EIE2211 Logic Design Cluesheet

bin/hex→dec

23.375

2|23 1 | 2|.375 0
2|11 1 | 2|.75 1
2|5 1 | 2|1.5 1
2|2 0 |
1 1

Unsigned int

Signed int ~ magnitude
- ~1's complement: FF−num or num
- ~2's complement: 100−num or 1's+1

dec digit 0~9
- 8,4,2,1 (BCD): common
- Excess 3: 8,4,2,1 (+3)
- 8,4,−2,−1: 2,1 均為0→common 否則 8,4加0, 2,−1 取相反加
- Gray: = common ⊕ common ≫1

ASCII (7 bits) 94 printable (32~125) 34 non-p~ (control) (0~31 & 126,127)

48 ... 0 1 2 3 4 ... 9 ... @ A B C ... J ... T ... Y Z ... ` a b c ...

57 64 65 75 85 90 96 97

Unicode→65536

Boolean Algebra

- A+(BC)=(A+B)(A+C) distributive law
- AB+ĀC+BC=AB+ĀC } consensus law
- (A+B)(Ā+C)(B+C)=(A+B)(Ā+C)

Minterm XYZ: X̄=0, X=1

Maxterm X+Y+Z: X=1, X̄=0

F(A,B,...) = { Σm( ) / ΠM( )

cost
1. literal cost
2. Gate input cost
   - G
   - GN

K-map

- prime implicant: maximum possible
- essential prime implicant: the only that contain a term

Base: Hex: For 0→X To F/1F/3F/7F not:(1−x)
Step 1/2/4: X=8 and 1→A: X+4 and 1→B: D+2C+4B+8A
X=2 and 1→C: X and 1→D: exp ▲Next Z+2Y+4X

XOR = XNAND = AB̄+ĀB

XAND = XNOR = AB+ĀB̄

Buffer → { electronic Amplifier / increase the speed

$A_1 \oplus A_2 \odot \dots \odot A_{2n+1} = A_1 \oplus A_2 \oplus \dots \oplus A_{2n+1}$

$A_1 \odot A_2 \odot \dots \odot A_{2n} = A_1 \oplus A_2 \oplus \dots \oplus A_{2n}$

Hi-Z (≈ disconnected)
- 3state buffer X → Y, EN
- transmission gate
- any gate
- have 1, 0, Hi-Z
- can be connected with other input
- can transmit signal on 2 directions.

Functions/Functional Blocks: those useful in design { SSI circuits / MSI ~ / LSI ~ } parts in VISI circuits nowadays

Decoders, Encoders, Mux(selecting), Demux

74138 ↓

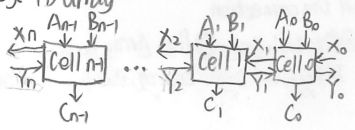
$\overline{EN_1}, \overline{EN_2}, \overline{EN_3}$ = { 100: $Y_{CBA}=0$, other $Y_s=1$ / else: all $Y_s=1$ }

Iterative combinational circuits

cell—subfunction block

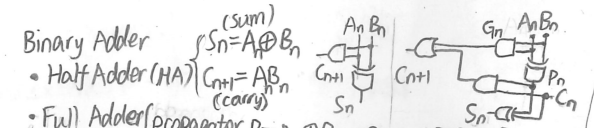
Iterative array—array of interconnected cells

eg. 1D array



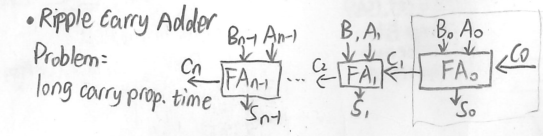
Binary Adder

- Half Adder (HA): $S_n = A_n \oplus B_n$ (sum); $C_{n+1} = A_n B_n$ (carry)
- Full Adder (FA): propagator $P_n = A_n \oplus B_n$; generator $G_n = A_n B_n$; $S_n = A_n \oplus B_n \oplus C_n = P_n \oplus C_n$; $C_{n+1} = G_n + P_n C_n$



- Ripple carry Adder

Problem: long carry prop. time



- Carry Lookahead Adder (CLA):

$C_{n+1} = G_n + P_n C_n$ need to wait

$= G_n + P_n(G_{n-1} + P_{n-1} C_{n-1})$

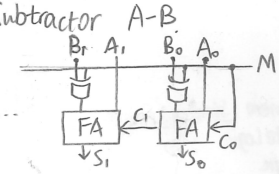
Sol. $= G_n + P_n(G_{n-1} + P_{n-1}(G_{n-2} + P_{n-2} C_{n-2}))$

$= \sum_{i=0}^{n} \left[G_i \left(\prod_{j=i+1}^{n} (P_j) \right) \right] + C_0 \prod_{i=0}^{n} (P_i)$

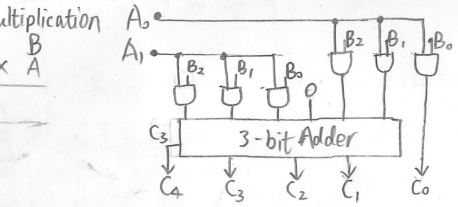
Overflow $OF = (A_{n-1} \odot B_{n-1})(C_n \oplus C_{n-1})$

same sign? sign bit changed?

