コンピュータシステムの理論と実装

2025/2/23

1章 ハードウェア実装

最初にNANDゲートのみが与えられる。
NANDゲートから各ゲートを実装していく。

NAND (Not And)

a	b	out
0	0	1
1	0	1
0	1	1
1	1	0

√Not

a	out
0	1
1	0
0	1
1	0

√Or

a	b	out
0	0	0
1	0	1
0	1	1
1	1	1

√And

a	b	out
0	0	0
1	0	0
0	1	0
1	1	1

✓ Xor (And, Or, Not)

a	b	out
0	0	0
1	0	1
0	1	1
1	1	0

a,bが異なる時にビットが立つ
a,bが同じ時にビットは立たない。

Xor.hdlファイルのif文を見て絶句してしまったので表からif文作ってhdl文作成したほうが勉強になりそう

Mux (マルチプレクサ)

sel	out
0	a
1	b

if sel==0 out=a
else out=b

aOrSel =out
bAndsel =out

(a Or sel)	or	(b And sel)
0	0	0
0	0	0
1	0	0
1	0	0

Mux

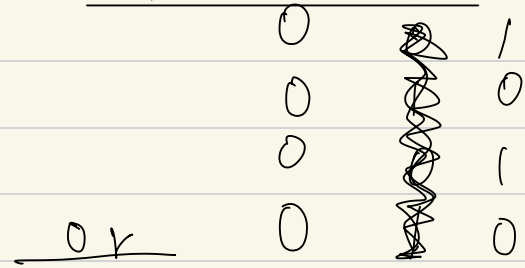
(a Or sel) Or (b And sel)

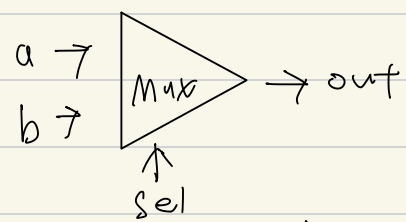
Not

a	sel	a Or sel
0	0	0
0	1	1
1	0	1
1	1	1

b	sel	b And sel
0	0	0
0	1	1
1	0	0
1	1	0

a Or sel	b And sel	Or
0	0	0
0	1	1
1	0	1
1	1	1





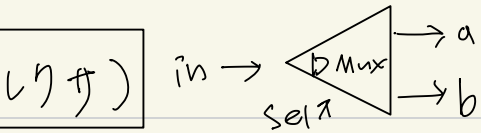
sel	out
0	a
1	b

Xor

a Or sel	Not b And sel	Xor
0	0	0
0	1	1
1	0	1
1	1	0

(a Or sel) Xor (Not b And sel)

DMux（デマルチプレクサ）



sel	a	b
0	in	0
1	0	in

入力　in, sel

出力　a, b

if (sel == 0)
  {a, b} = {in, 0}
else
  {a, b} = {0, in}

in	sel	a
0	0	0
1	0	1
0	1	0
1	1	0

Xor	And	or	NAnd
0	0	0	1
1	0	1	1
1	0	1	1
0	1	1	0

in	Sel	b
0	0	0
1	0	0
0	1	0
1	1	1

outb = And(a=in, b=sel, out=b);

outa = Xor(a=in, b=outb, out=a);

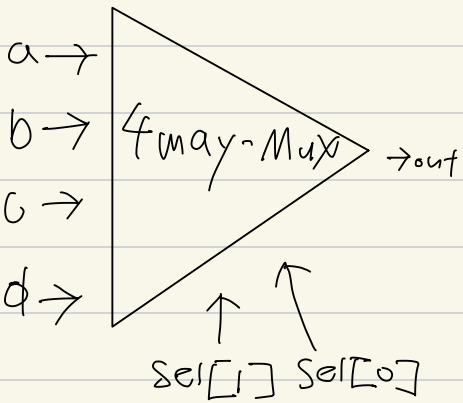
HDL:

```
And(a=in, b=sel, out=outb, out=b);
Xor(a=in, b=outb, out=a);
```

