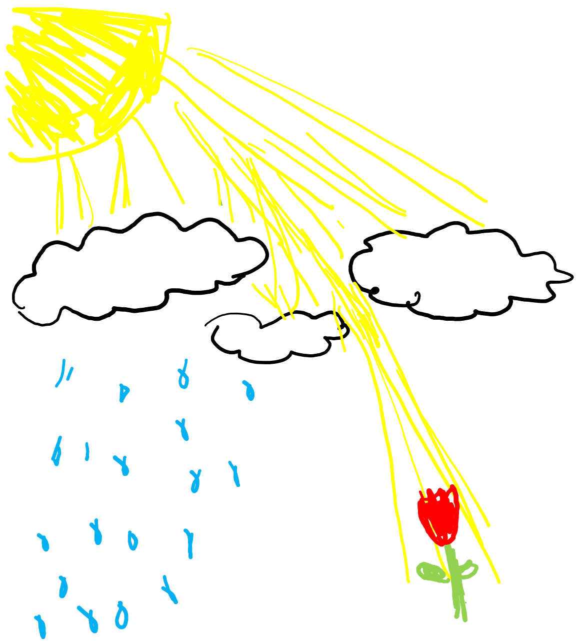
Warm up

Try to prove $p \rightarrow q \equiv \neg q \rightarrow \neg p$ if you didn't get all the way through it last time.

'A' lecture is a few minutes ahead of 'B' lecture.

First thing today is making sure we've answered all questions on this proof

Still confused on something? Ask in chat now and we'll start there.



Digital Logic

CSE 311 Autumn 2020
Lecture 4

Announcements

Everyone should have access to gradescope (you should have gotten a sign-up email if you don't already have an account).

If you can't access the course on gradescope, let us know as soon as possible.

Turning in an assignment to gradescope often takes about 15 minutes.

You have to tell gradescope which page each problem is on.

Contrapositive

$$
\begin{aligned}
p \to q &\equiv \neg p \lor q && \text{Law of Implication} \\
&\equiv q \lor \neg p && \text{Commutativity} \\
&\equiv \neg\neg q \lor \neg p && \text{Double Negation} \\
&\equiv \neg q \to \neg p && \text{Law of Implication}
\end{aligned}
$$

3+5

3-(-5)

All of our rules deal with ORs and ANDs, let's switch the implication to just use AND/NOT/OR.

And do the same with our target

It's ok to work from both ends. In fact it's a very common strategy!

Now how do we get the top to look like the bottom?

Just a few more rules and we're done!

Today

It’s notation day!
Two new different ways to represent propositions.

Also vocabulary catch-up.

Digital Logic

Digital Circuits

Computing With Logic

- **T corresponds to 1** or “high” voltage
- **F corresponds to 0** or “low” voltage

Gates

- Take inputs and produce outputs (functions)
- Several kinds of gates
- Correspond to propositional connectives (most of them)

And Gate

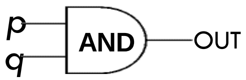
AND Connective vs. AND Gate

AND Connective

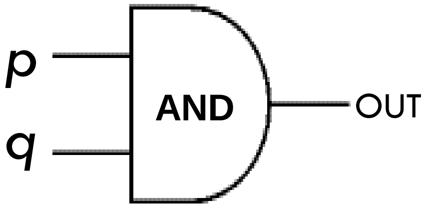
$p \wedge q$

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

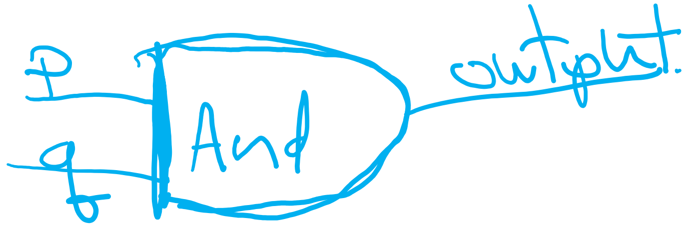
AND Gate



p	q	OUT
1	1	1
1	0	0
0	1	0
0	0	0



"block looks like D of AND"

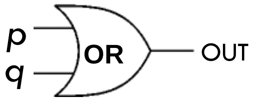


Or Gate

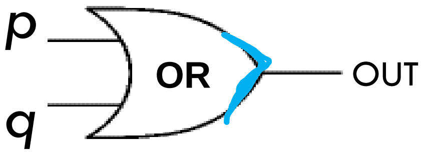
OR Connective **vs.** **OR Gate**

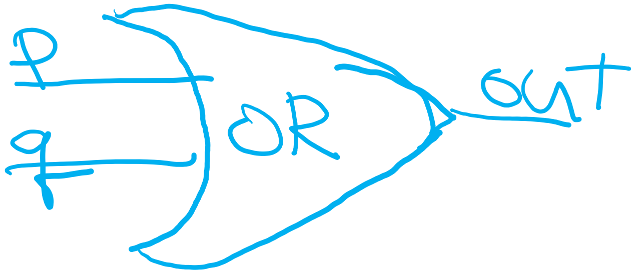
$p \vee q$

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F



p	q	OUT
1	1	1
1	0	1
0	1	1
0	0	0





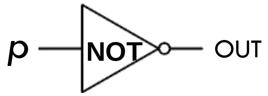
"arrowhead block looks like V"

