \times 2^1) + (1 \times 2^{-1})$
$= 2.5_{10}$

Base 10 to Base N Conversion

→ (Base 10 Num) / N until 0
→ Write down the remainder
→ LSD = First Remainder, MSD = Last Remainder

LSB → 1 | 2 | 5_{10}
0 | 2 | 2_{10}
MSB → 1 | 2 | 1_{10}
0

$5_{10} = 101_2$

Binary Coded Decimal

0 → 0000 ↳ Leftover are illegal
1 → 0001
⋮
8 → 1000
9 → 1001

Excess-3

0 → 0011
1 → 0100
⋮
8 → 1011
9 → 1100

} +3 from BCD

Gray Code

→ Solves the issue where BCD at times changes 2 bits
↳ Gray code only allows 1 change

Dec	Gray Code			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

Binary to Gray Code

1101 → 1011 (13)

} XOR

1, 1 } XOR → 1, 0
0 } XOR → 1
1 } XOR → 1

Parity Bit

→ Even Parity [1] 01101 , [0] 11000 Parity makes it even 1s
(1: 4, 0: 2)

→ Odd Parity [0] 01101 [1] 11000 Parity makes it odd 1s
(0: 3, 1: 3)

* Transmiter & Reciever must agree on parity system } Drawbacks
* Error checking only for ~~en~~ 1 bit.

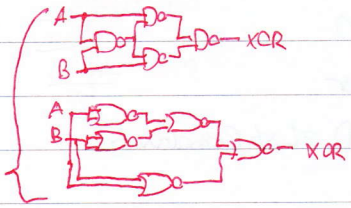
Truth Tables

0 0 | 1
0 : | 0
: : | :
1 1 | :

} 2^N rows → Number of inputs

Logic Gates * Remember Propagation Delays

→ OR → XNOR
→ AND
→ NOT
→ BUFFER
→ NOR
→ NAND
→ XOR



Absorption Laws * Memory

→ $A + AB = A$ → $A(1+B) = A$

→ $A + A'B = A + B$ → using ① $(A+AB) + A'B = A + B(A+A') = A + B$

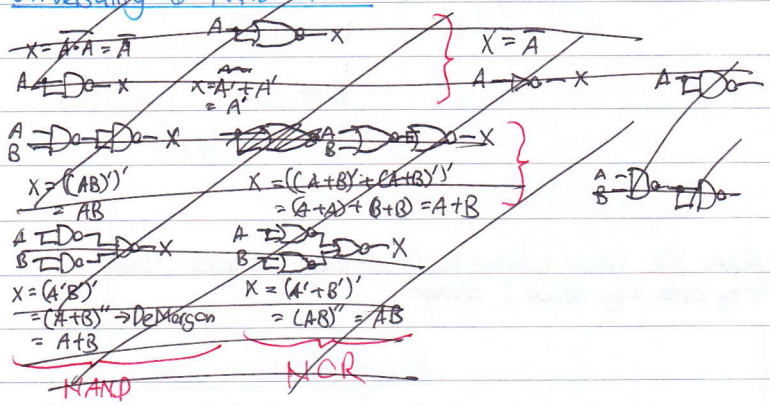
Consensus * Memory

→ $AB + A'C + BC = AB + A'C$

↳ $(BC) = A(BC)(A+A')$ ①
$= BEABC + A'BC$
$AB + A'C + BC = AB + A'C + CABC + A'BC)$
$= AB + (ABC) + A'C + (A'BC)$
$= AB(1+C) + A'C(1+B)$
$= AB + A'C$

Universality of NAND & NOR

~~Universality of NAND & NOR~~



NAND NOR

$X = \overline{A \cdot A}$ $X = \overline{A + A}$ → NOT

$X = \overline{A \cdot B}$ $X = \overline{A + B}$ → AND

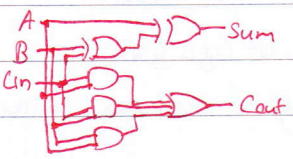
$X = \overline{\overline{A} \cdot \overline{B}}$ $X = \overline{\overline{A} + \overline{B}}$ → OR

Date No.

