

Complex synthetic biology: Logic to the Moon (and back)

Dr Tom Ellis

January 2019

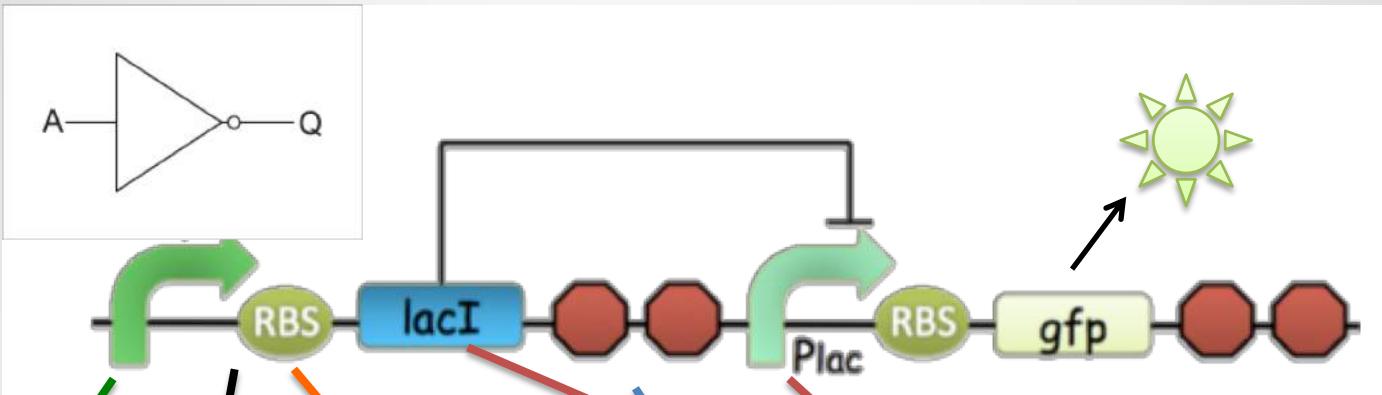
Lecture Content

- Bacterial Logic Gates from Promoters
- Circuitry that is exclusively NOT/NOR gates
- How NOT gates become NOR gates
- Cello – a Verilog for Synthetic Biology
- How Cello works and can be implemented
- Logic with the dCas9 customisable transcription factor system
- Parts performing logic for less cost to the cell

Learning Objectives

- To explain logic functions in synthetic biology
- To understand how all logic is possible with NOR gates
- To describe efforts towards predictable design
- To show how design is now automatable
- To demonstrate progress of the field towards large complex networks built with professional parts

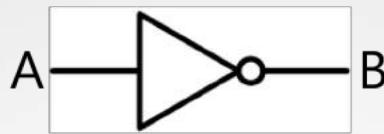
Inverter Network



- 100+ constitutive promoters
- RiboJ part or Csy4 sites to remove 5' context
- RBS custom design using RBS Calculator
 - BCD method gives 50+ modular RBS parts
- 16 TetR-related TF-promoter pairs\
- ZF-TFs/TALES in yeast
- 10000+ dCas9 options
- 600+ measured terminators
- Two terminator design models

Standard Logic Gates

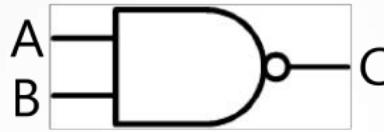
NOT



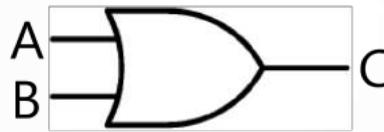
AND



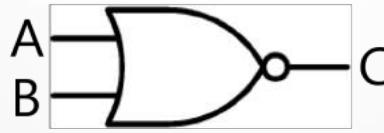
NAND



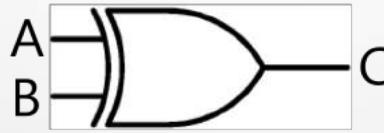
OR



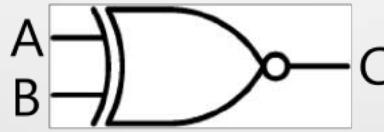
NOR



XOR



XNOR



A	B	C
1	0	
0	1	

1	1	1
1	0	0
0	1	0
0	0	0

1	1	0
1	0	1
0	1	1
0	0	1

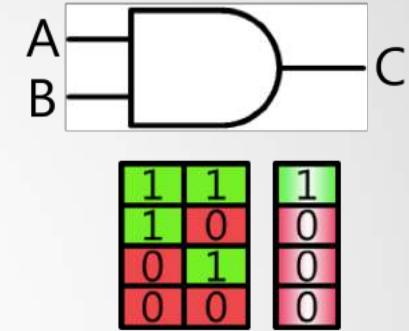
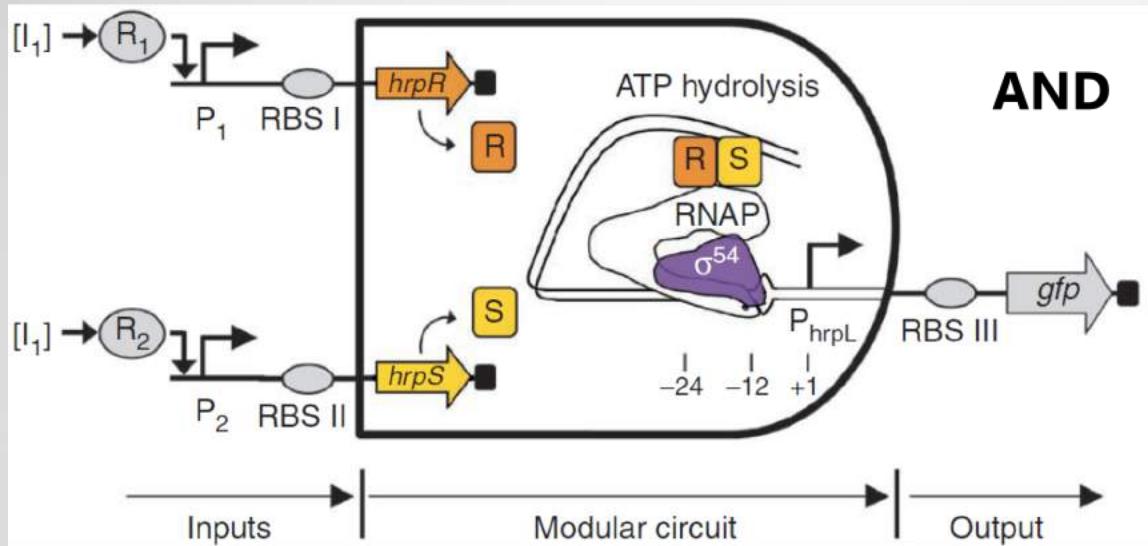
1	1	1
1	0	1
0	1	1
0	0	0

1	1	0
1	0	0
0	1	0
0	0	1

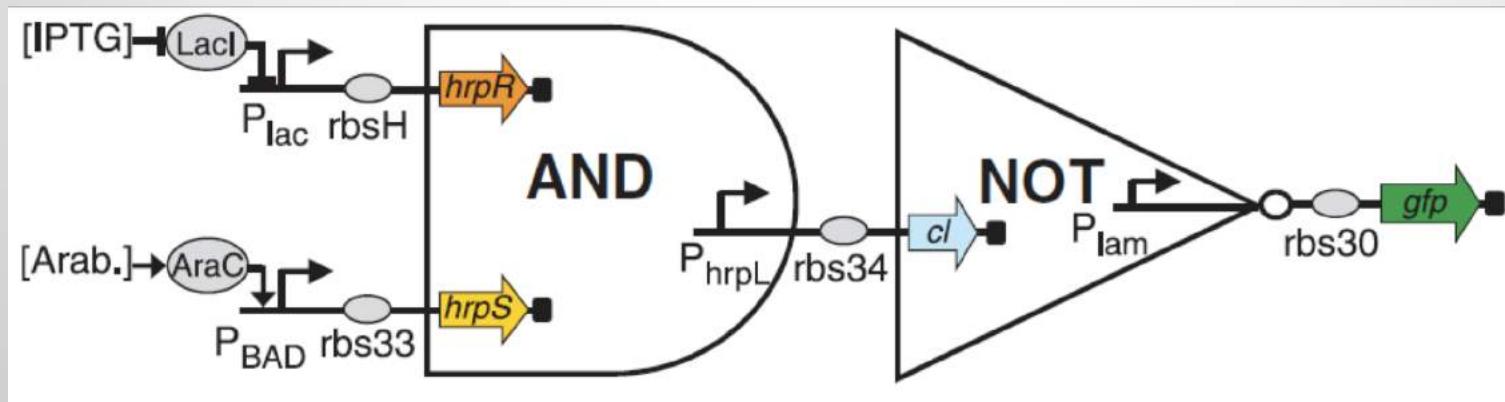
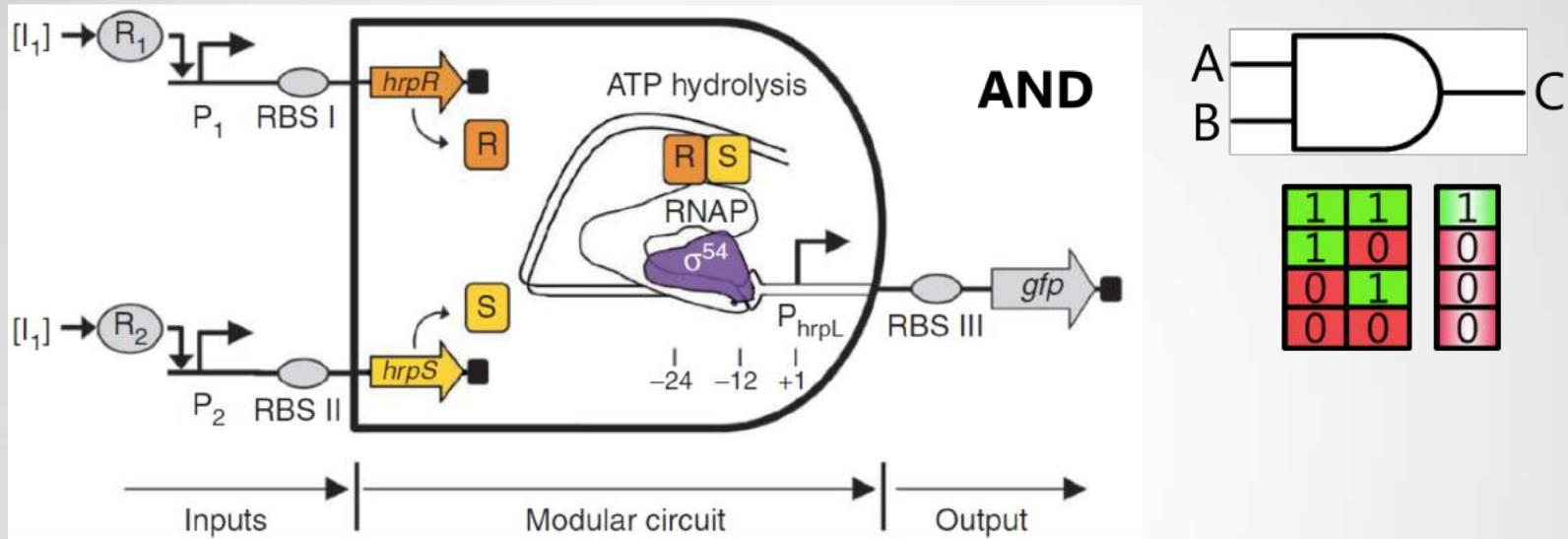
1	1	0
1	0	1
0	1	1
0	0	0

1	1	1
1	0	0
0	1	0
0	0	1

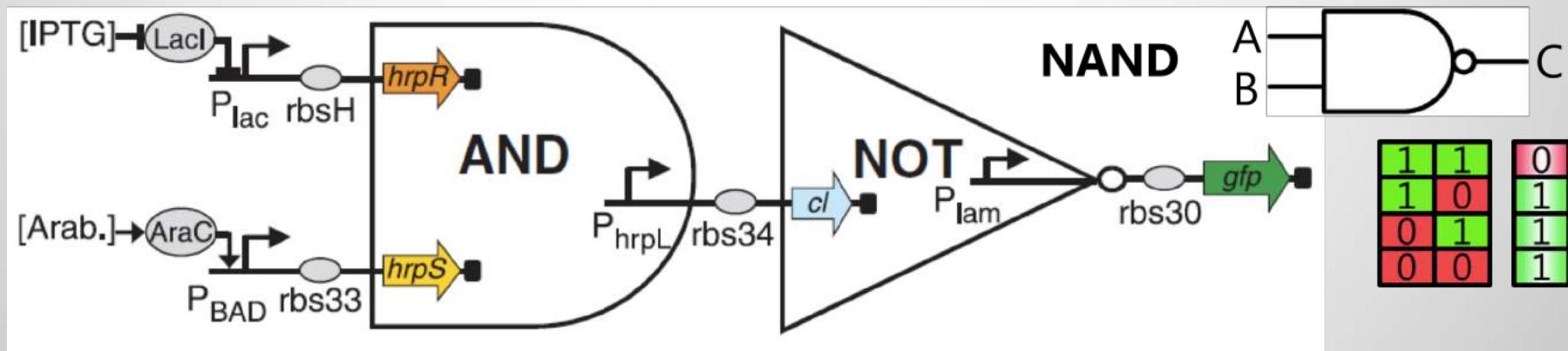
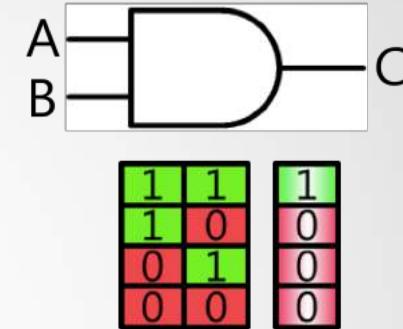
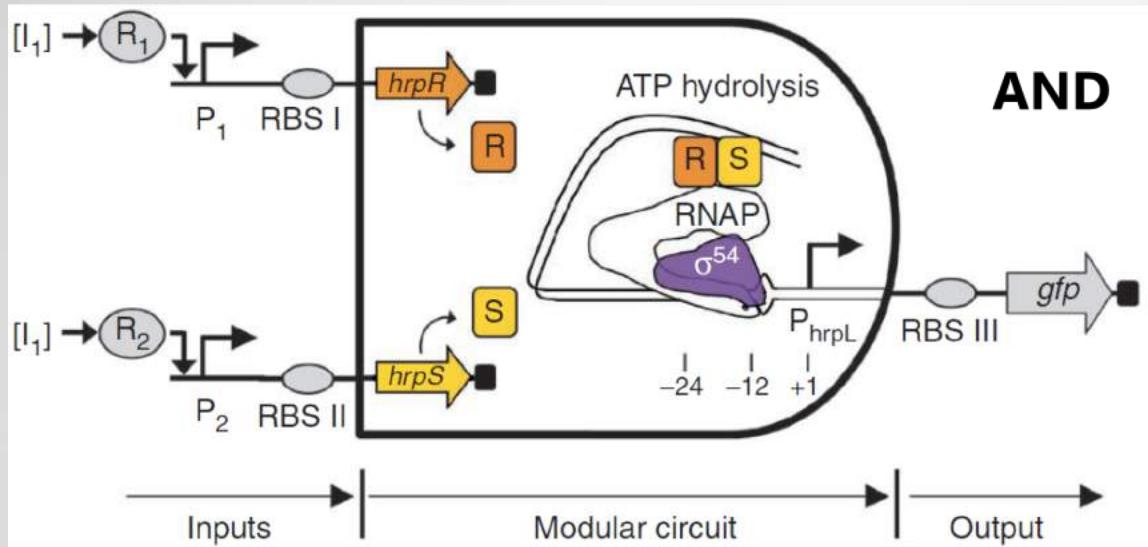
Bacterial Logic Gates - Examples