Not Gates

NOT Connective vs. **NOT Gate**

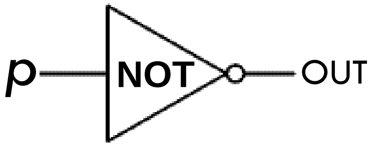
$\neg p$

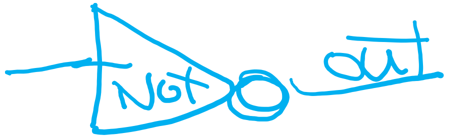
p	$\neg p$
T	F
F	T



Also called *inverter*

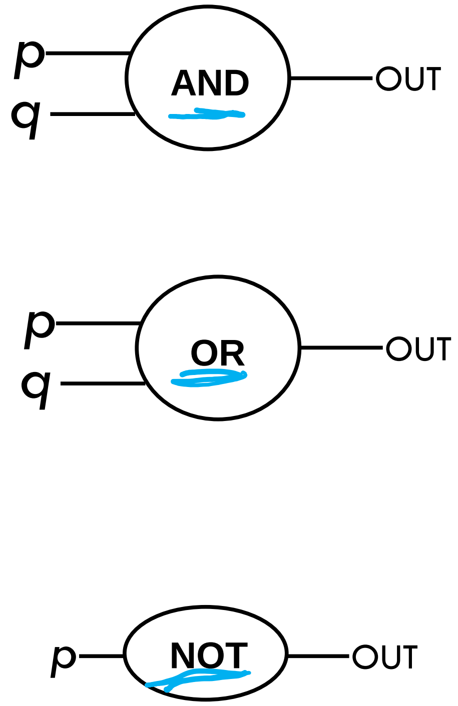
p	OUT
1	0
0	1



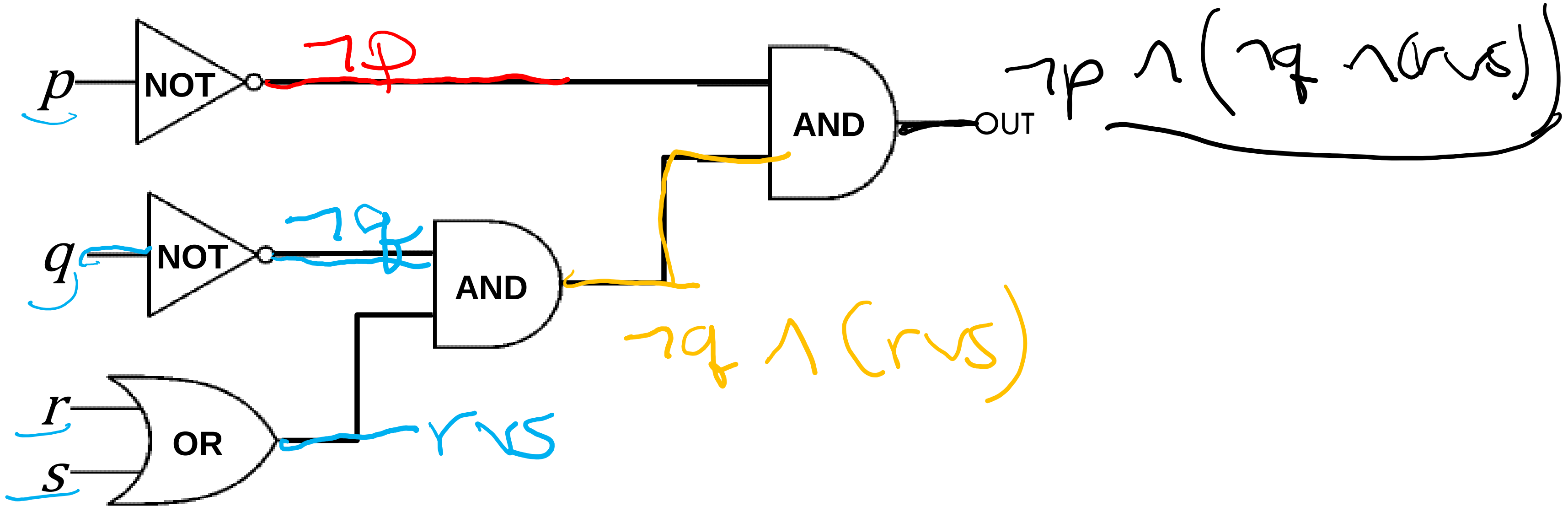


Blobs are Okay!

You may write gates using blobs instead of shapes!