Adders

→ Half Adder Carry = A·B, Sum = A ⊕ B

→ Full Adder Cout = A·B + B·Cin + A·Cin, Sum = A ⊕ B ⊕ Cin



Full Adder with 2 Half Adder

S1 = A ⊕ B S2 = (A ⊕ B) ⊕ Cin → Sum

C1 = A·B C2 = Cin·(A ⊕ B)

Cout = A·B + Cin·(A ⊕ B)

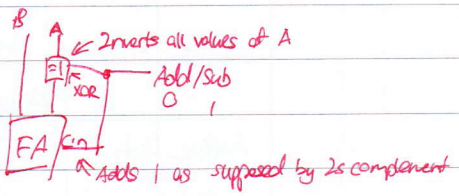
2's complement *Used so that sub can be used as addition, on the same hardware

→ Invert every bit

→ Add 1 to it

7_{10} = 0111 (sign) $_2$

$-7_{10} = 1000 + 1 = 1001_2$



2's Complement Range

→ (-2^n) to $(2^n - 1)$ * n is without counting sign bit

Detect Overflow

→ Addition

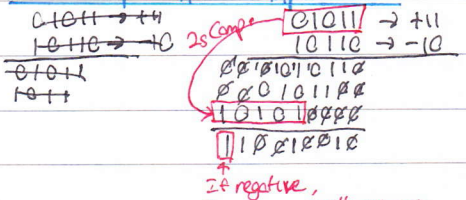
↳ No overflow when operands have different signs (+ −)

↳ Overflow when operands have same sign & resulting sign has an opposing sign.

→ Subtraction

↳ No overflow when operands have same signs (+ + / − −)

2's Complement Multiplication



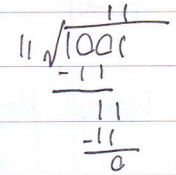
2's Complement Division

→ Convert to unsigned

→ Divide

→ Add the Appropriate sign

9/3 = 3



BCD Addition

→ Normal addition

→ When addition exceeds 9, a correction is required by adding 6

Combinational Logic Circuits

→ Combination of Logic gates

→ No memory characteristics, unlike sequential circuits

Forms of Boolean Expression

Canonical → active high output Standard

Sum of minterms expr (SOm) = Sum of products expression (SOP) → Cares for 1s

~~Sum of minterms expr (SOm)~~ ↳ $\Sigma_{xyz}(1,3,4,7)$ ↳ Factored form

Product of maxterms expr (POM) = Product of Sums expression (POS) → Cares for 0s

↳ active low output ↳ $\Pi_{xyz}(2,5,6)$

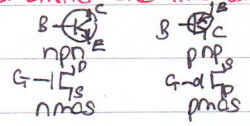
Active High vs Active Low

	Asserted	Not Asserted
Active High	1	0
Active Low	0	1

Transistors * Source and Emitter are like direct connections to Live & Gnd

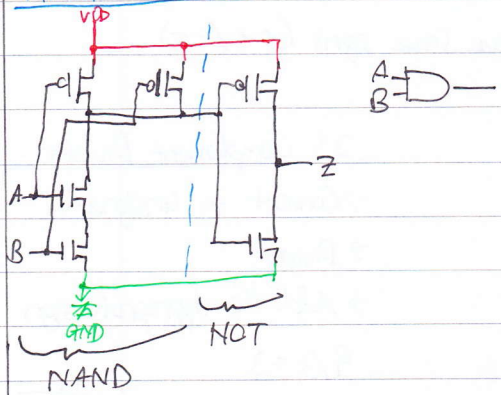
BJT Base Emitter Collector

MOSFET Gate Source Drain



↳ MOS Field Effect Transistors

NAND with Transistors



Voltage Parameters

	Min	Max
Input	V_{IH}	V_{IL}
Output	V_{OH}	V_{OL}

Min Voltage recognized as 1

Max Voltage recognized as 0

Noise Margin

For Logic 0

Low State DC Noise Margin = $V_{IL} - V_{OL}$ [Max]

High State DC Noise Margin = $V_{OH} - V_{IH}$ [Min]

For Logic 1

Date No.