P.388 参照

Mux4Way16

sel[1]	sel[0]	out
0	0	a
0	1	b
1	0	c
1	1	d

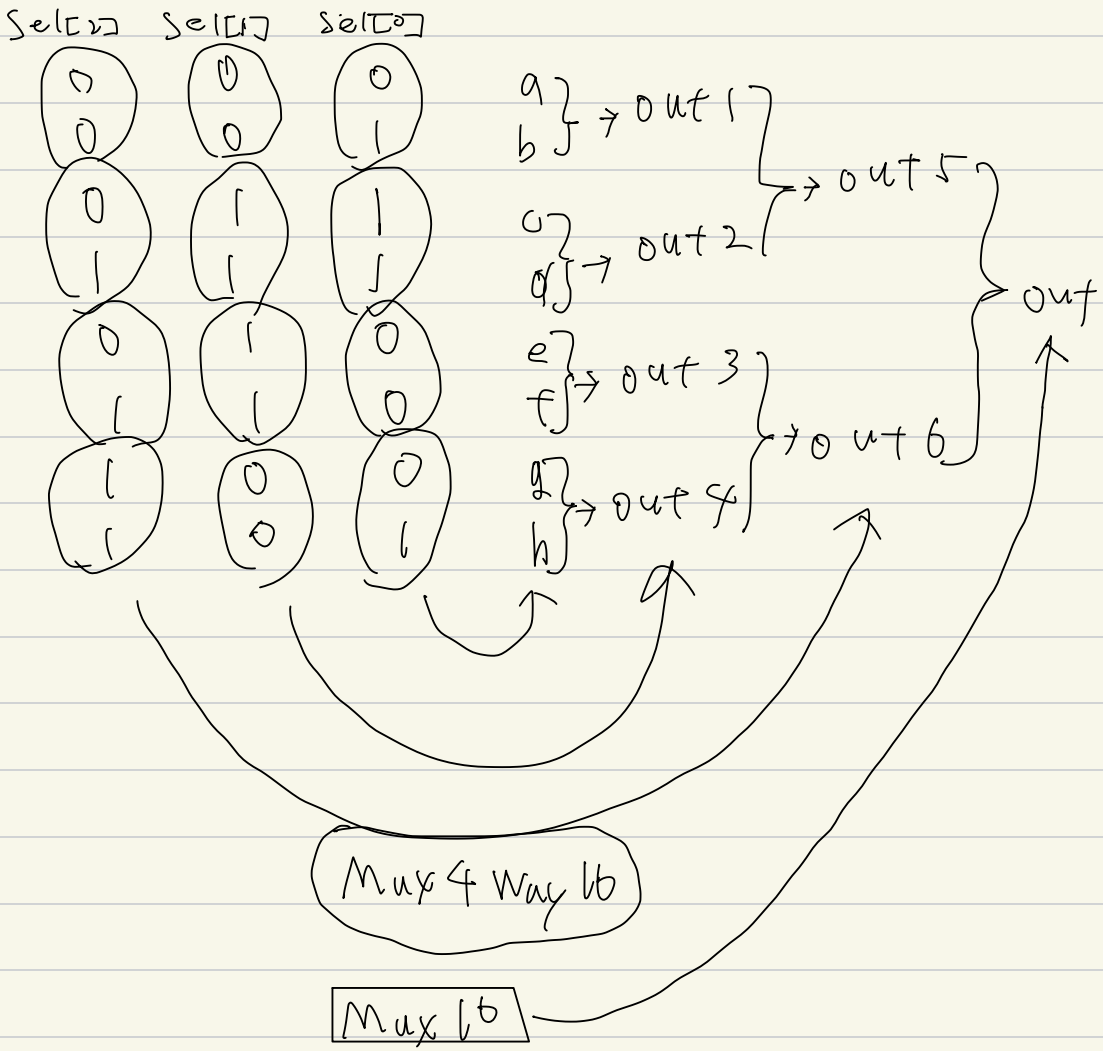


a	b	c	d	sel[1]	sel[0]	out
1	0	0	0	0	0	1
1	1	0	0	0	1	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