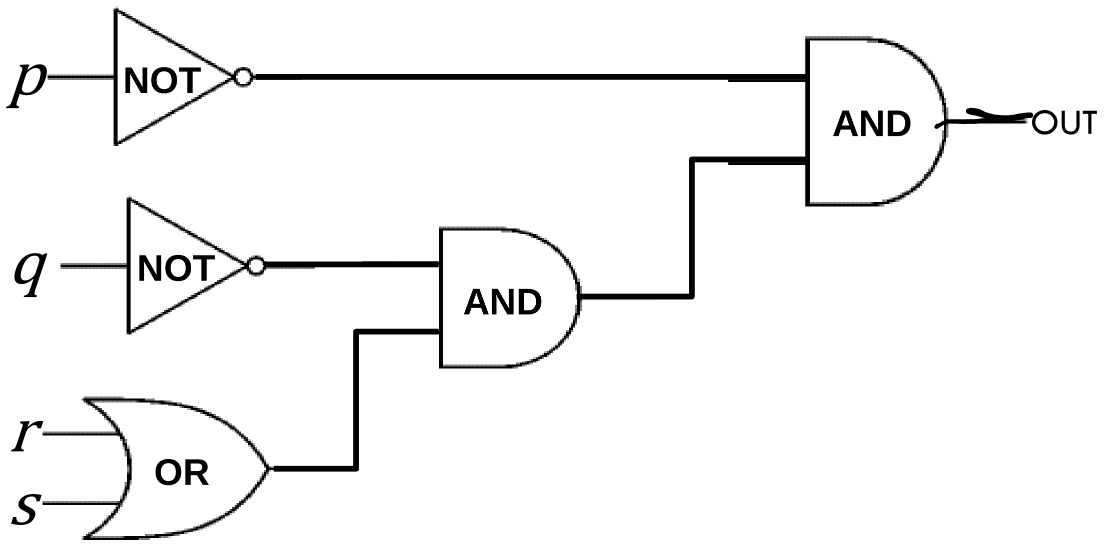


Combinational Logic Circuits



Values get sent along wires connecting gates

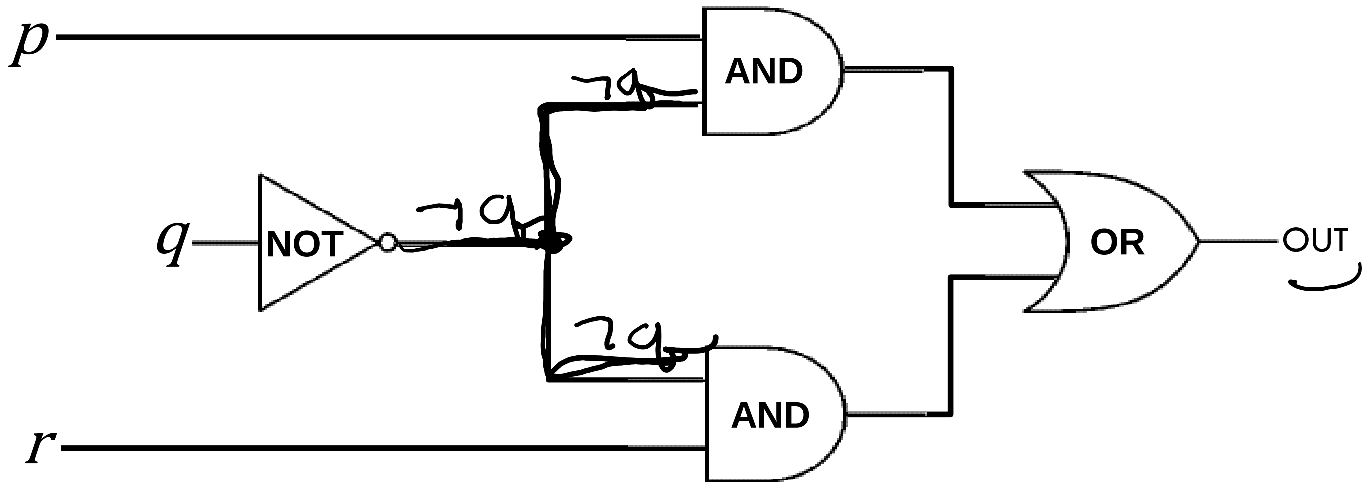
Combinational Logic Circuits



Values get sent along wires connecting gates

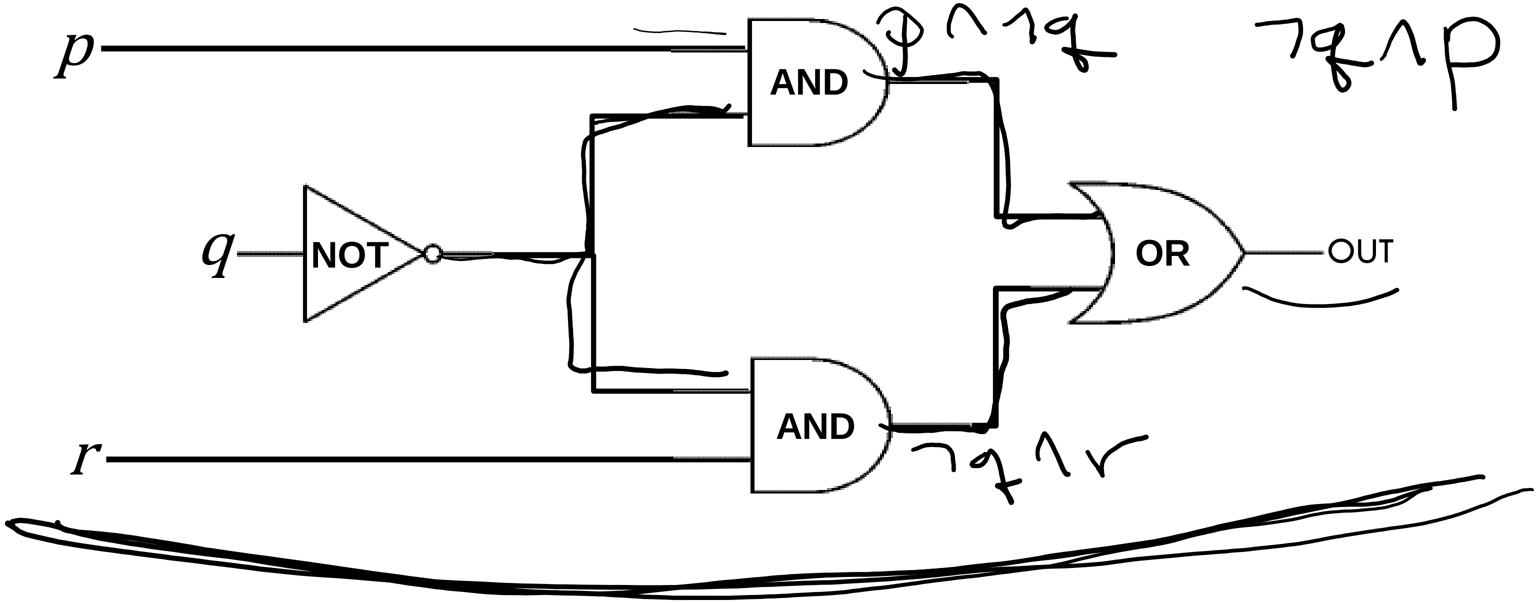
$$\neg p \land (\neg q \land (r \lor s))$$

Combinational Logic Circuits



Wires can send one value to multiple gates!

Combinational Logic Circuits



Wires can send one value to multiple gates!

$$(p \wedge \neg q) \vee (\neg q \wedge r)$$

Vocabulary Break!

Vocabulary!

A proposition is a….

Tautology if it is always true.
Contradiction if it is always false.
Contingency if it can be both true and false.

$p \vee \neg p$

Tautology

If p is true, $p \vee \neg p$ is true; if p is false, $p \vee \neg p$ is true.

$p \oplus p$

Contradiction

If p is true, $p \oplus p$ is false; if p is false, $p \oplus p$ is false.

$(p \rightarrow q) \wedge p$

Contingency If p is true and q is true, $(p \rightarrow q) \wedge p$ is true;
If p is true and q is false, $(p \rightarrow q) \wedge p$ is false.

More Vocabulary

$$p \to q$$

p is called the "hypothesis" or "antecedent" (or other names…)

q is called the "conclusion" or "consequent" (or other names…)

Back to Notation Day

On notation…

Logic is fundamental. Computer scientists use it in programs, mathematicians use it in proofs, engineers use it in hardware, philosophers use it in arguments,….

…so everyone uses different notation to represent the same ideas.

Since we don't know exactly what you're doing next, we're going to show you a bunch of them; but don't think one is "better" than the others!

Meet Boolean Algebra

Preferred by some mathematicians and circuit designers.

"or" is +

"and" is • (i.e. "multiply")

"not" is ' (an apostrophe after a variable)

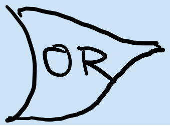
Why?

boolean semiring

Mathematicians like to study "operations that work kinda like 'plus' and 'times' on integers."

Circuit designers have a lot of variables, and this notation is more compact.

Meet Boolean Algebra

Name	Variables	"True/False"	"And"	"Or"	"Not"	Implication
Java Code	`boolean b`	`true,false`	`&&`	`||`	`!`	No special symbol
Propositional Logic	"p, q, r"	T, F	$\wedge$	$\vee$	$\neg$	$\rightarrow$
Circuits	Wires	1, 0				No special symbol
Boolean Algebra	a, b, c	1,0	$\cdot$ ("multiplication")	$+$ ("addition")	$'$ (apostrophe after variable)	No special symbol

Propositional logic

$(p \wedge q \wedge r) \vee s \vee \neg t$

Boolean Algebra

$pqr + s + t'$

Comparison

Propositional logic	Boolean Algebra
$(p \wedge q \wedge r) \vee s \vee \neg t$	$pqr + s + t'$

Remember this is just an alternate notation for the same underlying ideas.

So that big list of identities? Just change the notation and you get another big list of identities!

Boolean Algebra

Axioms

Closure
$a + b$ is in $\mathbb{B}$
$a \bullet b$ is in $\mathbb{B}$

Commutativity
$a + b = b + a$
$a \bullet b = b \bullet a$

Associativity
$a + (b + c) = (a + b) + c$
$a \bullet (b \bullet c) = (a \bullet b) \bullet c$

Identity
$a + 0 = a$
$a \bullet 1 = a$

Distributivity
$a + (b \bullet c) = (a + b) \bullet (a + c)$
$a \bullet (b + c) = (a \bullet b) + (a \bullet c)$

Complementarity
$a + a' = 1$
$a \bullet a' = 0$

Boolean Algebra

Theorems

Null

$$X + 1 = 1$$
$$X \bullet 0 = 0$$

Idempotency

$$X + X = X$$
$$X \bullet X = X$$

Involution

$$(X')' = X$$

Uniting

$$X \bullet Y + X \bullet Y' = X$$
$$(X + Y) \bullet (X + Y') = X$$

Boolean Algebra

Absorbtion

$$X + X \bullet Y = X$$
$$(X + Y') \bullet Y = X \bullet Y$$
$$X \bullet (X + Y) = X$$
$$(X \bullet Y') + Y = X + Y$$

DeMorgan

$$(X + Y + \cdots)' = X' \bullet Y' \bullet \cdots$$
$$(X \bullet Y \bullet \cdots)' = X' + Y' + \cdots$$

Consensus

$$(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z) = X \bullet Y + X' \bullet Z$$
$$(X + Y) \bullet (Y + Z) \bullet (X' + Z) = (X + Y) \bullet (X' + Z)$$

Factoring

$$(X + Y) \bullet (X' + Z) = X \bullet Z + X' \bullet Y$$
$$X \bullet Y + X' \bullet Z = (X + Z) \bullet (X' + Y)$$

An Exercise in Notation

The rest of today we're solving a problem.