Tristate Outputs

→ 3 output states, Logic 0, 1 or High Z (High Impedance)

→ Tristate Inverter, Tristate Buffer * Have an enable input

→ Used when 2 or more outputs connected together

↳ but only one can be activated at a ~~time~~ time

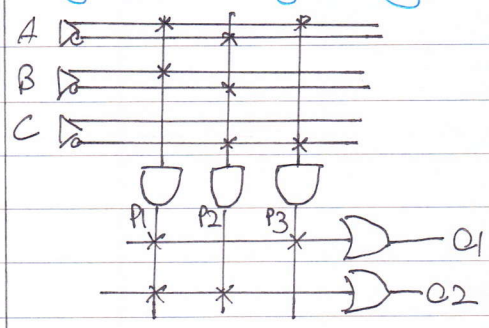
Schmitt-Trigger ~~Inputs~~ Inverter

→ V_{IN} rises above $VT+$, V_{out} = LOW

→ V_{IN} drops below $VT-$, V_{out} = HIGH

→ Provides clean transition

Programmable Logic Array



$P1 = AB$

$P2 = A'B'C'$

$P3 = AC'$

$O1 = AB + AC'$

$O2 = A'BC' + AB$

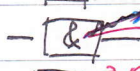
Floating point numbers

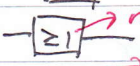
(Floating number) $= S \times R^{E}$ ← Exponent

↑ Significand ↑ Radix

IEEE Logic Symbols

→ NOT  ← repr bubble

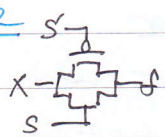
→ AND  → All inputs

→ OR  → more than 1 input

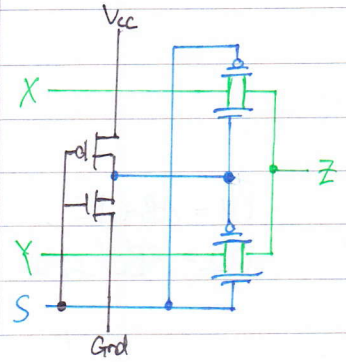
→ XOR  → only 1 input

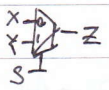
CMOS Transmission gate

→ Used for multiplexors



S	f
0	z
1	x





Combinational Design Process

→ Capture the function → Truth tables or eqns

→ Convert to Circuit → creating circuit for each output

7 Segment Decoder

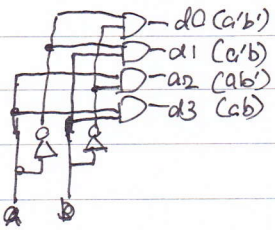
→ Decode 4 bits to 7 bits

Decoder

→ N inputs → 2^N outputs

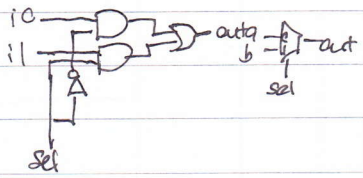
→ One-hot output

* 00 → 0001 it is not 0



Multiplexor

→ Using decoder to select which AND gate to use



Date No.

Verilog Module Declaration → Structural Verilog

→ module <name> (input port1, port2, output port3); ← comma

Input can still be added

→ endmodule ← No semicolon!

Rmb! Statements has semicolon

Gate level primitives

→ and (<output port>, <in1>, <in2>, ... <in-n>);

↳ same for ~~nand~~ or, nor, nand, xor

→ Ordering is not important

Wire

→ Wire <var_name>, <v2>, <v3>;

↳ can be bus too

Calling another module

→ <module name> <module var.> (.<mod param>(<var mod param>));

eg. add_half M1 (.a(A)); or add_half M1 (A);

~~Ordered~~ Named Indentation Named connection. * can be ignored Ordered Indentation

Verilog Assignments → Supports basic arithmetic + - * /

→ assign y = a & b (and (y, a, b)) → gate level primitive

↳ Continuous assignment

assign y = sel ? x1 : x0;