Subtractor A−B



Multiplication



- Incrementing
- Decrementing
- Multiply by const
- Div by const
- Zero fill and exten

Sequential circuits = combinational + storage elements

- Synchronous: Discrete
- Asynchronous: Continuous

SR-latch
- nor: S, R, Q, Q̄; normal=0 input=1; S=R=1: Q=Q̄=0 then S=R=0: unstable
- nand: S, R, Q, Q̄; normal=1 input=0; S=R=0: Q=Q̄=1 then S=R=1: unstable

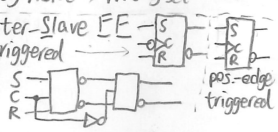
D latch (transparent ~) Trigger: change in value on input ⇒ latch switches (as long as CLK=1)

Latch-timing problem: CLK, Q

1's catching problem: input (S/R) influence by noice ⇒ wrong set

Sol. Flip-flop: ① S-R Master-Slave FF (neg. edge triggered) → pos-edge triggered

0 = neg/pos ▷ = edge/level



D flip-flop (= master-slave D FF) (neg. edge)

Direct input: FF with set/reset (At powerup/reset, asynchronously) eg.

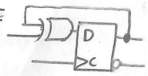
J-K FF

$D = J\bar{Q} + \bar{K}Q$

J=1 → set K=1 → reset

J=K=0 → keep J=K=1 → Toggle

T FF



For a seq. circuit

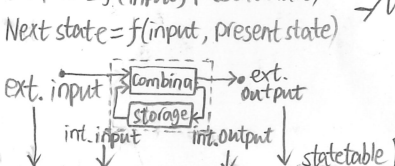
Output = f(input, present state)

Next state = f(input, present state)

⇒ Common logic design (K-map & D-FF)

⇒ unused state ⇒ don't care

practical (eg. powerup, noice): no dead loop



statetable

X,Y in f_z, f_w = mealy model (state) input/output (no output ⇒ no'/') (state/output) (input)

X,Y not in f_z, f_w = moore model

Application = Sequence recognizer, register, counter

- Ripple counter → CLK → D Q Q̄ → CLK of next bit; problem: slow to prop.
- Syn. counter: $A_n(t+1) = A_n \oplus (\Pi_{i=0}^{n-1}(A_i))$ or $T_n = \prod_{i=0}^{n-1}(A_i)$
- Down cntr (syn: $A_n(t+1) = A_n \oplus (\Pi_{i=0}^{n-1}(\bar{A}_i))$ or $T_n = \prod_{i=0}^{n-1}(\bar{A}_i)$
- Up/down cntr

for D-FF for T-FF or J-K FF

- Parallel load cntr: $A_n = I_n$ (load init value) when LD=1
- Cntr with arbitary seq.: common counter + combinational logic or redesign seq.

Hardware Description Language = Programmable Language
Parallel Execution = Descriptive
Very High Speed IC HDL = General Purpose, more clearly
Verilog: efficient simulation, much similar to C.

VHDL Code Structure
Library & Define symbol:
```
library IEEE, ICDF_VHDL;
use IEEE.STD_LOGIC_1164.ALL;
use ...;...
```
Entity: Define model
```
Entity <entityname> is
  Port(
    <portname>: in/out <type>;
    ...;
  );
End <entityname>;
```
Architecture: structure, behavior, dataflow
```
Architecture <funcname> of <entityname> is
Signal <varname> : <type>;
Signal ...;
Begin
...
End <funcname>;
```

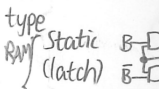
<type> { std_logic '1','0','Z','X'(未知), 'U'(未初始化) ...(in total 9)
std_logic_vector(_ downto _)

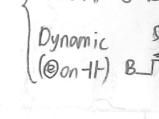
Sequential (Process) CLK'event (clock edge)
```
Process (sensitivity list / CLK)
Begin
...
End Process;
```
<var> <= ...; Must be signal

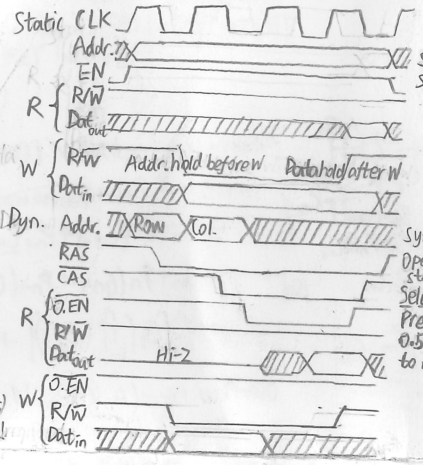
<= Assign/connect + -
vector(:) <= vector(:), vector <= 'val'((:) when part)
not ___, ___and___, ___nand___, ___or___, ___nor___, ___xor___, ___xnor___
conditional value (combinational): ___when___else___

conditional Behavior
① if (___) then ...; ...; elsif (___) then ...; ...; else ...; ...; end if;
② case ___ is
when <value> => ...;
when <value> => ...;
when others => ...;
end case;
only sen_list item or CLK'event