See the concepts we learned the last few days "in action"

And practice Boolean algebra and propositional logic.

Today's Goal

Go from a problem statement to code to logical/circuit representation to an "optimized" version.

Why?

- Practice translating between different representations.
- Practice applying simplification laws
- Historical context! This process is reminiscent of "hardware acceleration" – designing custom hardware to do a single task very fast.
  - Most design is done automatically these days, but it's still nice to see once.

Our Goal

Given what day of the week it is and what kind of question you have, what's the quickest way to get it answered?

(this is an example, not actual advice)

Input: day of the week, Boolean talkToSomeone

Output: The way to get your question answered, according to the following rules:

On M,Tu,W,F if you want to talk, go to office hours

On Th if you want to talk, go to section

Monday through Friday, if you don't want to talk ask on Ed

On Saturday or Sunday, text a friend (whether you want to talk or not)

Step One

Input: day of the week, Boolean talkToSomeone

Output: The way to get your question answered, according to the following rules:

On M,Tu,W,F if you want to talk, go to office hours

On Th if you want to talk, go to section

Monday through Friday, if you don’t want to talk ask on Ed

On Saturday or Sunday, text a friend (whether you want to talk or not)

Take 2 minutes **plan** what your code might look like.

Step One

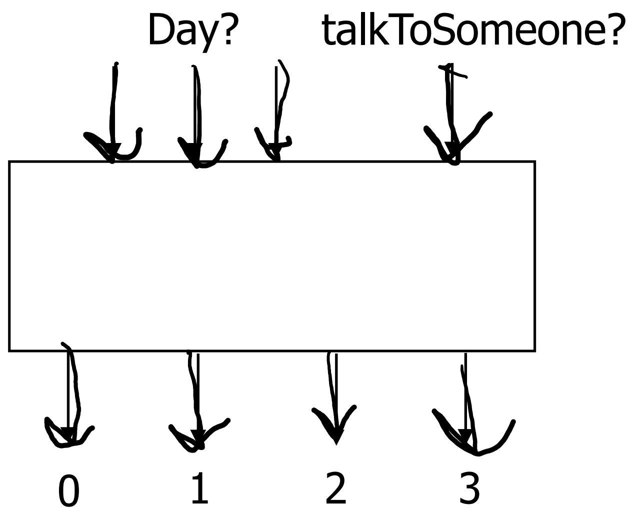
```
if( (day==Monday || day==Tuesday || day==Wednesday || day==Friday) ) {
    if(talkToSomeone)
        return "office hours";
    else
        return "Ed";
}
else if(day==Thursday) {
    if(talkToSomeone)
        return "section";
    else
        return "Ed";
}
else //day is Saturday or Sunday
    return "text a friend";

```

One possibility (there are many)

Step Two

Go from a problem statement to code to logical/circuit representation to an “optimized” version.

We want a logical/circuit representation.



Step Two

Input? Day in binary and talkToSomeone

Monday – 000

Tuesday – 001

Wednesday – 010

Thursday – 011

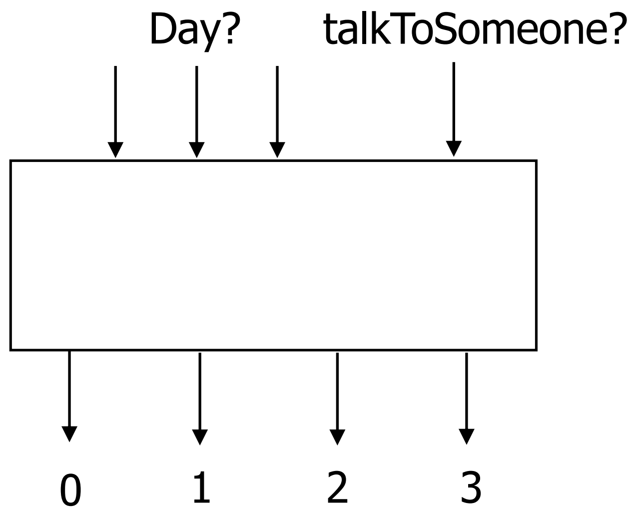
Friday – 100

Saturday – 101

Sunday – 110

(invalid) – 111

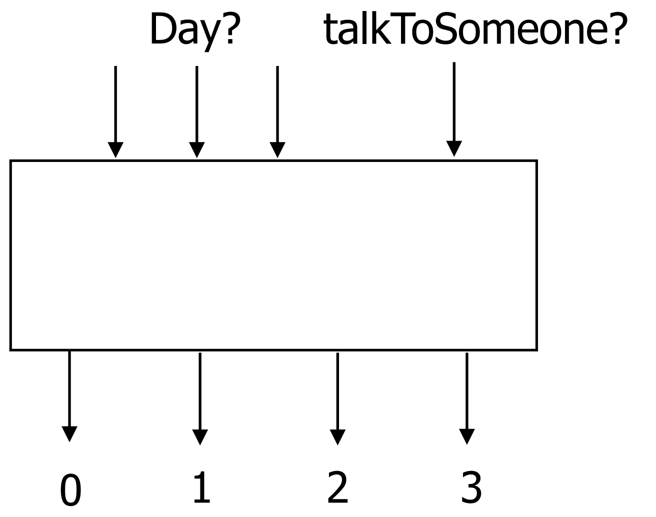
0 for false, 1 for true.



Step Two

Output? We'll turn on only the wire for what to do

called a "one-hot" encoding, because one wire is on ('hot')



Office Hour – 0

Section – 1

Ed – 2

Text a Friend – 3

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				1
Sunday	1	1	0	1				1
---	1	1	1	0				
---	1	1	1	1				

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

$d_2\ d_1\ d_0\ s$

$\neg d_2 \land \neg d_1 \land \neg d_0 \land s$

$\neg d_2 \land \neg d_1 \land d_0 \land s$

$\neg d_2 \land d_1 \land \neg d_0 \land s$

$d_2 \land \neg d_1 \land \neg d_0 \land s$

$$out_0 = (\neg d_2 \land \neg d_1 \land \neg d_0 \land s) \lor (\neg d_2 \land \neg d_1 \land d_0 \land s) \lor (\neg d_2 \land d_1 \land \neg d_0 \land s) \lor (d_2 \land \neg d_1 \land \neg d_0 \land s)$$

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)	
Monday	0	0	0	0			1		
Monday	0	0	0	1	1				$d_2'd_1'd_0's$
Tuesday	0	0	1	0			1		
Tuesday	0	0	1	1	1				$d_2'd_1'd_0s$
Wednesday	0	1	0	0			1		
Wednesday	0	1	0	1	1				$d_2'd_1d_0's$
Thursday	0	1	1	0			1		
Thursday	0	1	1	1		1			
Friday	1	0	0	0			1		
Friday	1	0	0	1	1				$d_2d_1'd_0's$
Saturday	1	0	1	0				1	
Saturday	1	0	1	1					
Sunday	1	1	0	0					
Sunday	1	1	0	1					
---	1	1	1	0					
---	1	1	1	1					

$$out_0 = d_2'd_1'd_0's + d_2'd_1'd_0s + d_2'd_1d_0's + d_2d_1'd_0's$$

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

$d_2'd_1'd_0's$

$d_2'd_1'd_0s$

$d_2'd_1d_0's$

$d_2d_1'd_0's$

$$out_0 = (d_2'd_1'd_0' + d_2'd_1'd_0 + d_2'd_1d_0' + d_2d_1'd_0')s$$

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				1
Sunday	1	1	0	1				1
---	1	1	1	0				
---	1	1	1	1				

Find the formula for out_2 in both Boolean algebra and propositional logic.