↳ Conditional assignment

Vectors

→ multi-bit signals

eg. wire [7:0] databus

↳ 8-bit signal

→ Assign a range

eg. assign z = some[4:3]

assign x[2:1] = some[4:3]

Number Literals

→ <size>'<radix><value>

inbits b=binary h=hex o=octal d=dec the number

Concatenation

→ combine 2 buses into 1

~~→ cat~~ eg. assign y = {a, b3};

→ can use in assign as well

assign {cout, sum3} = A + B + Cin

Replication

→ Repeat an element

eg. assign y = {{4{a}}, b3}

a is replicated 4 times

Parameters

→ Constant value local to module → can have multiple

eg. module some_mod #(parameter SIZE=8)(

input [SIZE-1:0] X, Y,

output [SIZE-1:0] Z);

→ Used to redefine other submodules

eg. submod #(.SIZE(SIZE)) u1 (.X(a));

parameter before variable name

Combinational Always Block → Behavioral Design

```
always @(a,b)
begin
  y = a & b;
end
```

↳ Use always@* instead

Sensitivity list: inputs that trigger this always block

Output

* Signals in always block must be reg, not wire
  - ↳ like variable
  - ↳ like connection

* Input & output must contain reg as well
* Never assign the same var in other always blocks or assign statements

Considerations

→
```
y = a & b;
y = x | c;   ← will be ignored.
y = x ^ c;
```

→ Avoid latches by using default assignments

Case Statement → Implement decoders, multiplexers

```
case(sel)
  2'b01 : y = a;
  2'b10 : begin
            y = b;
            z = c;
          end
  2'b11 : y = c;
  default : 4'bxxxx;
endcase
```

No semicolon

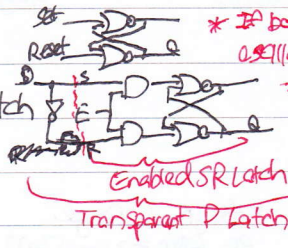
→ No break statements

Sequential Circuits

* Rmb Propagation Delay
→ Brings states (memory)

→ Set-Reset (SR) Latch

→ Gated / Enabled SR Latch



* If both asserted, oscillation may occur
→ Prevents S&R from occurring at the same time

SR Latch

S	R	Q+	Func
0	0	Q	Store
0	1	0	Rst
1	0	1	Set
1	1	?	Und

Enabled SR Latch

E	S	R	Q+	Func
0	X	X	Q	Store
1	0	0	Q	Same as SR
1	0	1	0	
1	1	0	1	
1	1	1	?	

Trans D Latch

E	D	Q+	Func
0	X	Q	Store
1	0	0	Set
1	1	1	Set

→ Transparent D Latch

Active Low Latches

→ Swap NOR for NAND & OR for AND

Limitations of Latches

~~→ Clk must be asserted for the whole propagation delay.~~

→ Output may change during the whole period when clk is enabled

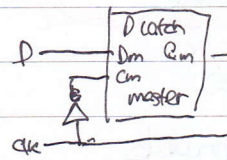
Clocks

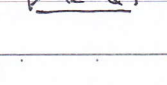
→ Period = seconds
- → milli 10^{-3}
- → micro 10^{-6}
- → nano 10^{-9}

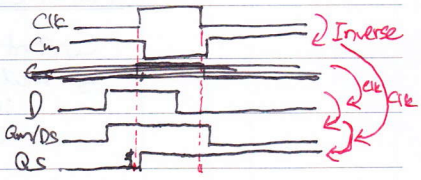
→ Frequency = Hz = $\frac{1}{\text{Period}}$

Edge Triggered Flip Flops

→ Combination of 2 D Latches





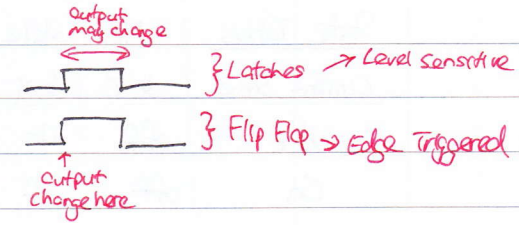


Date No.