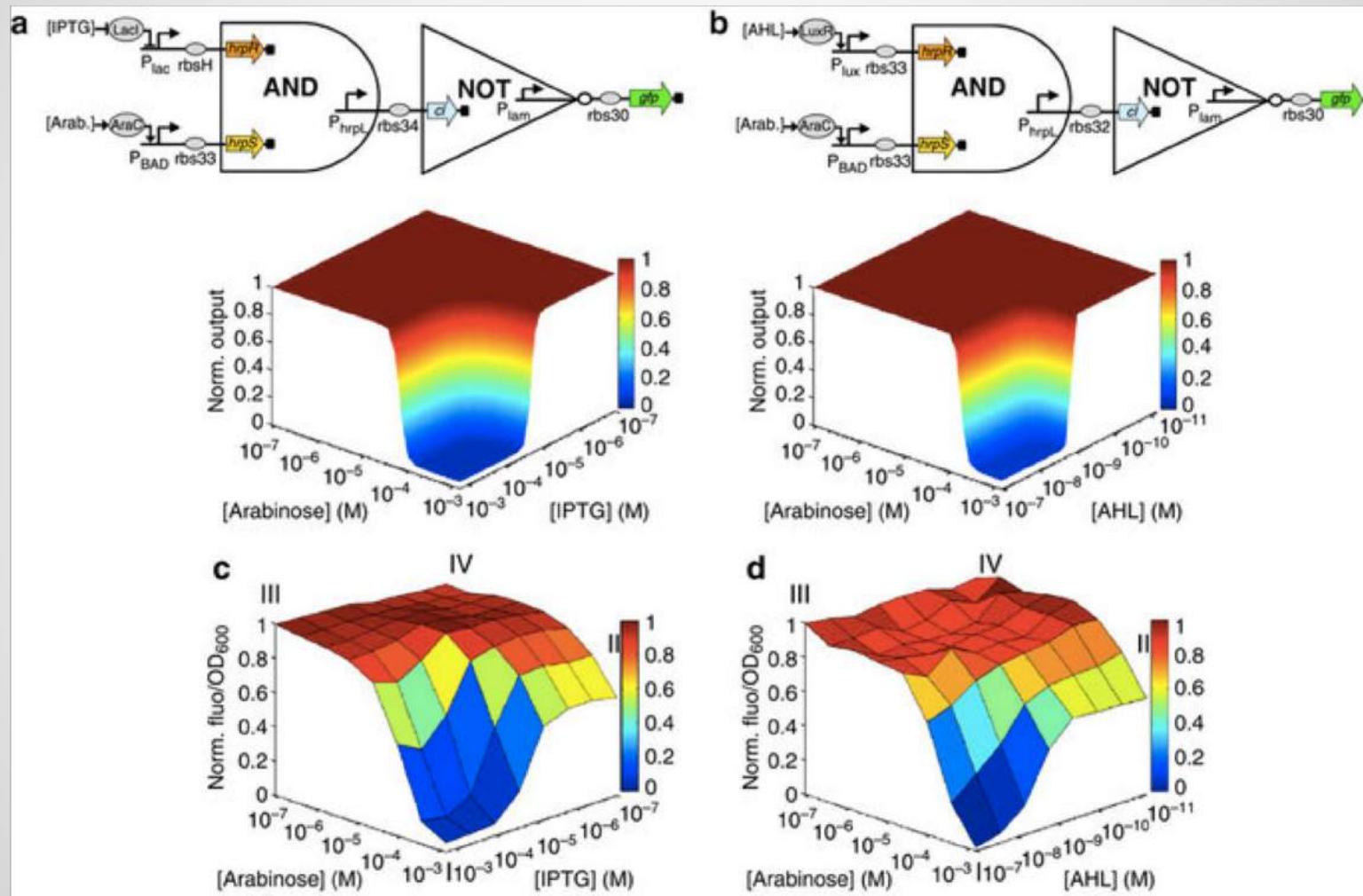
Bacterial Logic Gates - Examples



Bacterial Logic Gates - Examples



Bacterial Logic Gates - Examples

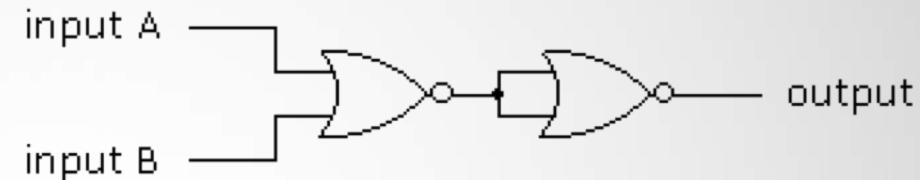


Universality of NOR gates

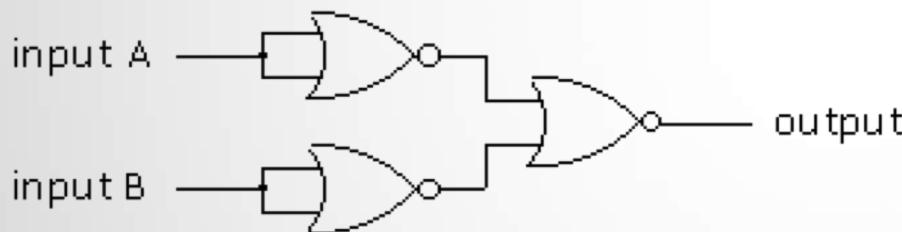
NOT gate (inputs joined together)



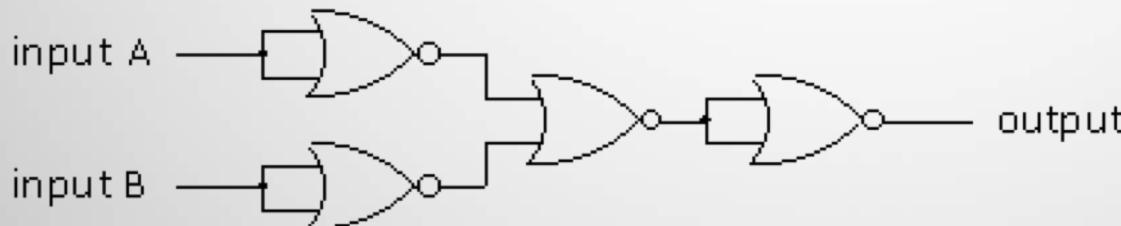
OR gate (NOR followed by NOT)



AND gate (NOT of each input followed by NOR)

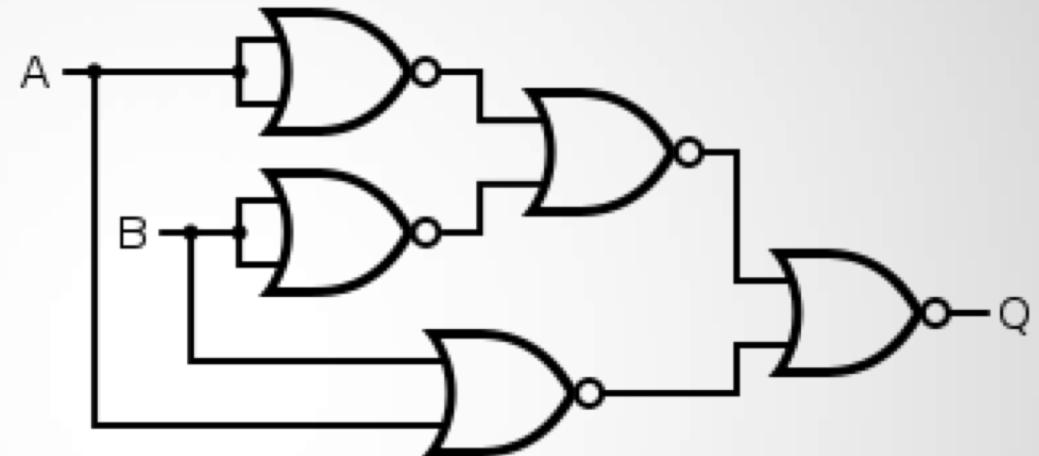


NAND gate (AND followed by NOT)

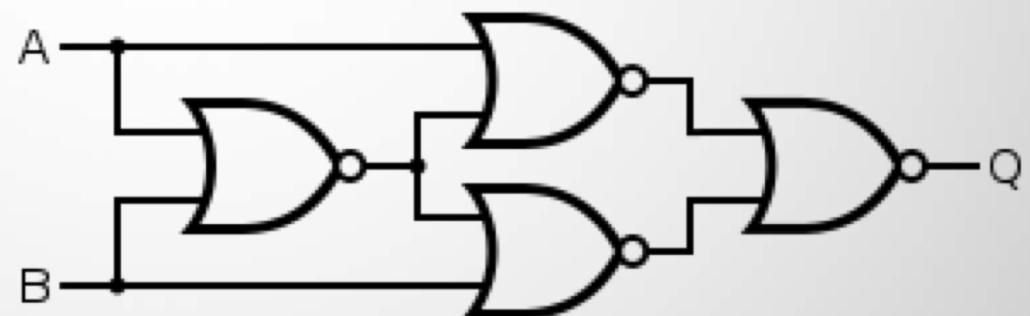


Universality of NOR gates

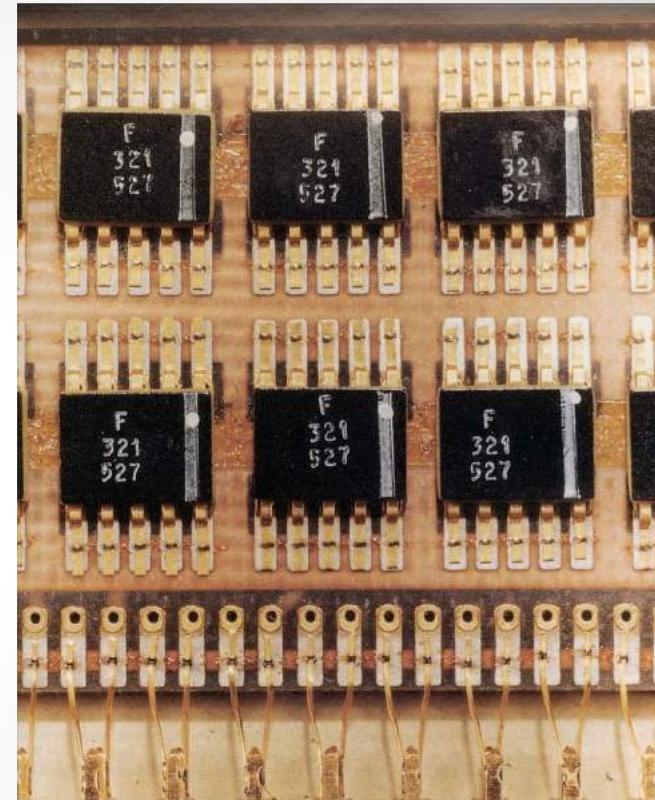
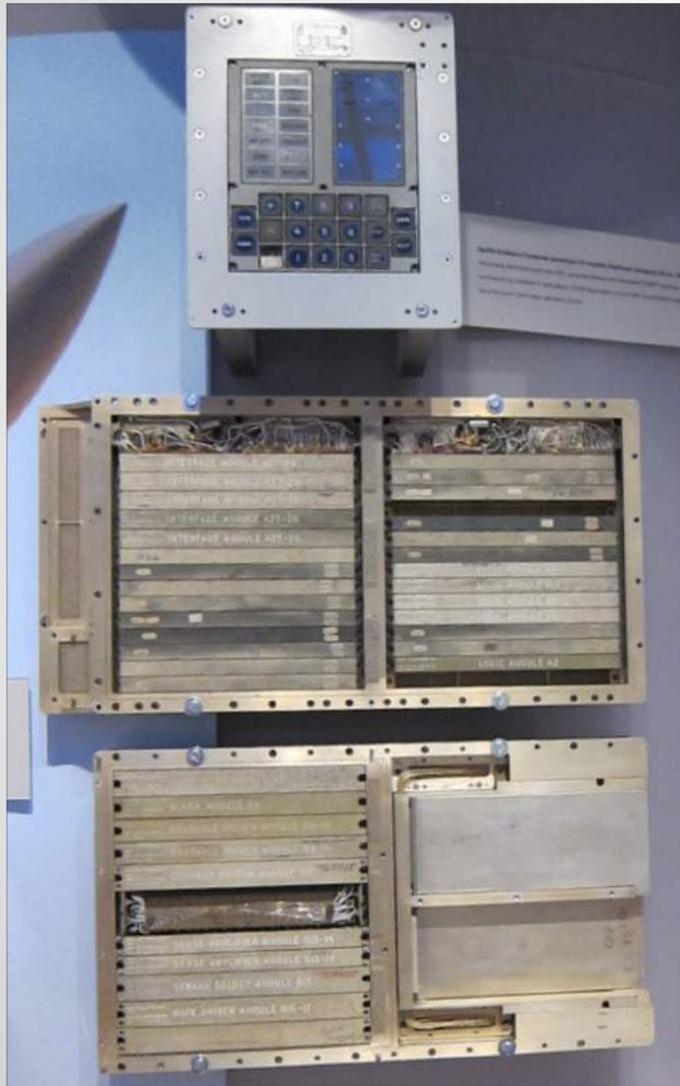
XOR