If you have extra time, draw the circuit representation.

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse311 and login with your UW identity
Or text cse311 to 22333

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

$$out_1 = d_2' d_1 d_0 s$$

$$out_1 = \neg d_2 \wedge d_1 \wedge d_0 \wedge s$$

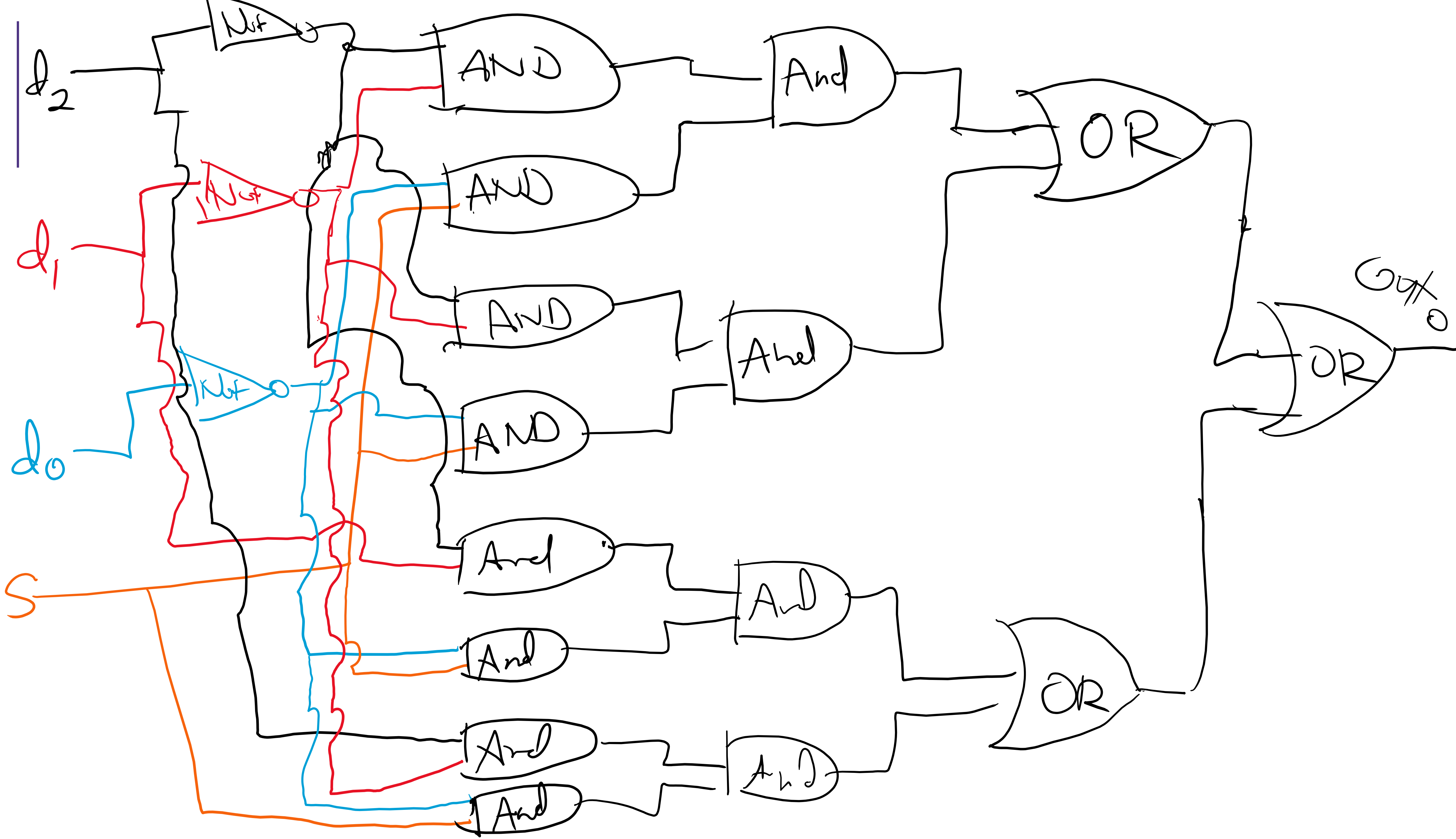
Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0			1	
Tuesday	0	0	1	1	1			
Wednesday	0	1	0	0			1	
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				
Sunday	1	1	0	1				
---	1	1	1	0				
---	1	1	1	1				