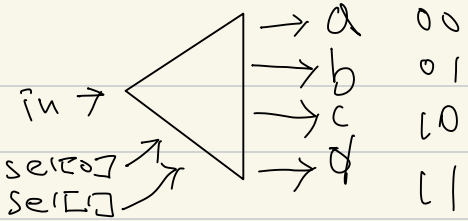
sel[1]が0ときは a,b を出力，sel[1]が1のときは c,d を出力

```
Mux16(a=a, b=b, sel=sel[0] out=out0); ←sel[1]=0
Mux16(a=c, b=d, sel=sel[0], out=out1); ←sel[1]=1
Mux16(a=out0, b=out1, sel=sel[1], out=out);
```

Mux 8 Way 16


sel[2] sel[1] sel[0]
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
a
b
→ out 1
c
d
→ out 2
e
f
→ out 3
g
h
→ out 4
→ out 5
→ out 6
out
Mux 4 Way 16
Mux 16

DMux4Way



in	sel[1]	sel[0]	a	b	c	d
0	0	0	0	0	0	0
1	0	1	0	1	0	0
0	1	0	0	0	0	0
1	1	1	0	0	0	1
1	0	0	1	0	0	0
0	0	1	0	0	0	0
1	1	0	0	0	1	0
0	1	1	0	0	0	0

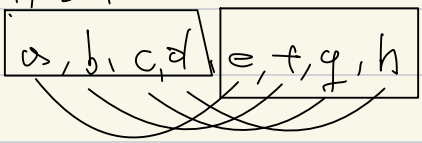
sel[1]が0のときにc,dを0にする。1のときはa,bを0に。

```
DMux(in=in, sel=sel[0], a=outa, b=outb);
~~DMux(in=in, sel=sel[0], a=outc, b=outd);~~
DMux(in=outa, sel=sel[1], a=a, b=c);
DMux(in=outb, sel=sel[1], a=b, b=d);
```

DMux 8Way

入力 in, sel[3]

出力 a, b, c, d, e, f, g, h



- sel[2] が 0 のとき e, f, g, h が in に一致する
- sel[2] が 1 のとき a, b, c, d が in に一致する。

```
DMux4Way(in=in, sel=sel[0..1], a=a, b=b, c=c, d=d);
DMux4Way(in=in, sel=sel[0..1], a=e, b=f, c=g, d=h);
DMux(in=a, sel=sel[2], a=a, b=e);
DMux(in=b, sel=sel[2], a=b, b=f);
DMux(in=c, sel=sel[2], a=c, b=g);
DMux(in=d, sel=sel[2], a=d, b=h);
```

2025/2/27 2章 ブール算術

半加算器 (HalfAdder)

入力 a, b

出力 Sum, Carry

a+bの { 最下位ビット → Sum / 最上位ビット → Carry

a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

```
Xor(a=a, b=b, out=sum);
AND(a=a, b=b, out=carry);
```

全加算器 (FullAdder)

入力 a, b, c
出力 carry, sum

a	b	carry	sum	c	sum	ccarry	csum
0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1
1	0	0	1	0	1	0	1
1	1	1	0	0	0	0	0
0	0	0	0	1	0	0	1
0	1	0	1	1	1	1	0
1	0	0	1	1	1	1	0
1	1	1	0	1	0	1	1

abcarry or ccarry = carry

csum = sum

```
HalfAdder(a=a, b=b, carry=abcarry, sum=absum);
HalfAdder(a=c, b=absum, carry=ccarry, sum=sum);
Or(a=abcarry, b=ccarry, out=carry);
```

16ビット加算器（Add16）

入力　a[16], b[16]

出力　out[16]

2ビットの場合

```
 10    a          11    a
 01  + b        11  + b
-----            -----
 11    out        110   out
```

（overflow（入力ビット幅を超える）は無視）

```
HalfAdder(a=a[0], b=b[0], carry=oc1, sum=out[0]);
FullAdder(a=a[1], b=b[1], c=oc1, carry=oc2, sum=out[1]);
        ⋮ nビット繰り返し
```

a[0]=1, b[0]=1, oc1=1, out[0]=0　　i=0

a[1]=1, b[1]=1, c=1, oc2=1, out[1]=1　　i=1

16ビットインクリメント [Inc16]　　入力 In[16]

出力 out[16]