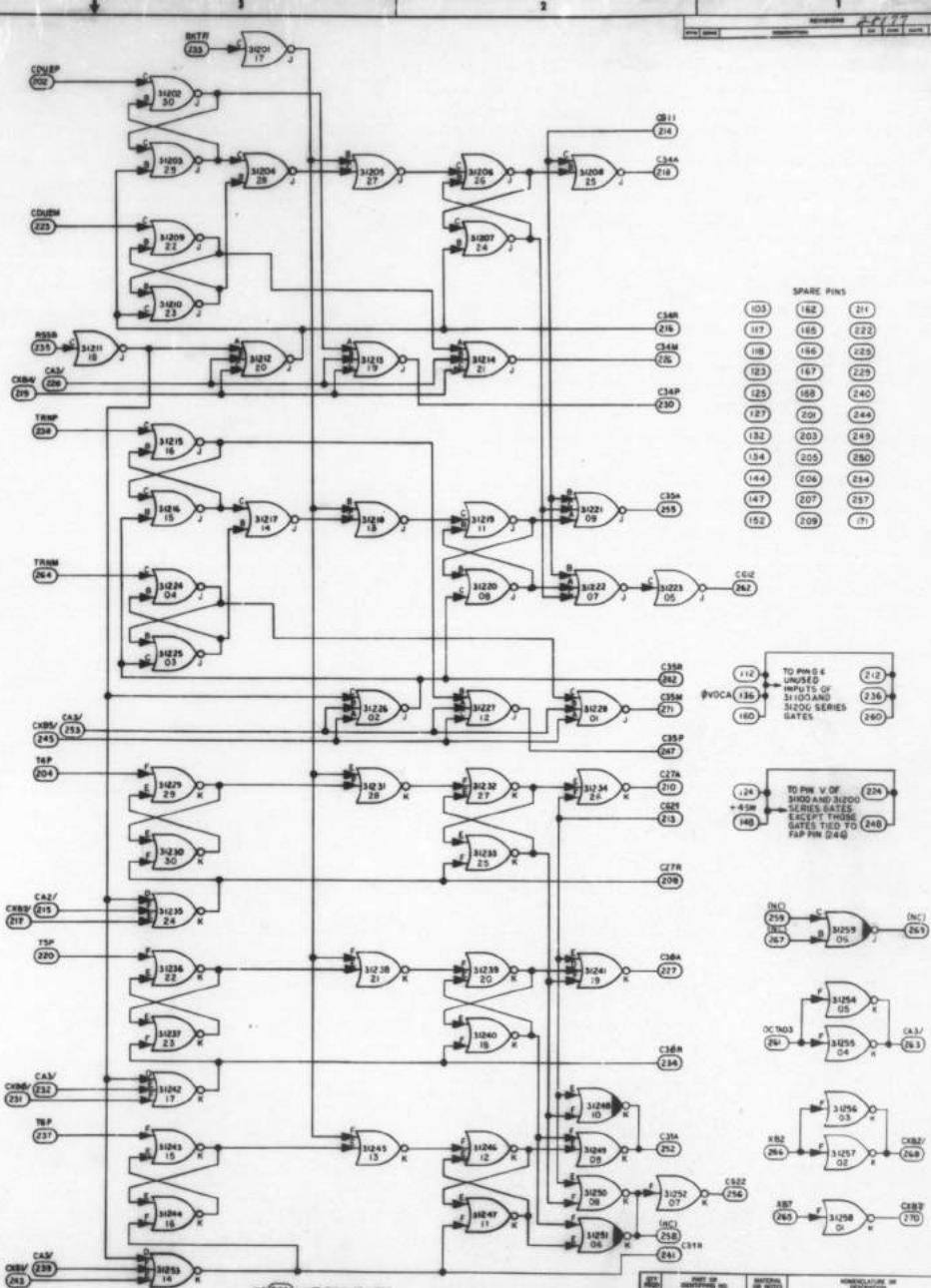
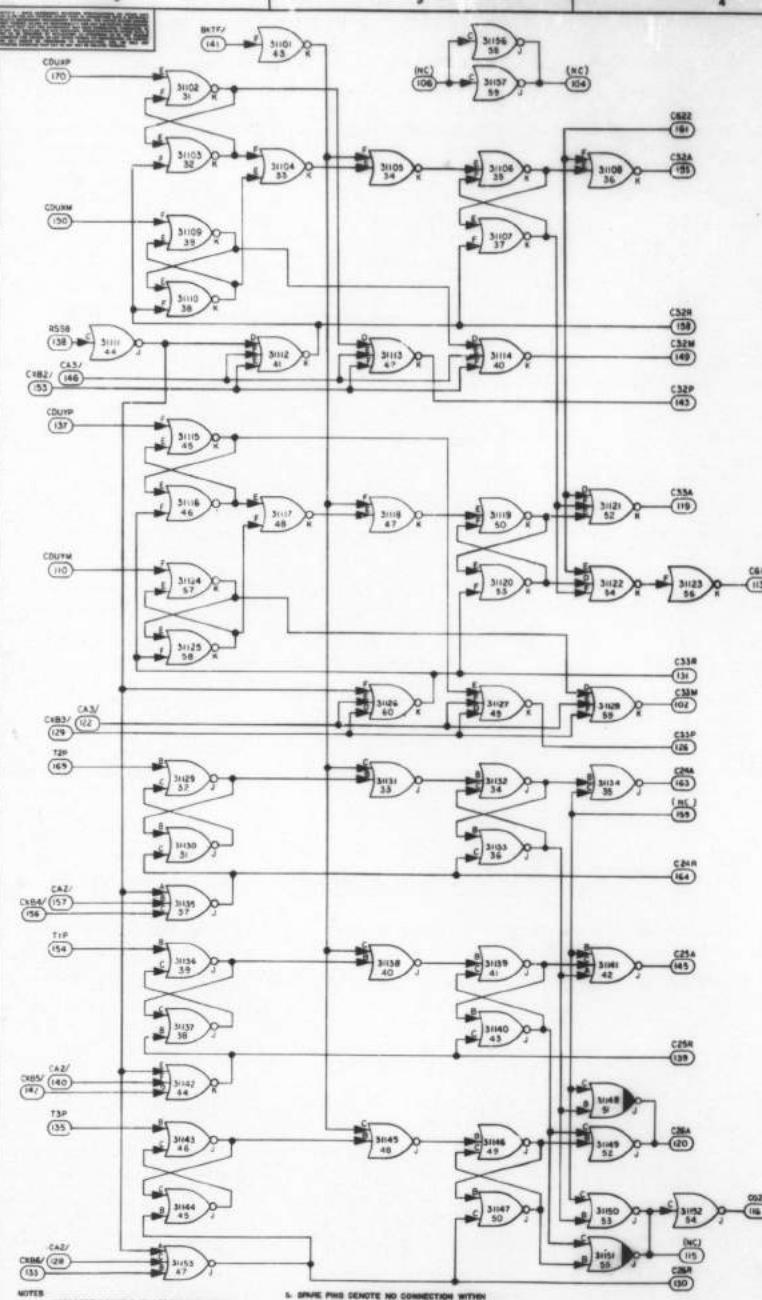
XNOR



NOT/NOR-gate exclusive circuitry



DSKY and AGC on display
at the Computer History
Museum ([Wikipedia](#))



NOTES

1. INTERPRET DRAWING IN ACCORDANCE WITH
STANDARDS ENFORCED BY NS-0-30001

STANDARDS PRESCRIBED BY MIL-D-70327

3. SEE QWS NO.2000SH FOR LOGIC DESIGN CRITERIA
4. SYMBOLIC REPRESENTATION OF DUAL

Q. SYMBOLIC REPRESENTATION OF DUAL NOR GATE

DEFIN

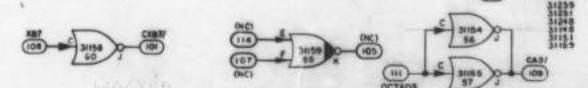
5. SPARE PINS DENOTE NO CONNECTION WITH THE MODULE FOR EXTERNAL CONNECTION

INFORMATION SEE COMPUTER WIRE LIST

6.  UPPER NUMBER DENOTES GATE
LOWER NUMBER DENOTES PHYSICAL
LOCATION OF GATE ON CIRCUIT
MULTILEVEL LOGIC IDENTITY

MULTILAYER SCD HD-1006386

SYMBOL FOR EXPANDER DUAL
S50 HQ.10005384

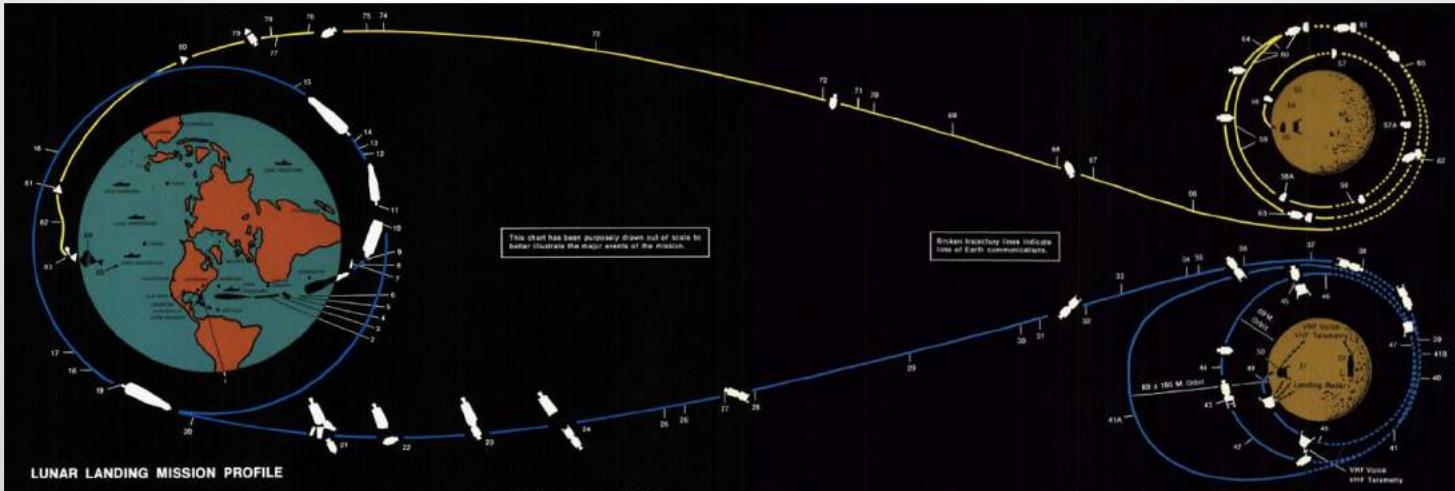


UNLESS OTHERWISE STATED DIMENSIONS ARE IN INCHES DEGREE VALUES ARE IN DEGREES PREFERRED VALUES ARE IN DRAFTS		LIT. BY RELEASER	
		MIT RESEARCH LAB CAMBRIDGE MASS.	
		MANAGED SPACESHIP CENTER HOUSTON, TEXAS	
		 APPROVED FOR RELEASE TO THE GENERAL PUBLIC BY NAME TITLE DATE 1968	
		LOGIC FLOW DIAGRAM MODULE NO A20 COUNTER CELL I	
PRINTED	USED ON	PRINTED NO	ISSUED NO
APPLICATION		80230	E 2005254

NOT/NOR-gate exclusive circuitry



Apollo Moon Mission Circuits



Apollo Guidance System = 5600 gates

http://klabs.org/history/ech/agc_schematics/



NASA Office of Logic Design

A scientific study of the problems of digital engineering for space flight systems,
with a view to their practical solution.

APOLLO GUIDANCE COMPUTER (AGC) Schematics

The documents below, courtesy of [Eldon C. Hall](#), are schematics for the Block II Apollo Guidance Computer.