$$out_2 = d_2'd_1'd_0's' + d_2'd_1'd_0s' + d_2'd_1d_0s' + d_2d_1'd_0's'$$

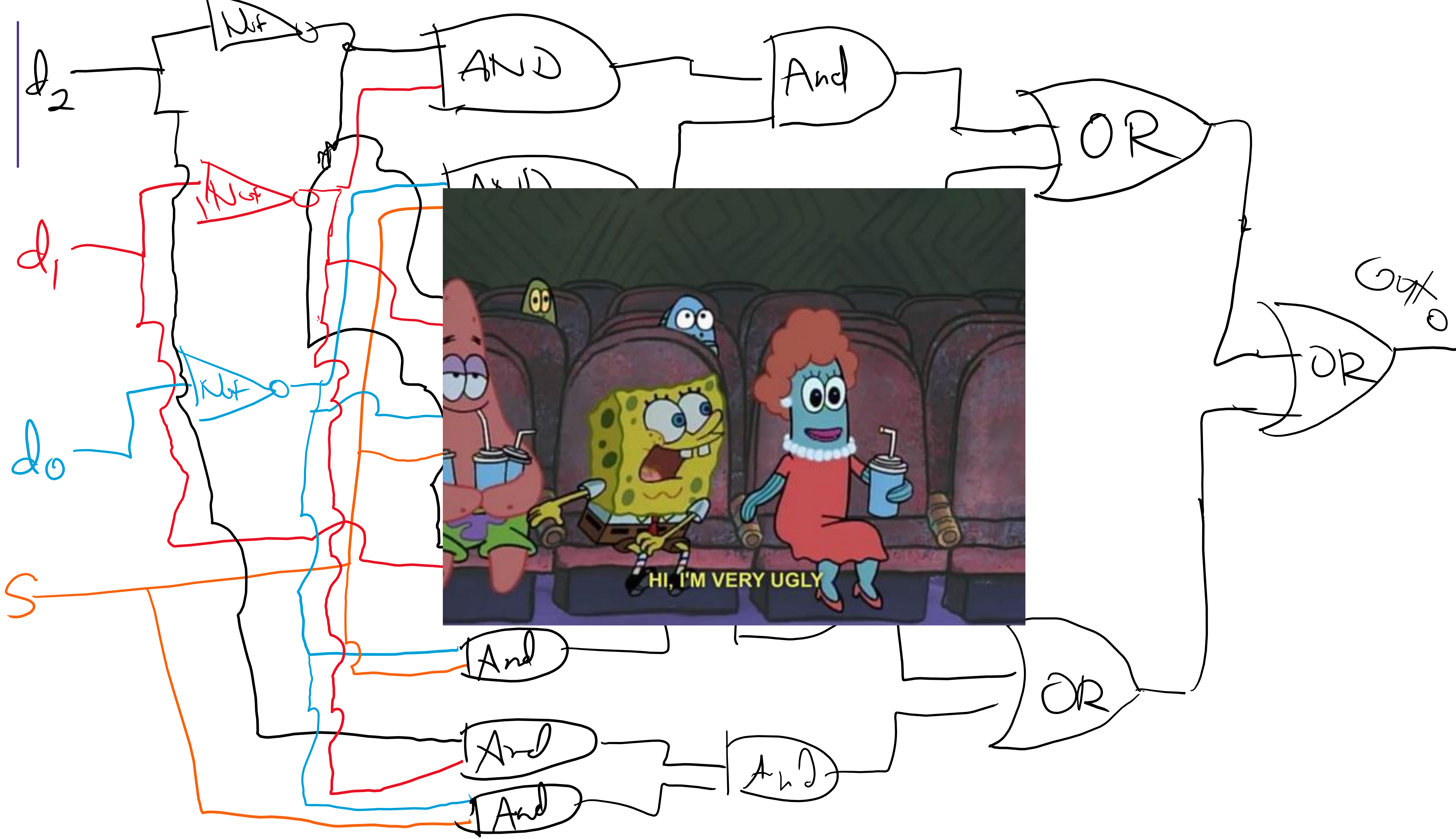
$$out_2 = d_2's'(d_1'd_0' + d_1'd_0 + d_1d_0) + d_2d_1'd_0's'$$

Day	d_2	d_1	d_0	talkToSomeone	out_0 (OH)	out_1 (Se)	out_2 (Ed)	out_3 (TF)
Monday	0	0	0	0			1	
Monday	0	0	0	1	1			
Tuesday	0	0	1	0				
Tuesday	0	0	1	1				
Wednesday	0	1	0	0				
Wednesday	0	1	0	1	1			
Thursday	0	1	1	0			1	
Thursday	0	1	1	1		1		
Friday	1	0	0	0			1	
Friday	1	0	0	1	1			
Saturday	1	0	1	0				1
Saturday	1	0	1	1				1
Sunday	1	1	0	0				1
Sunday	1	1	0	1				1
---	1	1	1	0				
---	1	1	1	1				

$$out_3 = d_2(d_1' d_0 + d_1 d_0' + d_1 d_0)$$



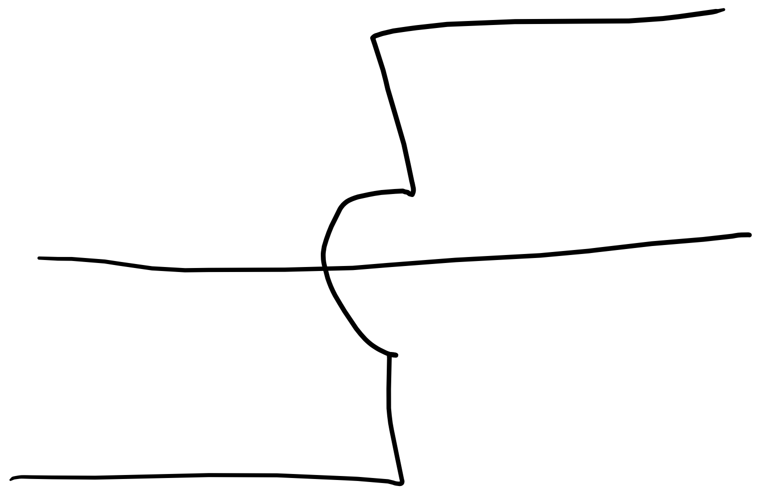
d2
d1
d0
S
Not
Not
Not
AND
AND
AND
AND
And
And
And
And
And
And
And
And
And
OR
OR
OR
Out0

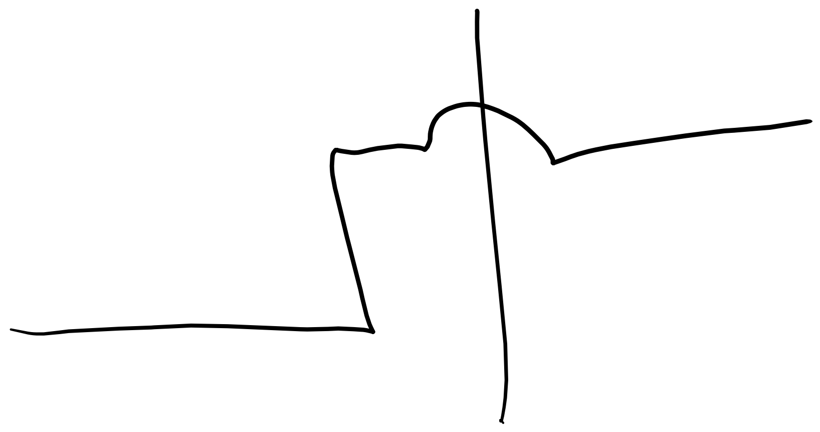

d2
d1
d0
S
Not
Not
Not
AND
And
AND
And
And
And
And
OR
OR
OR
Out0
HI, I'M VERY UGLY

Ick

WOW that's ugly.