関数 out = in + 1

・インクリメントするための 1 が必要 → true ピン4

・最下位ビット +1 は Xor で求められる → Sum = out[i]

・繰り上がりが発生するかは And で求められる。→ Carry

```
Xor(a=true, b=in[0], out=out[0]);
And(a=inc, b=in[0], out=carry0);
Xor(a=carry0, b=in[i], out=out[i]);
And(a=carry0, b=in[i], out=carry1);
```

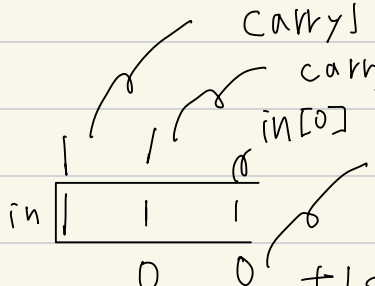
この方法で実装すると (Xor×16)+(And×15) の構成となる。

Add16.hdl を利用するよりも効率が良い

carry1 = carry0 And in[1]

carry0 = inc And in[0]

out[0] = inc Xor in[0]



+1 = inc

out[1] = carry0 Xor in[1]

加算器チップ (Add16.hdl) は

(HalfAdder×1)+(FullAdder×15) で構成される。

(Xor×1)+(And×1)+(HalfAdder×2)+(Or×1)　(Xor×3)+(And×3)+Or

(Xor+And)×2

Add16チップを利用すると (Xor×31)+(And×31)+Orを利用する。

算術論理演算器（ALU）

16ビット
入力 x, y, out,
zx, nx, zy, ny, f, no, out

出力 out[16], zr, ng

zx=1の時の x=0　And16(a=false, b=x, out=x);

nx=1の時の x=!x　Not16(in=x, out=x);

f=1の時の out=x+y　Add16(a=x, b=y, out=out);

f=0の時の out=x&y　And16(a=x, b=y, out=out);

no=1の時の out=!out　Not16(in=out, out=out);

分岐出力をMuxのa,bに入力、selに条件(zx,nx,f,no)をx入力して出力すれば1条件分岐による値の出力ができる。

Not16(in=x, out=notx);

Add16(a=x, b=y, out=xaddy);

And16(a=x, b=y, out=xandy);

Mux16(a=x, b=false, sel=zx, out=ozx);
Mux16(a=ozx, b=notx, sel=nx, out=onx);
} yも同じ

Mux16(a=xandy, b=xaddy, sel=f, out=of);

Not16(in=of, out=notof);

Mux16(a=of, b=notof, sel=no, out=out, out[0..7]=onolow8, out[8..15]=onohigh8, out[15]=ng);

Or8Way(in=onolow8, out=or8way1);

Or8Way(in=onohigh8, out=or8way2);

Or(a=or8way1, b=or8way2, out=or);

Not(in=or, out=zr);

2025/3/6 ～ Xモリ

フリップフロップ（DFF）は自身で実装せずにビルトインを利用する。
入力 in
出力 out

（1ビットレジスタ）
ビット（Bit）の実装

入力 in, load
出力 out

if load == 1 then out(t+1) = in(t)
else out(t+1) = out(t)

Mux(a=odff, b=in, sel=load, out=omux);
DFF(in=omux, out=out, out=odff);

16ビットレジスタ（Register） 入力 in[16], load
出力 out[16]

1ビットレジスタを16個内包している。

Bit(in=in[0], load=load, out=out[0]);
⋮
Bit(in=in[15], load=load, out=out[15]);

入力 in[16], load, address[k] ($k = \log_2 n$)

RAM8 (n)

出力 out[16]

RAMに内包される16ビットレジスタの数

addressで指定されたレジスタにloadを渡す。DMux8Way

DMux8Way(in=load, sel=address, a=ola, b=olb, c=olc, d=old,
e=ole, f=olf, g=olg, h=olh);

Register(in=in, load=ola, out=org0);
= = (in=in, load=olb, out=org1);
⋮
load=olh, out=orh7);

Mux8Way16(a=org0, b=org1, c=org2, d=org3, e=org4,
f=org5, g=org6, h=org7, sel=address, out=out);