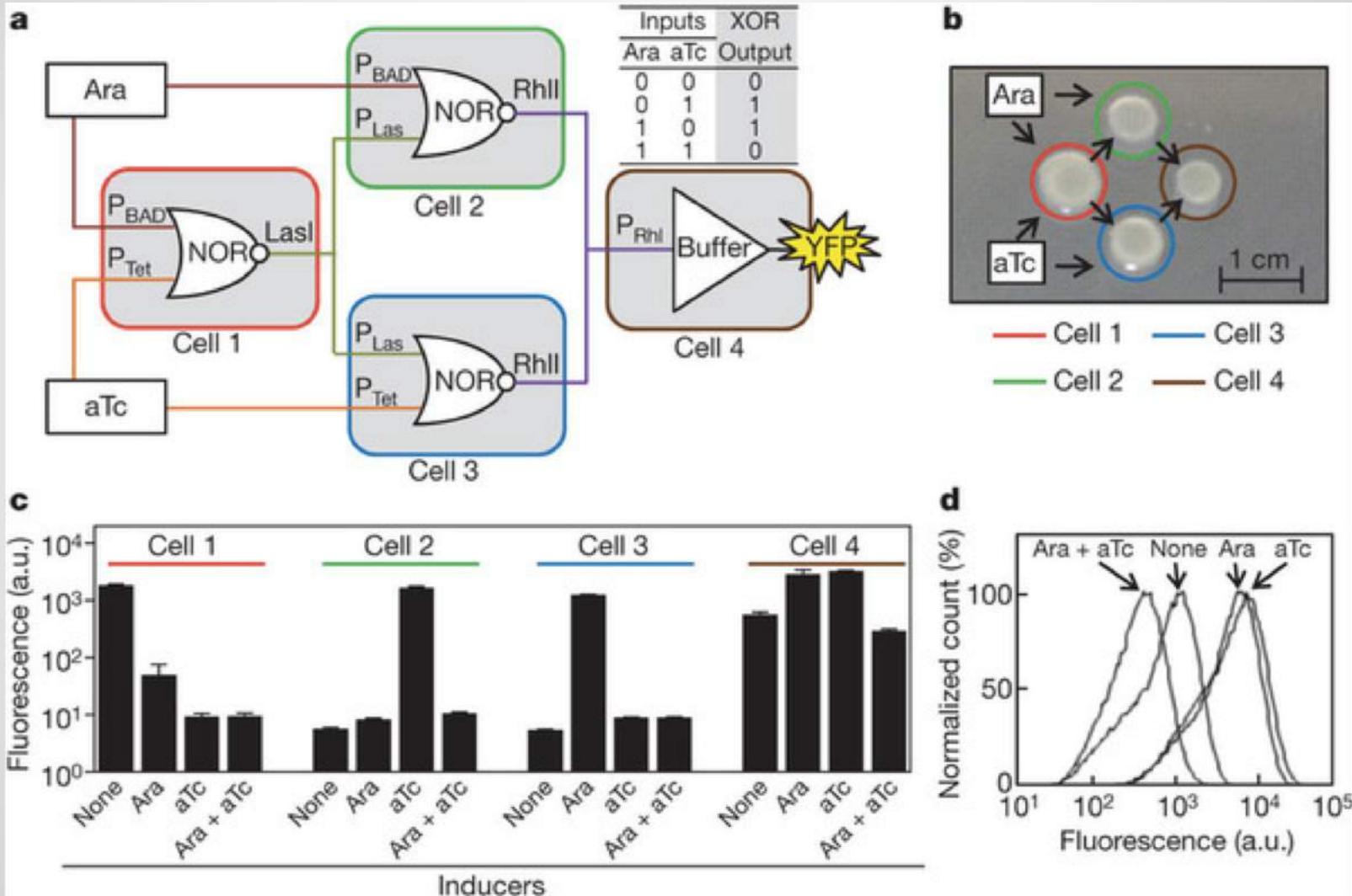
Please see related documents on the [AGC Integrated Circuits and Reliability Page](#). Detailed examination of the problems associated with the packages can be found in those reports, along with die related issues. Additionally, a partial [analysis](#) of Apollo devices has been conducted. The [AGC NOR Gate Specifications](#) goes along with this page. Additional [documents](#) related to and about the AGC are also available. Additionally, the [Apollo Guidance Computer and Other Computer History](#) will be of interest to readers of this page.

INDEX TO DRAWINGS

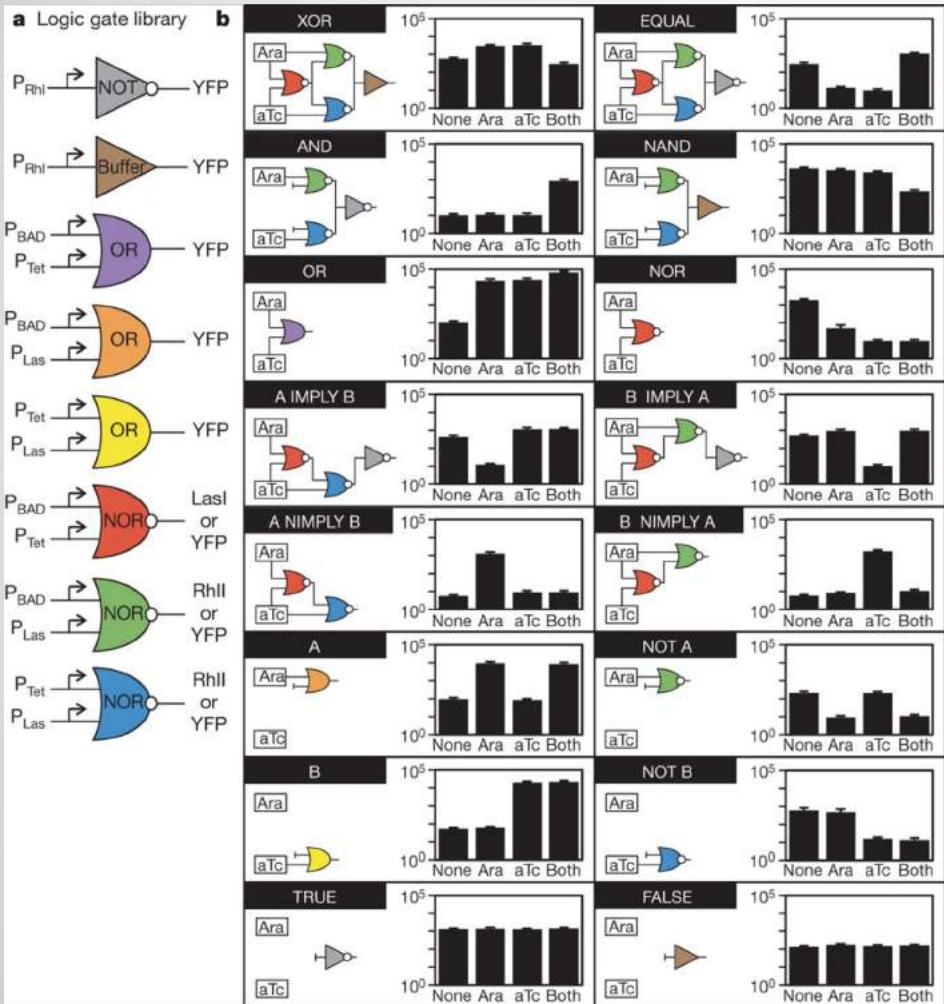
- [INTERFACE A25-A26](#)
- [INTERFACE A27-A29](#)
- [INTERFACE A30-A31](#)
- [CLOCK OSCILLATOR](#)
- [POWER SUPPLY](#)
- [DUAL NOR GATE](#)
- [RESTART MONITOR](#)
- [SCALER MODULE](#)
- [TIMER](#)
- [S-Q REGISTER AND DECODING](#)
- [STAGE BRANCH DECODING MODULE](#)
- [CROSS POINT GENERATOR NOI](#)
- [CROSS POINT GENERATOR II](#)
- [SERVICE GATES](#)
- [COUNTER CELL I](#)
- [COUNTER CELL II](#)
- [4 BIT MODULE \(1 OF 4\)](#)
- [4 BIT MODULE \(2 OF 4\)](#)
- [4 BIT MODULE \(3 OF 4\)](#)
- [4 BIT MODULE \(4 OF 4\)](#)
- [PARITY AND S REGISTER](#)
- [ALARMS](#)
- [MEMORY TIMING & ADDRESSING](#)
- [RUPT SERVICE](#)
- [INOUT I](#)
- [INOUT II](#)
- [INOUT III](#)
- [INOUT IV](#)
- [INOUT V](#)
- [INOUT VI](#)
- [INOUT VII](#)



Bacterial NOR Gates - Multicellular



Bacterial NOR Gates - Multicellular

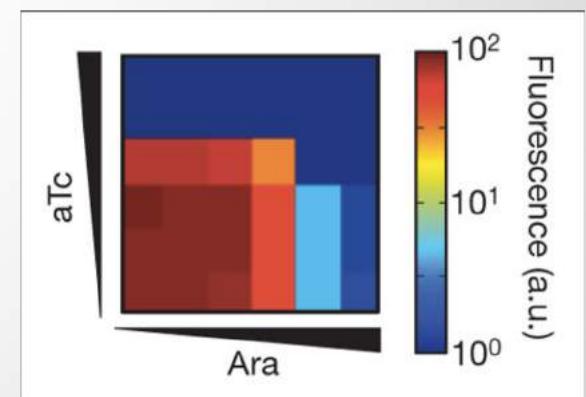
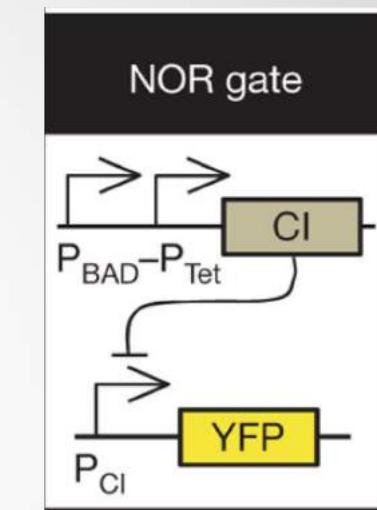
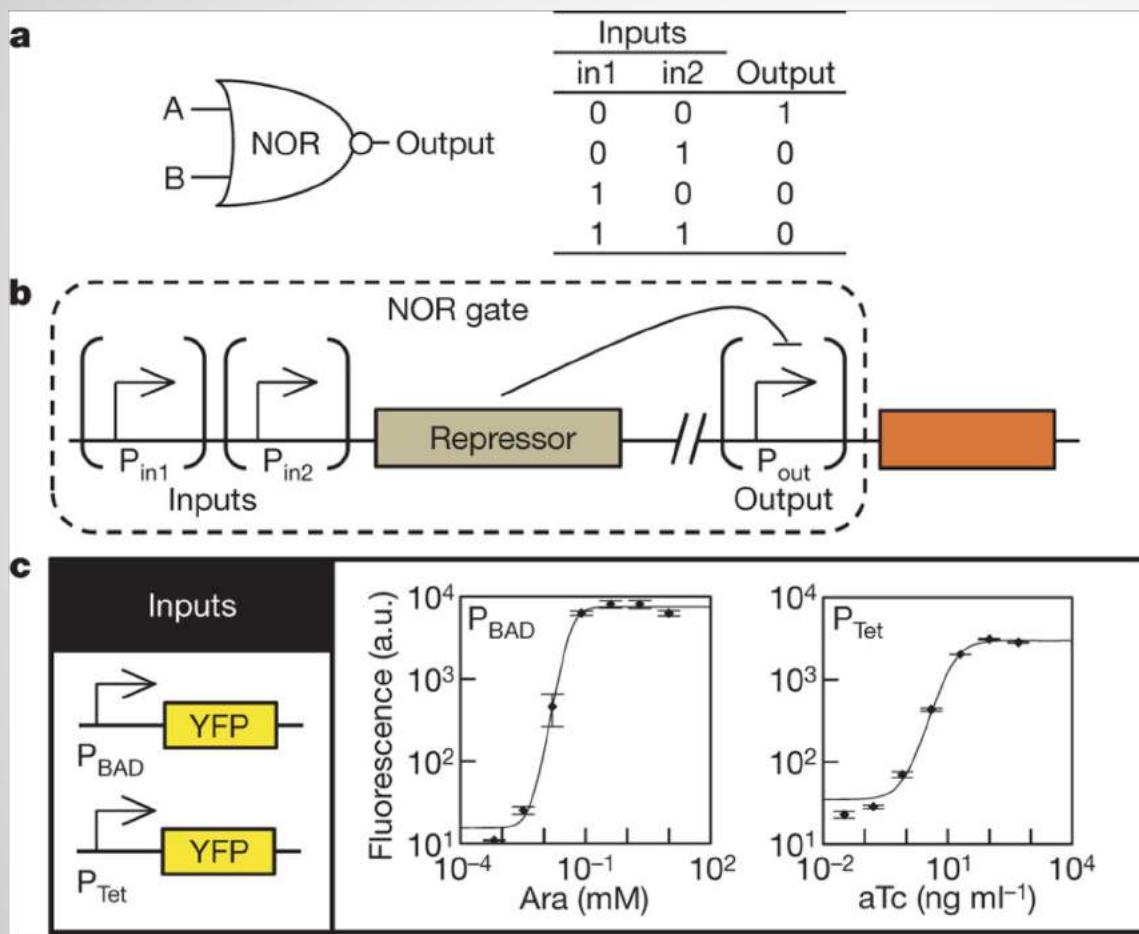


Each cell is a NOR gate with different input/outputs

Wires are diffusible signals

Plating 4 colonies in different places defines the logic

Bacterial NOR Gates from NOT Gates



2016: Automated Design of Complex Logic Functions... all inside an *E. coli*

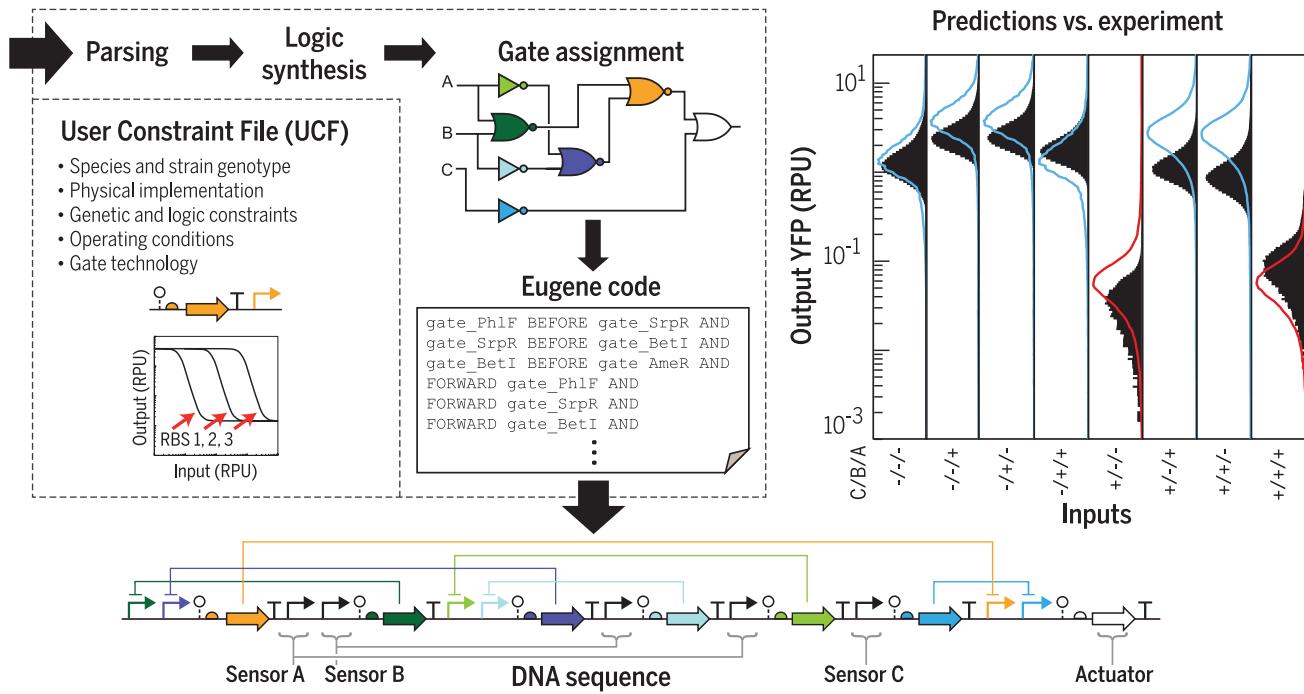
Cello design specification

Sensors			
name	low	high	promoter sequence
A	0.003	2.8	AACGATCGTGGCTGTGTTGACAATT
B	0.001	4.4	TACTCCACCGTGGCTTTTCCTA
C	0.008	2.5	ACTTTCATACTCCCCCATTAGAG