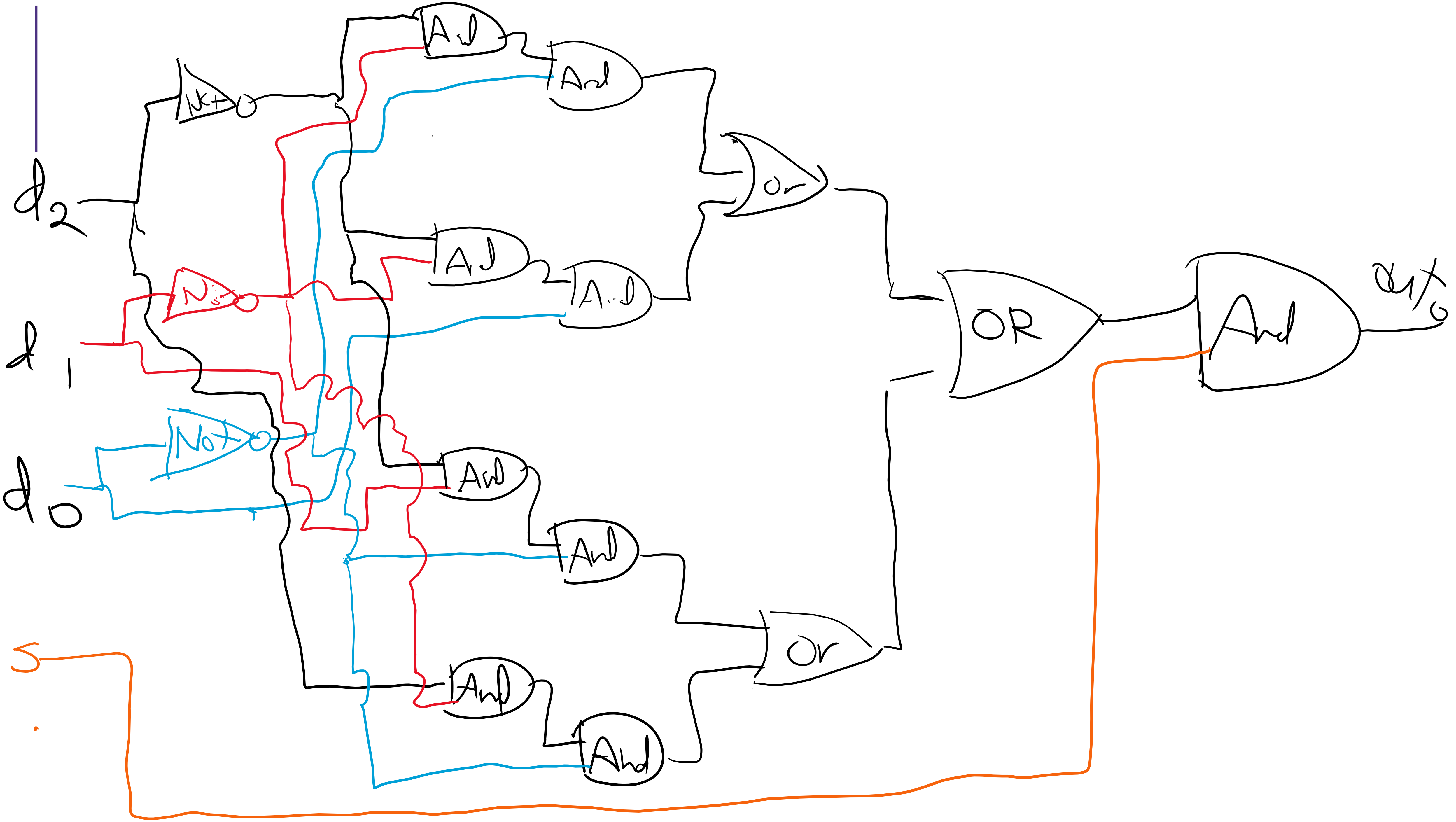
Be careful when wires cross – draw one "jumping over" the other.



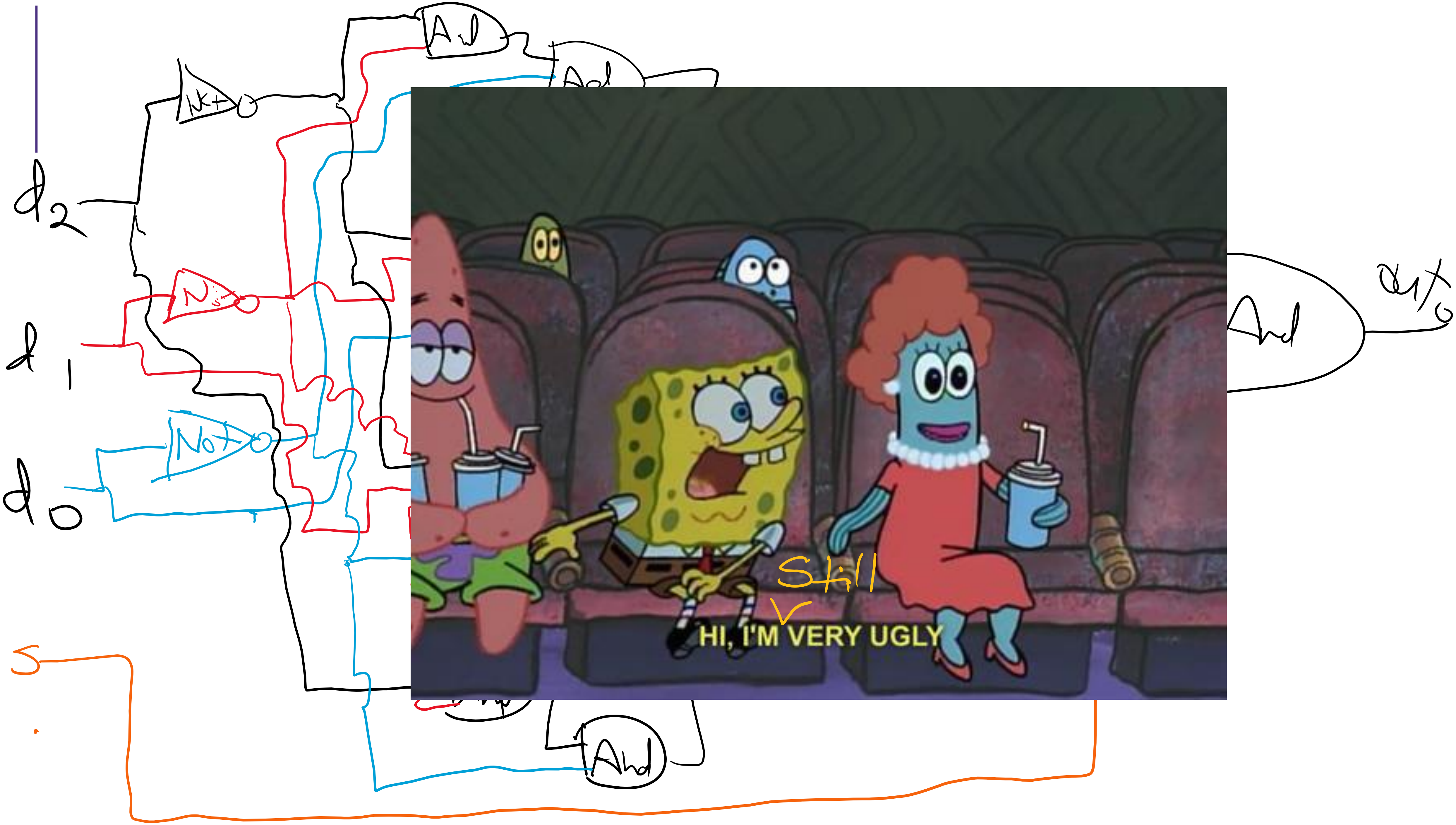


Can we do better

Maybe the factored version will be better?