Latches vs. Flip Flops

→ Output changes when (control) enable input is high

→ Output changes only at control input edges

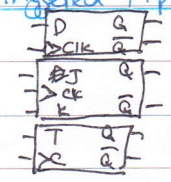


Forms of Edged Triggered Flip Flops

→ D Flip Flop

→ J-K Flip Flop

→ T Flip Flop



D Flip Flop

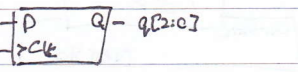
D	Clk	Q+
0	Pos	0
1	Pos	1

JK Flip Flop

J	K	Clk	Q+
0	0	Pos	no change
0	1	Pos	0
1	0	Pos	1
1	1	Pos	toggle

Registers

→ Multi bit version of Flip Flop



Registers in Verilog

always @ (negedge / posedge clk) [or posedge rst]

→ falling edge → trigger an asynchronous reset ↳ reset instantly

synchronous → reset only on clk edge

↳ Always block triggered when clock is positive edge

q <= d;

↳ Non-blocking assignment

Assignment Operator

→ (=) ⇒ Blocking Assignment ⇒ Order Matters

→ (<=) ⇒ Non-blocking assignment ⇒ Order don't matter

Applications of synchronous circuits (Sequential Verilog)

→ Counters

→ Shift Registers → single input passed through chain of flip flops

→ Serial-to-parallel & vice-versa

→ Memories

↳ Contains a (encode) decoder w. enable, (read) multiplexer, (write) data-wire

↳ Similar to shift registers

* Loading value to be shifted out

Standard

```
q1 <= y;
q2 <= q1;
q3 <= q2;
q <= q3;
```

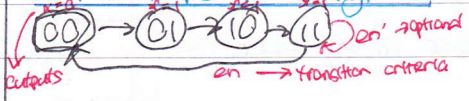
Vectors

```
q[1] <= y;
q[3:1] <= q[2:0];
q_out <= q[3];
```

→ assign q_out = rst ? q[3] : q[0]

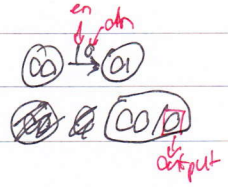
↳ with capability of switching

State Transition Diagram



→ Use legends → Inputs: en, dn eg.

↳ Outputs: x eg.



State Tables

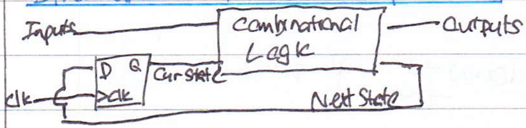
Current State	Next State b=0	Next State b=1	Output
off	off	on	0
on	off	off	1

Moore & Mealy Machines

→ Moore: Output dependant on State

→ Mealy: Output dependant on State & inputs
↳ outputs can change while in the same state

Structure of Finite State Machines



State Translation Table for Combinational Logic

→ Encode the states. eg. off=00

	Inputs s1	Inputs s0	Inputs b	Outputs x	Outputs n1	Outputs n0
off	0	0	0	0	0	0
	0	0	1	0	0	1
on	0	1	0	1	0	0
	0	1	1	1	0	0

Translate Outputs to ~~Combinational Logic~~ Logic Equations

→ Use Sum of minterms expression (add all the ones)

→ $x = s0$ → $n0 = s1's0'b$

→ $n1 = 0$

Implementing in Verilog

→ 3 things, Logic to drive output, State Register & Logic to determine next state

→ Assign statements for Logic Equations

→ always block for rst or transit to next state

Complex FSMs

→ Use parameter keyword.
↳ eg. parameter st1 = 2'b00, st2 = 2'b01;

→ Use case statements with another always block

```
nst = st;              ← Default Assignment
case (st)
  wait: if (s) nst = start;
  start: if (a)                          (cur states)
     if (r & ~s & ~g) nst = recall;
     else nst = wait;                    } transit states
  :
  default: nst = wait;                   ↳ Default case
endcase
```