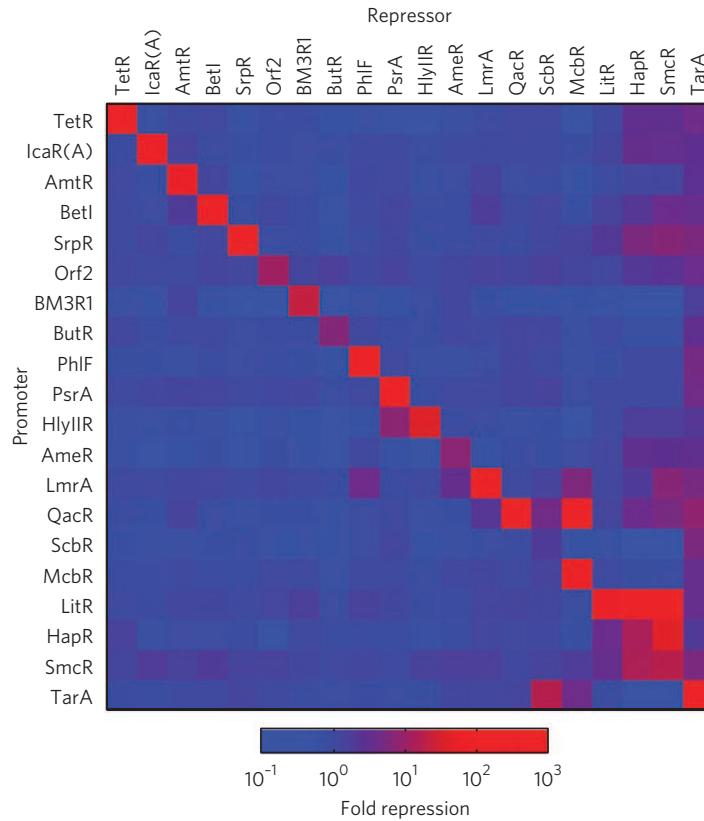
Verilog			
module OxF6(output out, input A,B,C); always@{C,B,A) begin case({C,B,A}) 3'b000: {out} = 1'b1; 3'b001: {out} = 1'b1; 3'b010: {out} = 1'b1; 3'b011: {out} = 1'b1; 3'b100: {out} = 1'b0; 3'b101: {out} = 1'b1; 3'b110: {out} = 1'b1; 3'b111: {out} = 1'b0; endcase end endmodule			

Actuators			
name	sequence		
YFP	ATGGTGAGCAAGGGCGAGGAGCTGTTACCGGGGT		

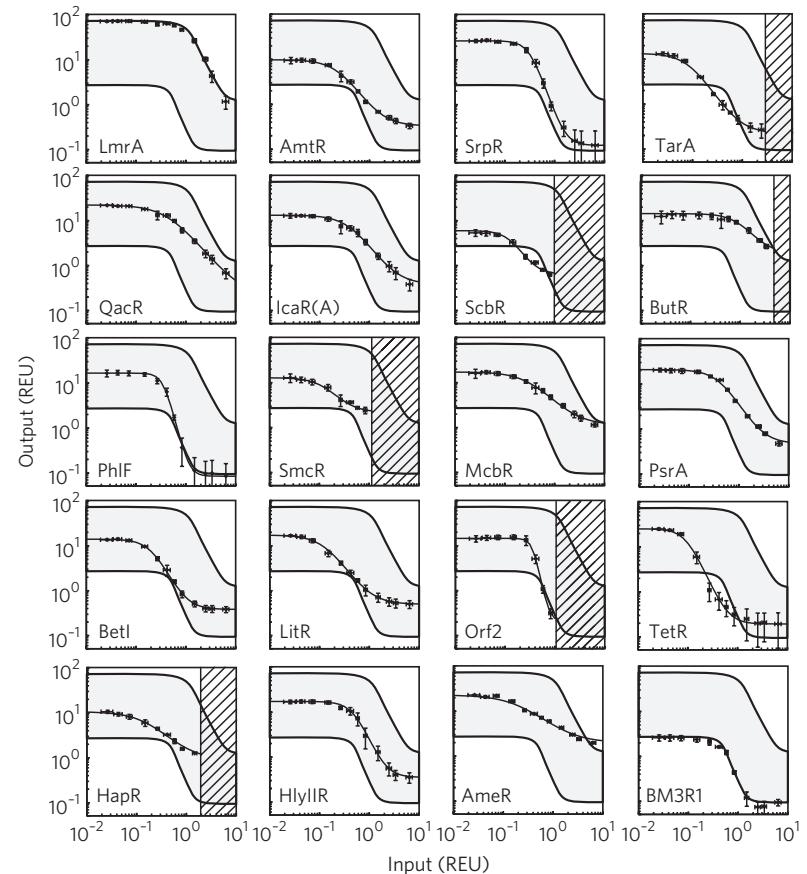
Run



Best logic in *E. coli* = 16 NOT gates

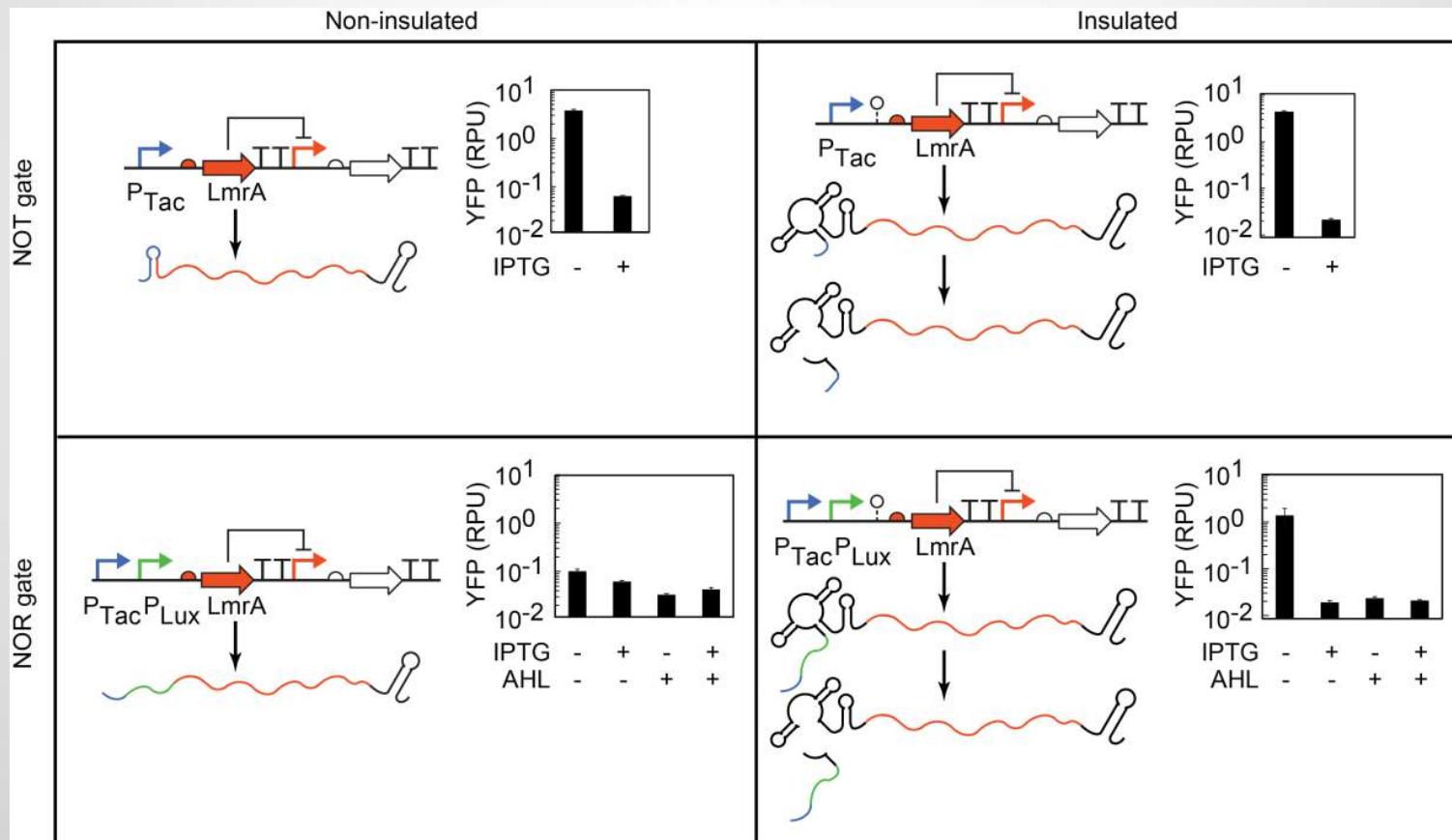


Stanton et al. Nature Chem Biol 2014



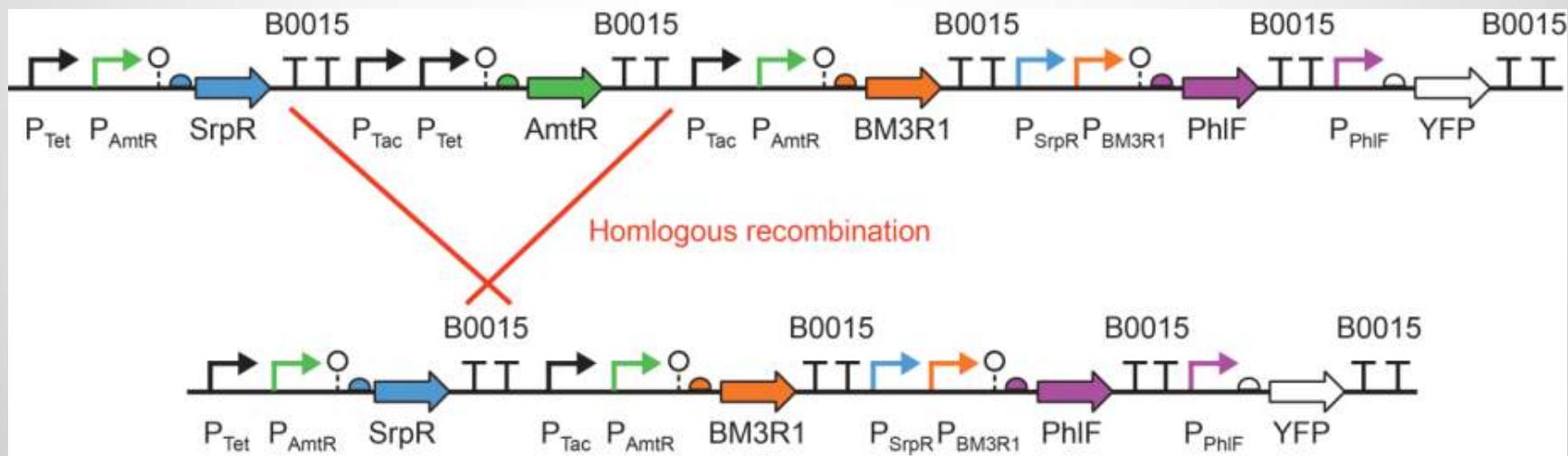
16 NOT gates = 256 different NORs

DNA circuit has to be built with Ribozyme insulators so that behaviour with one promoter is the same as with two



Prevent failures: insulate & don't repeat parts

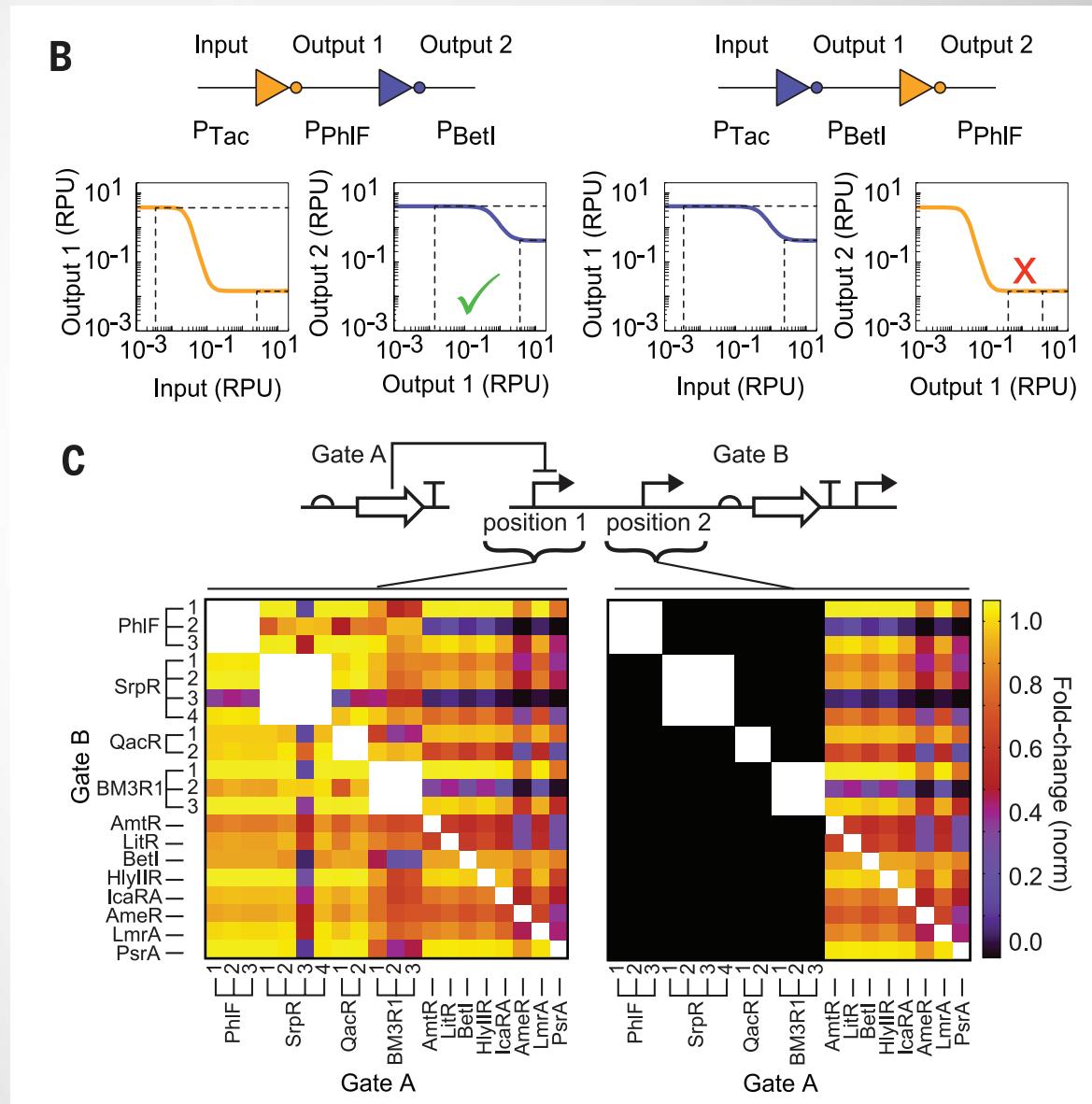
- If a plasmid has a part repeat within it, it will ‘recombine’ and delete out the DNA between
- Need many different versions of each part
- Use their terminator library
- Made a new RiboJ-type insulator library