d2
d1
d0
S
Not
Not
Not
And
And
And
And
And
And
And
And
Or
Or
OR
And



d2
d1
d0
S
Not
Not
Not
And
And
And
And
Out
Still
HI, I'M VERY UGLY

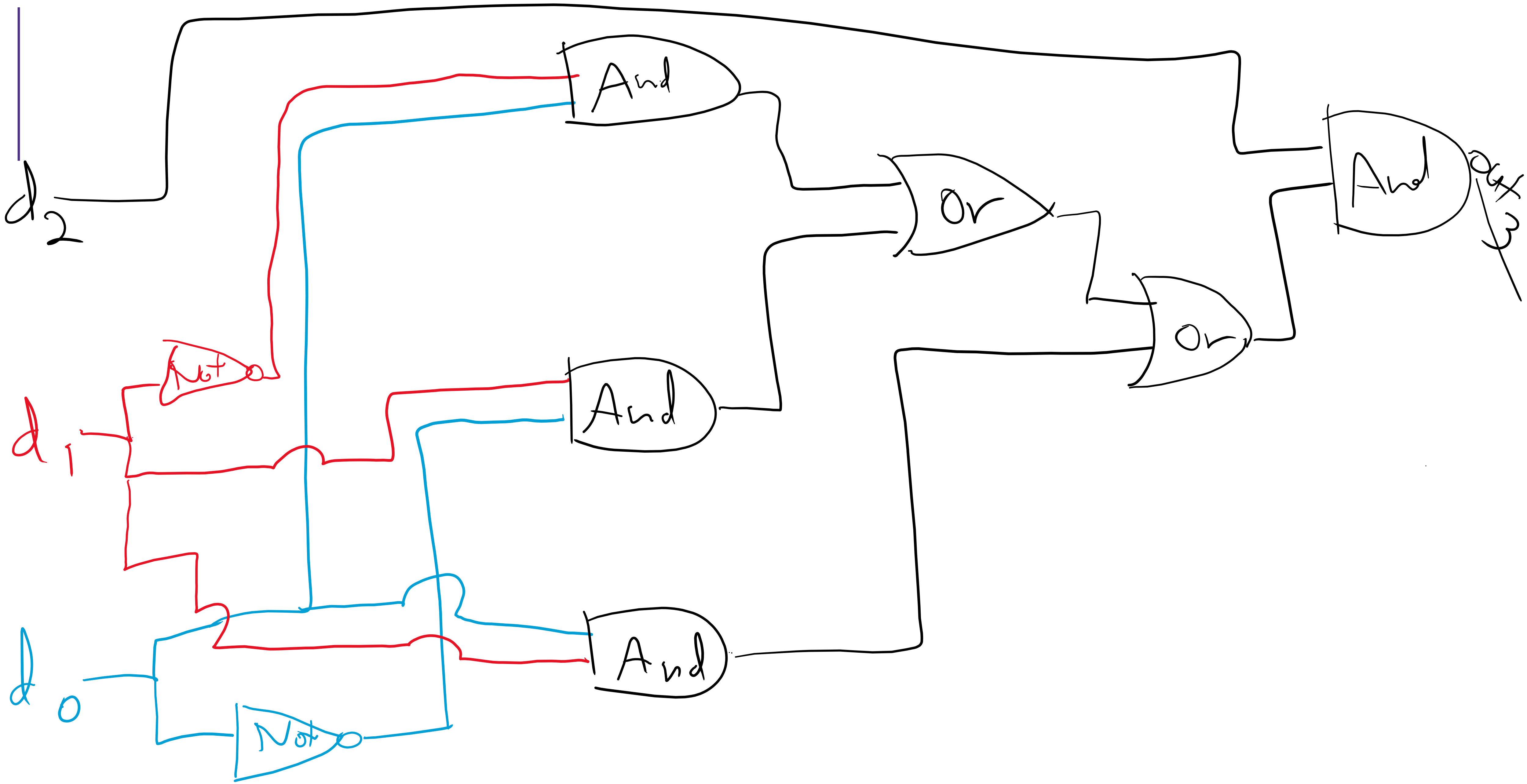
Ehhhhhhh, it’s a little better?

Part of the problem here is Robbie’s art skills.

Part is some layout choices – commuting the terms might make things prettier.

Most of the problem is just the circuit is complicated.

out_3 is a little better.



d2
d1
d0
Not
Not
And
And
And
Or
Or
And

Can we use these for anything?

Sometimes these concrete formulas lead to easier observations.

For example, we might have noticed we factored out s or s' in three of the four, which suggests switching s first.

```
if(talkToSomeone) {
    if( (day==Monday || day==Tuesday || day==Wednesday || day==Friday) )
        return "office hours";
    else if( day==Thursday)
        return "section";
    else
        return "text a friend";
}
else {
    if( (day==Monday || day==Tuesday || day==Wednesday || day==Thursday || day==Friday) )
        return "Ed";
    else
        return "text a friend";
}
```

Can we use these for anything?

Is this code better? Maybe, maybe not.

It's another tool in your toolkit for thinking about logic

Including logic you write in code!

```
if(talkToSomeone) {
    if( (day==Monday || day==Tuesday || day==Wednesday || day==Friday) )
        return "office hours";
    else if( day==Thursday)
        return "section";
    else
        return "text a friend";
}
else {
    if( (day==Monday || day==Tuesday || day==Wednesday || day==Thursday || day==Friday) )
        return "Ed";
    else
        return "text a friend";
}
```

Takeaways

Yet another notation for propositions.

These are just more representations – there's only **one** underlying set of rules.

Next time: wrap up digital logic and the tool really represent $x > 5$.

Another Proof

Let's prove that $(p \wedge q) \rightarrow (q \vee p)$ is a tautology.

Alright, what are we trying to show?

Another Proof

$$
\begin{aligned}
(p \wedge q) \rightarrow (q \vee p) &\equiv \neg(p \wedge q) \vee (q \vee p) \\
&\equiv (\neg p \vee \neg q) \vee (q \vee p) \\
&\equiv \neg p \vee (\neg q \vee (q \vee p)) \\
&\equiv \neg p \vee \big((\neg q \vee q) \vee p\big) \\
&\equiv \neg p \vee ((q \vee \neg q) \vee p) \\
&\equiv \neg p \vee (\mathrm{T} \vee p) \\
&\equiv \neg p \vee (p \vee \mathrm{T}) \\
&\equiv \neg p \vee p \\
&\equiv p \vee \neg p \\
&\equiv \mathrm{T}
\end{aligned}
$$

Proof-writing tip:

Take a step back.

Pause and carefully look at what you have. You might see where to go next…

Law of Implication

DeMorgan's Law

Associative (twice)

Commutative, Negation

Commutative, Domination

Commutative, Negation

It's easier if everything is AND/OR/NOT

Gets rid of some parentheses/just a gut feeling

Put $q, \neg q$ next to each other.

Simplify out the $q, \neg q$.

Simplify out the T.

Simplify out the $p, \neg p$.

We're done!

Another Proof

$(p \wedge q) \rightarrow (q \vee p)$	$\equiv \neg(p \wedge q) \vee (q \vee p)$	Law of implication
	$\equiv (\neg p \vee \neg q) \vee (q \vee p)$	DeMorgan's Law
	$\equiv \neg p \vee (\neg q \vee (q \vee p))$	Associative
	$\equiv \neg p \vee \big((\neg q \vee q) \vee p\big)$	Associative
	$\equiv \neg p \vee ((q \vee \neg q) \vee p)$	Commutative
	$\equiv \neg p \vee (\mathrm{T} \vee p)$	Negation
	$\equiv \neg p \vee (p \vee \mathrm{T})$	Commutative
	$\equiv \neg p \vee p$	Domination
	$\equiv p \vee \neg p$	Commutative
	$\equiv \mathrm{T}$	Negation