Memory: collection of storage cells { RAM { volatile; ROM { non-volatile
Address = identifier
Data { bit; byte: 8b = 1B; word: typical unit 1B, 2B, 4B...
operations
read: Place addr, wait for stable data
write: Place addr & data, toggle write control line

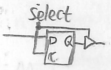
type RAM
Static (latch)


Dynamic (©on-it)


Static CLK


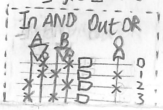
Open a row (strobe) store to buffer
Select a col. (CAS)
Precharge B to 0.5, sense amp to read/refresh

N_word × N_bit_len RAM
DATA
ADRS
CS (chip select)
R/W

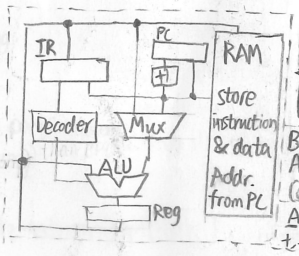
DRAM Cell


- Syn DRAM: syn registers on Addr in, Data in, Data O
  Col addr cntr: [colAddr, colAddr + burst − 1]
- Double Data Rate SDRAM: Transfer on both CLK edge.
- RAMBUS DRAM: packet based bus (syn, transfer on both edge)
  ↳ 3b row addr, 5b col addr, 16/18b data
  4 CLK/Pkt × 2 Trasfr/CLK 12b 20b 64b/72b ECC

Reprogrammable: I/O control the connection
Programmable ROM Mem ① Prog Logic Array ② Prog Array Logic
Logic Device In fuse, Out Prog In Prog, Out Prog In Prog, Out fuse, with Fdbk



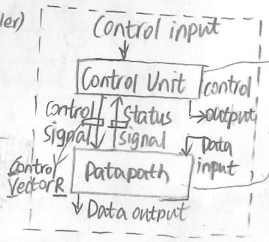
Digital Logic Sys
↓process control & data processing
Processing Unit
↓fabricated in a single IC
Microprocessor cheap, low-power than CPU
↓with mem, IO in one chip
Microcontroller (simple computer)



Register { Instru. R; Prog Cntr
Bus: Addr, Data, Control
Arith Logic U: +, −, ×, ÷, logic

Control Unit (controller)
direct instru & data flow
IR → Decoder → control signal

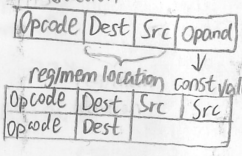
Data path
Data transfer & processing operation



Statemem Status Register
Nxt state Logic
Out Logic

Fetch, Decode, Execute each stage ≥ 1 CLK

Instru Set Archi { instruction set; word size; mem addr mode; addr & data format
Reduced IS Computing: simp, basic op, IS; fast, reliable, more space
Complex IS C: complx IS; multi op in one instru; slow run, fast coding

Instruction


Addr mode
- M0 = 0: Direct Addr
  LOAD | Dest | M0 | x | ADDR = 36e
- M0 = 1: Relative Addr
  LOAD | Dest | M0 | ±off
- M0 Register Indirect
  LOAD | Dest | M0 | Rsrc

Status Bit (Zero, Negative)
ADDR(I)
SUBR(I)
AND/OR
SLR

NOP		No operation
LOAD	RD	RD ← M[M[PC+1]] (M0=0)
	RD off	RD ← M[PC+off] (M0=1)
STORE	RD	M[M[PC+1]] ← RD
	RD off	M[PC+off] ← RD
ADDR	RD, RS1, RS2	RD ← RS1 + RS2 (Z,N)
ADDI	RD, RS1, IMM	RD ← RS1 + IMM (Z,N)
SUBR	RD, RS1, RS2	RD ← RS1 − RS2 (Z,N)
SUBI	RD, RS1, IMM	RD ← RS − 1 (Z,N)
AND	RD, RS1, RS2	RD ← RS1 AND RS2 (Z,N)
OR	RD, RS1, RS2	RD ← RS1 OR RS2 (Z,N)
SLR	RD, RS, Mode	RD ← RS ≪ₐ (≫)₁ 1 (Z,N)
JZ	off	IF Z=1, PC = PC+off
JN	off	IF N=1, PC = PC+off
JRZ	RD off	IF RD=0, PC = PC+off
JUMP	off	PC = PC+off off=0 ⇒ stop