Some NOR gates are better than others

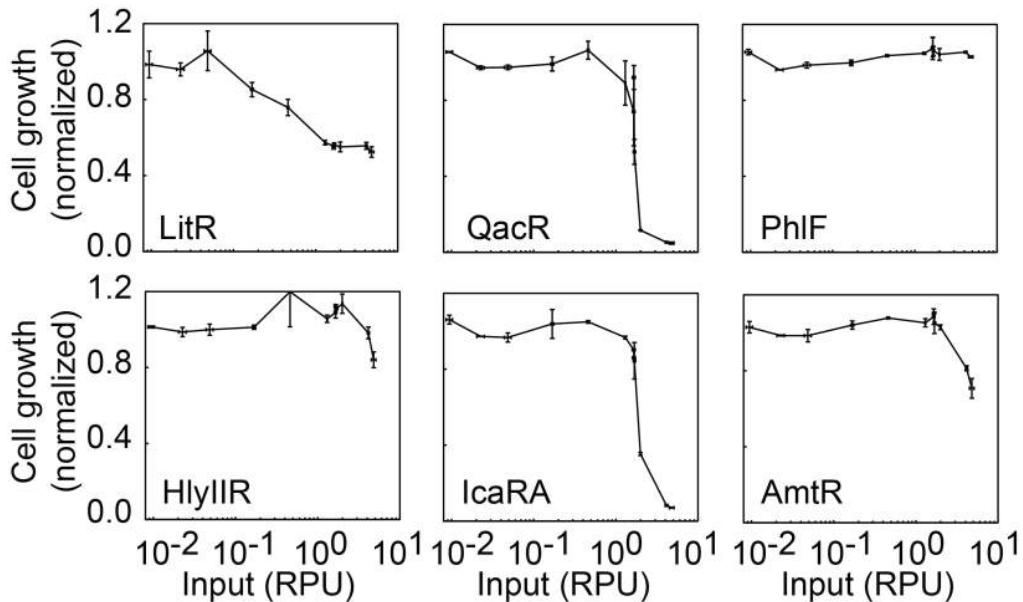
Range of outputs of one gate needs to match with the other gate's input range

Testing many combinations reveals ones with the best ranges and those not to use

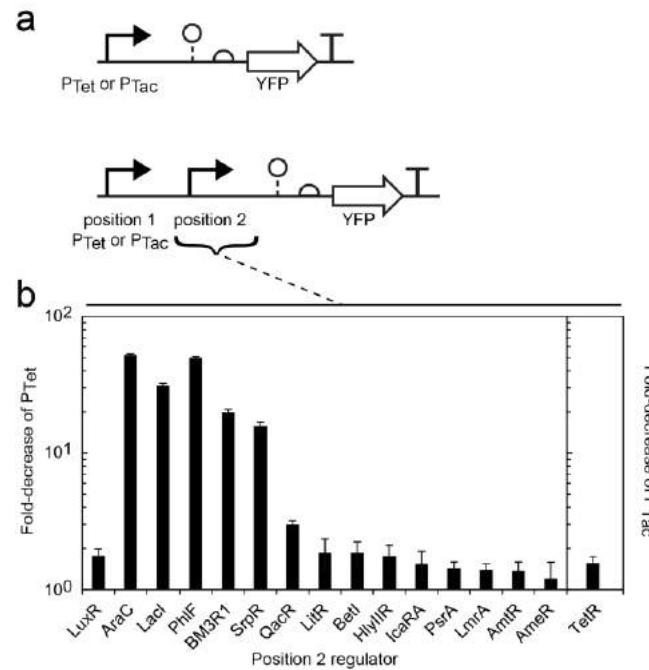


Some combinations are terrible to use

Toxic Transcription Factors

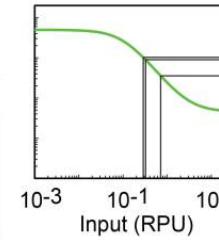
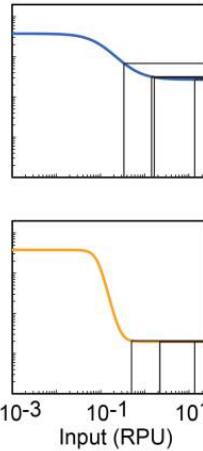
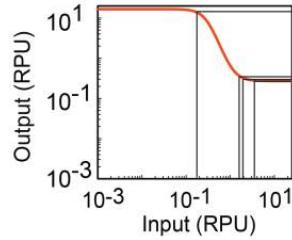
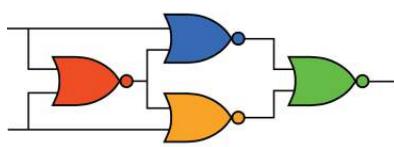


Roadblock Promoters



Insulating and choosing best parts works

a

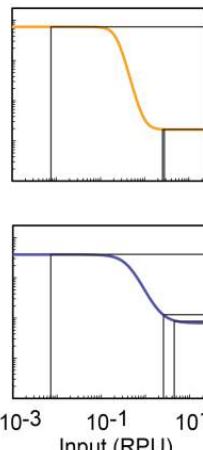
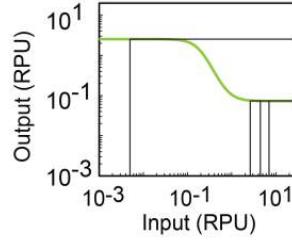
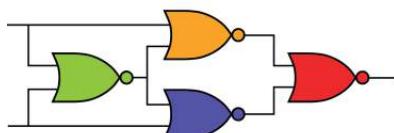


uninsulated

XNOR

Inputs
1 0 0 1

b

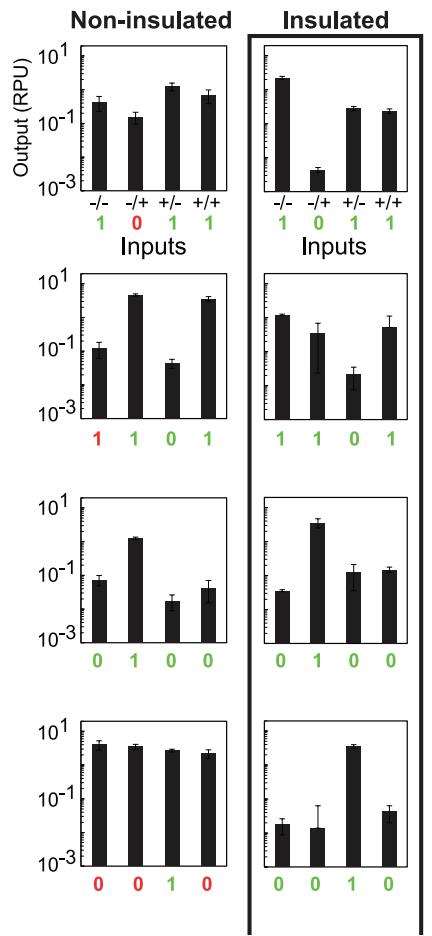
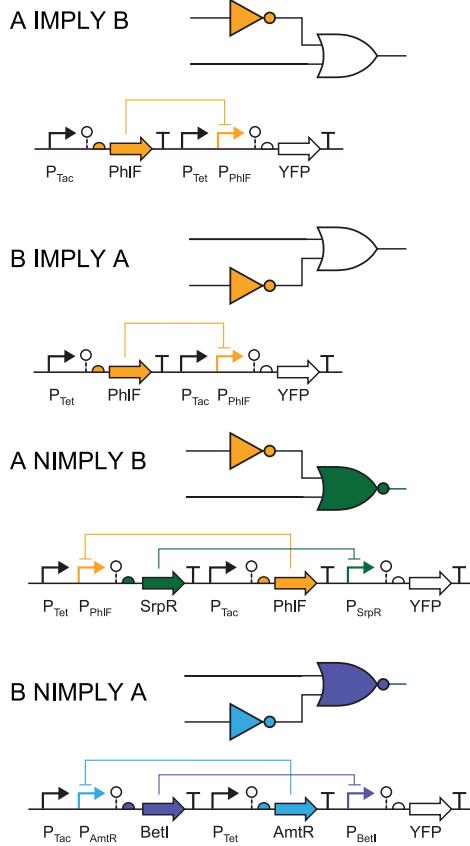


insulated

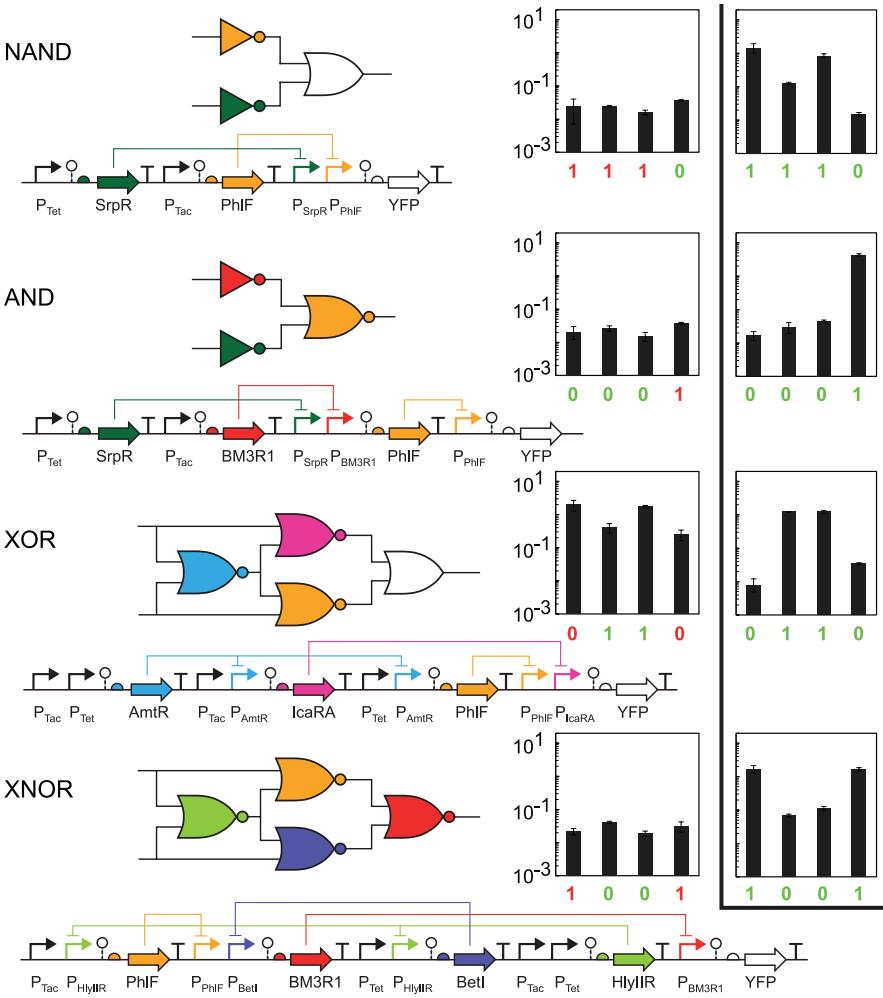
Inputs
1 0 0 1

Insulating and choosing best parts works

A



NAND



Design can now be rational and automatic

<http://www.cellocad.org/>

A Cello design specification

Sensors	name	low	high	promoter sequence
PTac	0.003	2.8		AACGATCGTTGGCTGTGACAATTAAATCATC
PTet	0.001	4.4		TACTCCACCGTGGCTTTTCCCTATCAGTGA
PBAD	0.008	2.5		ACTTTCTATACTCCGCCATTCAAGAGAAGAAC

```
Verilog
module 0x21(output out, input A,B,C);
  always@(C,B,A)
    begin
      case((C,B,A))
        3'b000: {out} = 1'b0;
        3'b001: {out} = 1'b0;
        3'b010: {out} = 1'b1;
        3'b011: {out} = 1'b0;
        3'b100: {out} = 1'b0;
        3'b101: {out} = 1'b0;
        3'b110: {out} = 1'b0;
        3'b111: {out} = 1'b1;
      endcase
    end
endmodule
```

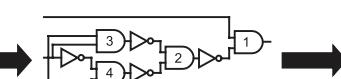
Run

Verilog parsing

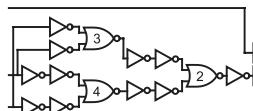
CBA	w1	w2	w3	w4	X
000	0	1	0	1	0
001	0	0	1	1	0
010	0	1	0	0	1
011	0	0	1	0	0
100	0	0	1	1	0
101	1	0	0	1	0
110	0	0	1	0	0
111	1	0	0	0	1

Logic synthesis

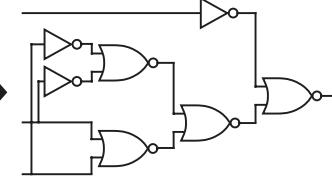
AND-Inverter



NOR/NOT



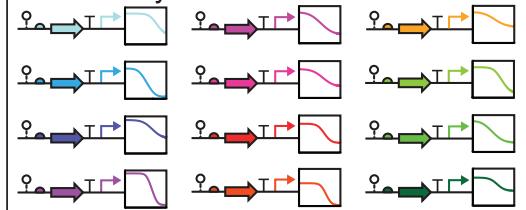
Final circuit diagram



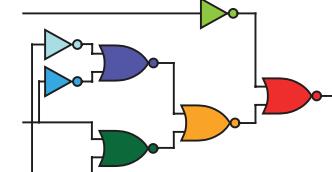
User Constraint File (UCF)

- Gate technology
- Physical implementation
- Genetic and logic constraints
- Strain genotype
- Operating conditions

Gate library



Assign gates



Combinatorial design



Genetic circuit DNA sequence

Verilog: design language for hardware

What is Verilog?

<https://www.youtube.com/watch?v=qIQwC3YIHG0>

Cello: Verilog for *E. coli* logic circuit design

CelloCAD

<http://www.cellocad.org/>

Cello Verilog Options Results About You are logged in as Teeeee



CELLO

Help Page	Information and instructions for using the cellocad.org web application.
Video	A video tutorial for using the cellocad.org web application.
Supp Info	Supporting Information PDF.
API	Endpoints, requests, and responses of Cello API. Submit jobs and retrieve results without downloading/installing.
CLI	A command-line interface facilitates use of the Cello API.
GitHub repo	Clone the repository to get the source code.
Readme	(GitHub md) Overview of genetic circuit design using Cello.
Install	(GitHub md) How to install and compile.
Run	(GitHub md) How to run, with demo files. Includes web app, API, CLI, and Java instructions.

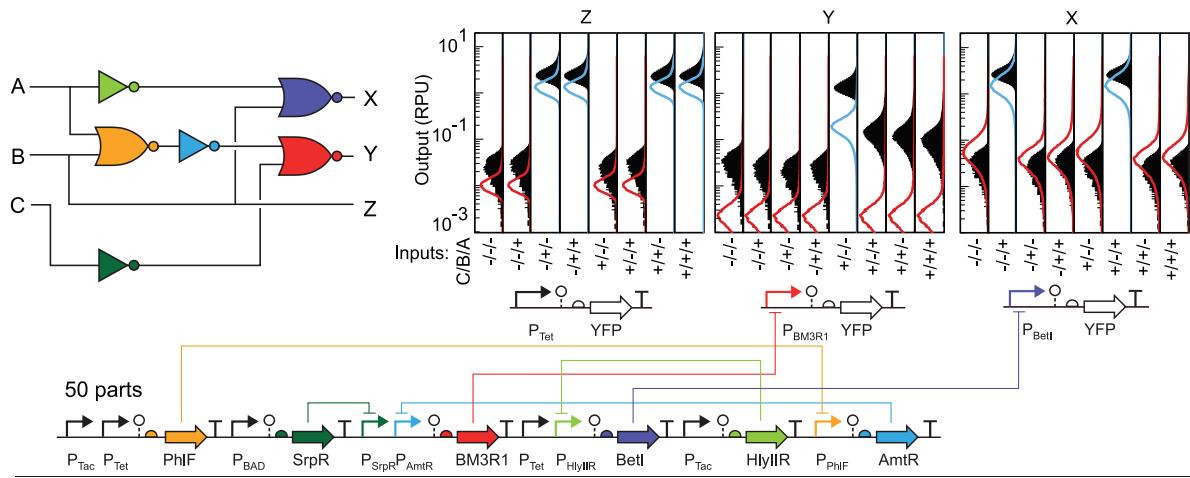
Cello: Verilog for *E. coli* logic circuit design

A

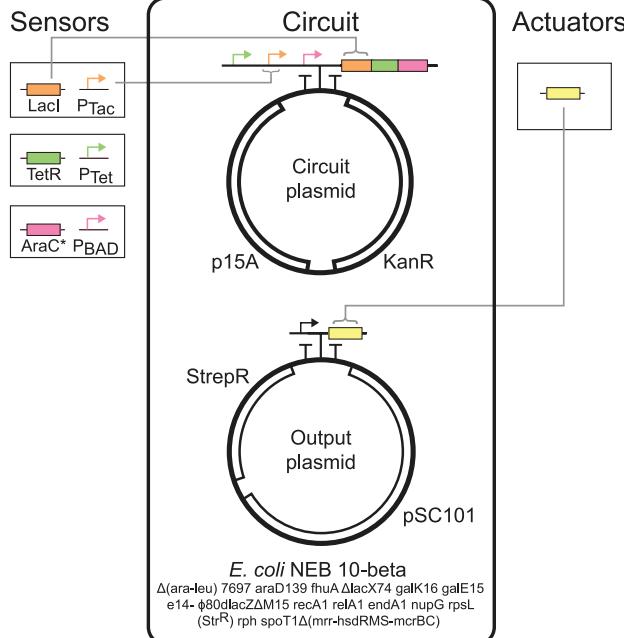
Priority detector

```
module priority_detector(output outZ, outY, outX, input
C, B, A);
always@(C,B,A)
begin
  case((C,B,A))
    3'b000: {outZ,outY,outX} = 3'b000;
    3'b001: {outZ,outY,outX} = 3'b001;
    3'b010: {outZ,outY,outX} = 3'b100;
    3'b011: {outZ,outY,outX} = 3'b100;
    3'b100: {outZ,outY,outX} = 3'b010;
    3'b101: {outZ,outY,outX} = 3'b001;
    3'b110: {outZ,outY,outX} = 3'b100;
    3'b111: {outZ,outY,outX} = 3'b100;
  endcase
end
endmodule
```

input	low	high
A.IPTG	0.003	2.8
B.aTc	0.001	4.4
C.Ara	0.008	2.5



B



- Inputs are sensors (for inducers)
- Outputs are fluorescent proteins or actuator transcription factors
- Actuators can be regulators that change *E. coli* behaviours

Cello: Inputs and Coding

The screenshot shows the Cello software interface. At the top, there is a navigation bar with tabs: Cello, Verilog, Options, Results, and About. To the right of the tabs, it says "You are log" and "powered by Screencastify Lite".

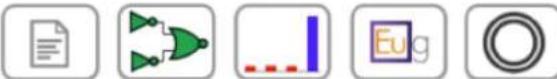
The main area is divided into sections:

- Verilog:** A dropdown menu set to "choose". Below it is a code editor window containing the number "1".
- Inputs:** A dropdown menu set to "choose". Below it are four buttons: index, name, low RPU, high RPU, and DNA sequence.
- Outputs:** A dropdown menu set to "choose". Below it are three buttons: index, name, and DNA sequence.
- Buttons:** A row of buttons at the bottom left: "design name", "Run" (highlighted in green), and other standard application buttons like back, forward, and close.

Cello: Results

Cello Verilog Options Results About

You are logged in powered by  **Screencastify Lite**



Design Name: multiplexer [Delete](#) [Download Zip](#)

input Verilog

```
module multiplexer(output out1,  input in1, in2, in3);
  always@(in1,in2,in3)
    begin
      case({in1,in2,in3})
        3'b000: {out1} = 1'b0;
        3'b001: {out1} = 1'b0;
        3'b010: {out1} = 1'b0;
        3'b011: {out1} = 1'b1;
        3'b100: {out1} = 1'b1;
        3'b101: {out1} = 1'b1;
        3'b110: {out1} = 1'b0;
        3'b111: {out1} = 1'b1;
      endcase
    end
endmodule
```

output log file

```
.....
```



Cello: Possibilities

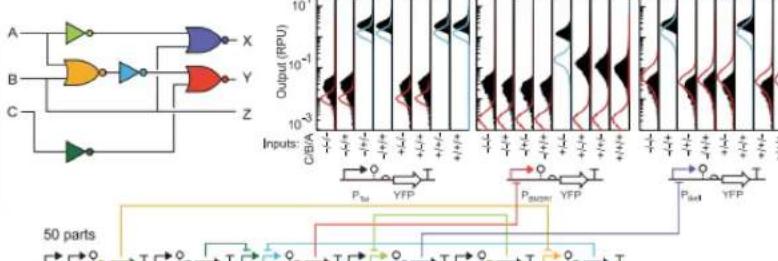
powered by
Screencastify Lite

RESEARCH | RESEARCH ARTICLE

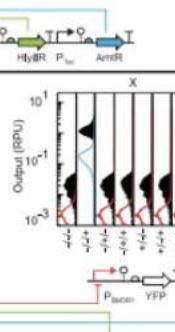
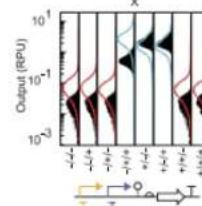
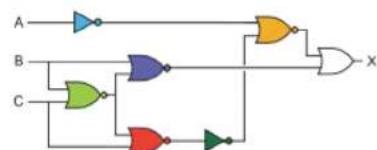
A

Priority detector

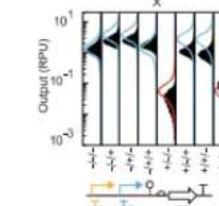
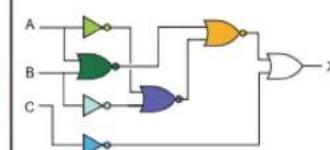
```
module priority_detector(output outZ, outY, outX, input
C, B, A)
begin
  case({C,B,A})
    3'b000: {outZ,outY,outX} = 3'b000;
    3'b001: {outZ,outY,outX} = 3'b001;
    3'b010: {outZ,outY,outX} = 3'b011;
    3'b011: {outZ,outY,outX} = 3'b101;
    3'b100: {outZ,outY,outX} = 3'b010;
    3'b101: {outZ,outY,outX} = 3'b000;
    3'b110: {outZ,outY,outX} = 3'b001;
    3'b111: {outZ,outY,outX} = 3'b100;
  endcase
end
endmodule
```



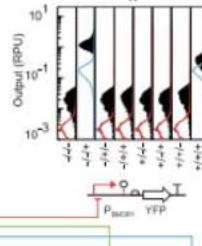
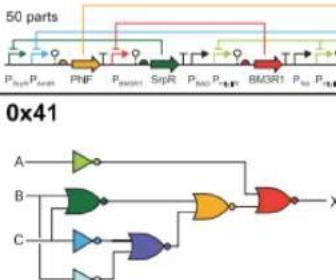
0x1C



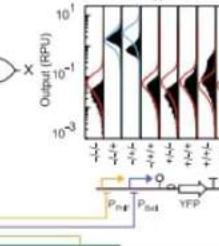
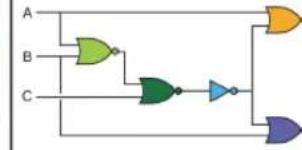
0xF6



0x41

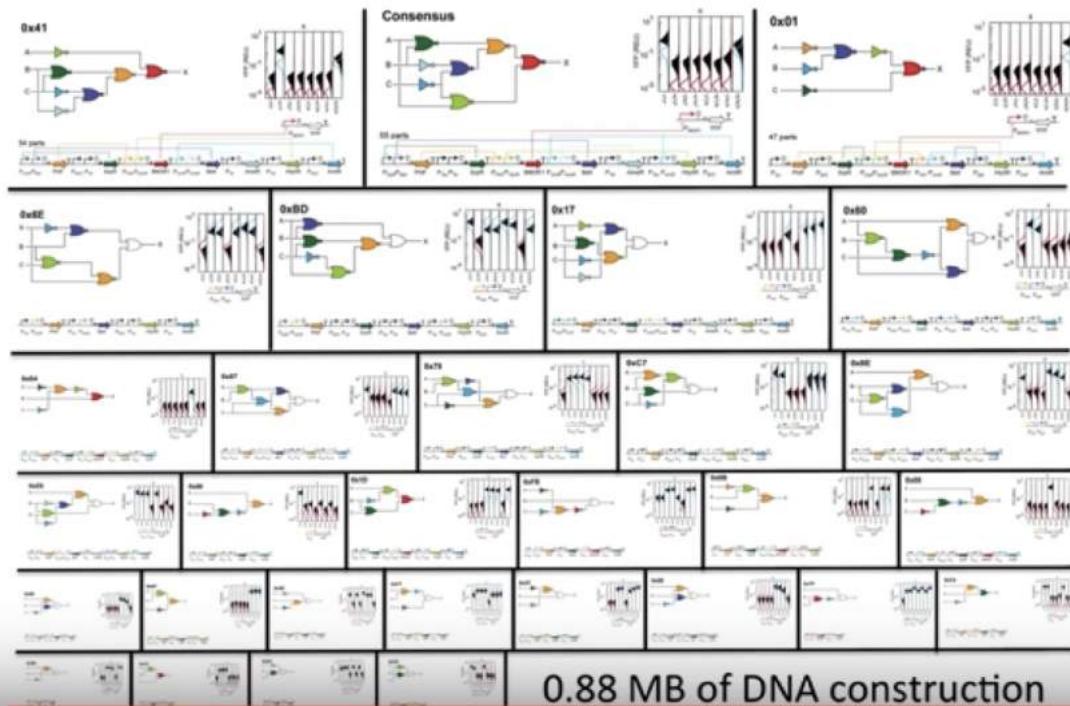


0x60



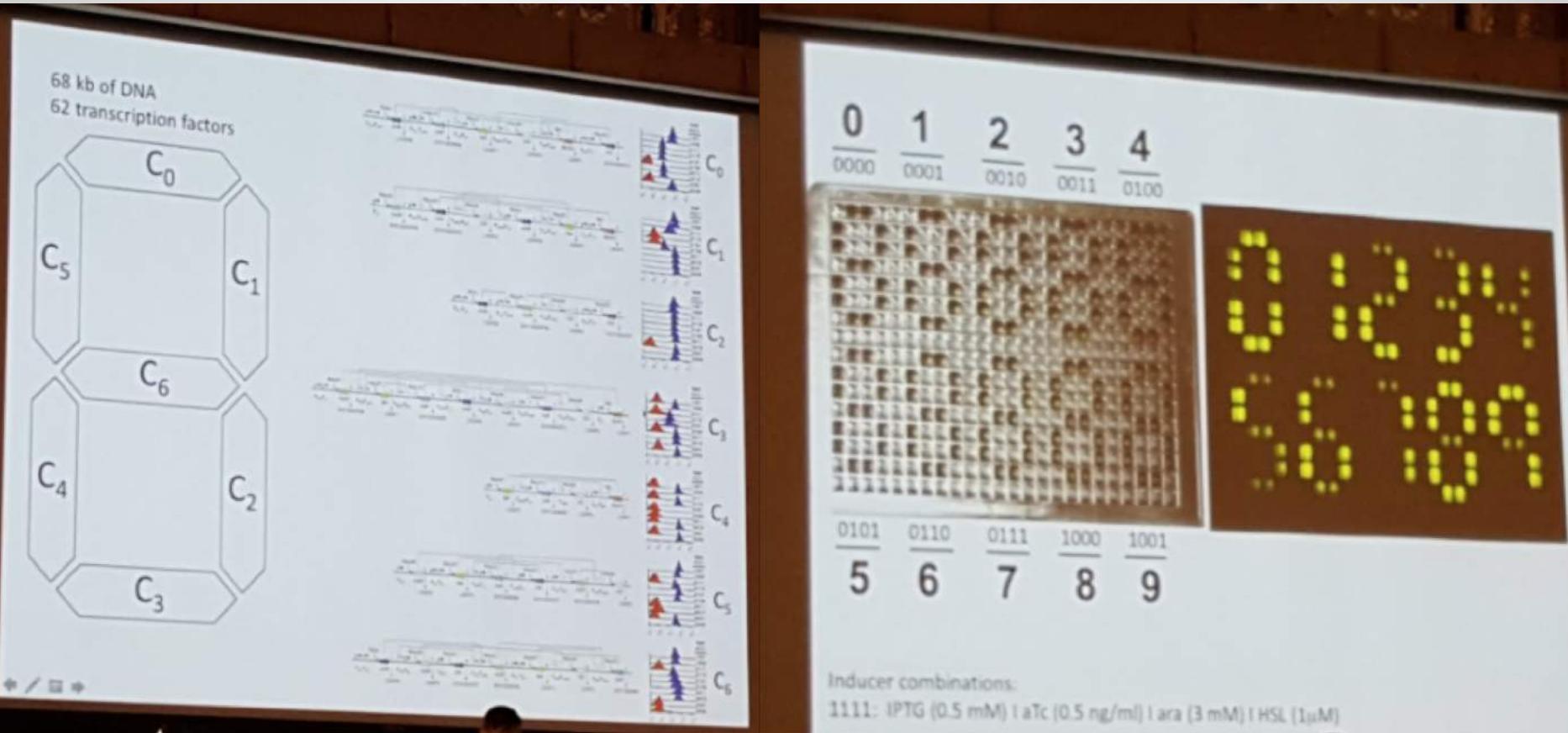
Automatable Circuit Design Done!

Many circuits tested...



<https://www.youtube.com/watch?v=INttxYdGHs4>

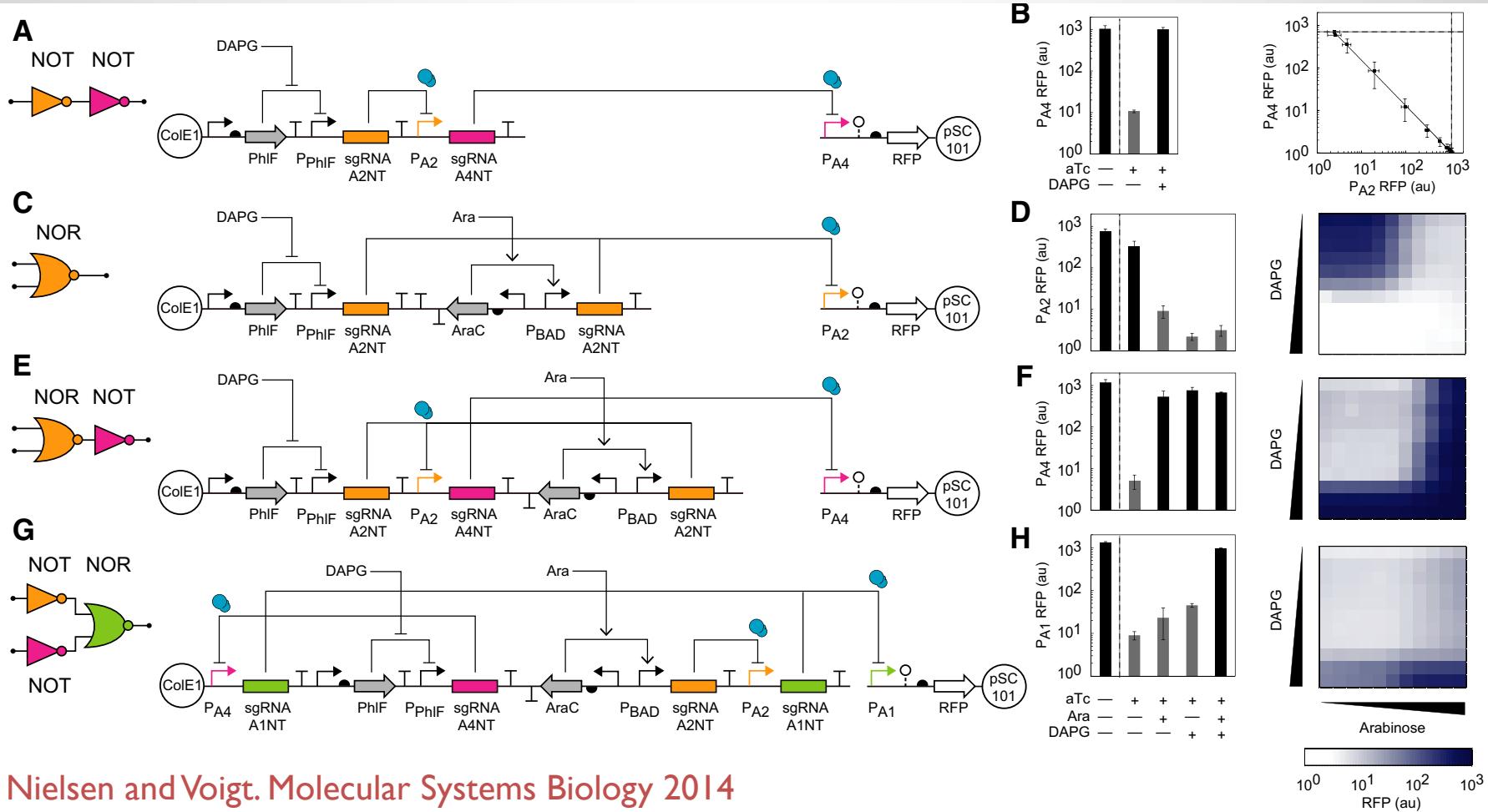
What they're doing with Cello now...



68 kbp DNA logic for LCD number display

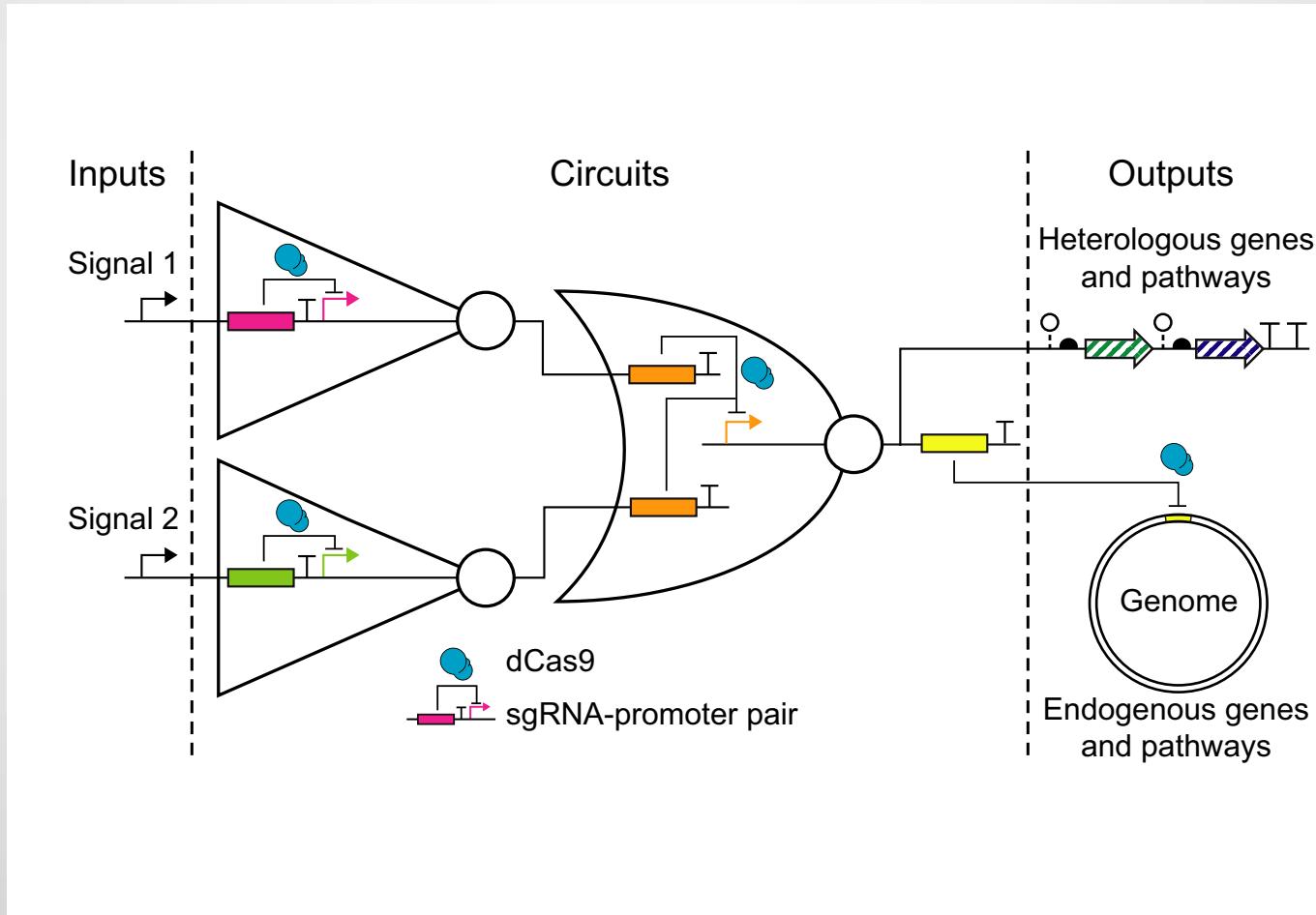
Can also do logic with CRISPRi

CRISPRi regulation can be combined and cascaded to make logic gates



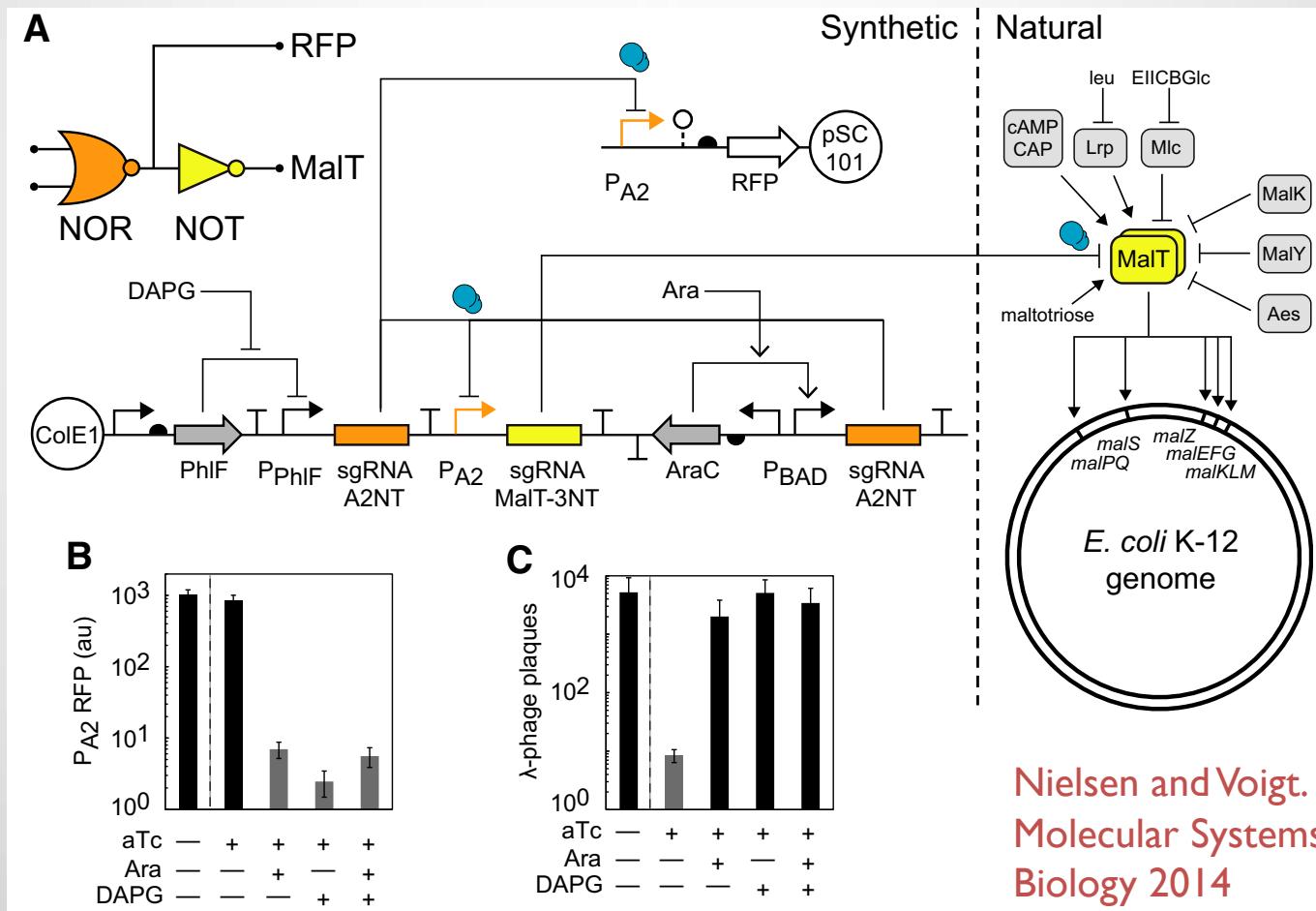
Applying CRISPRi logic in *E. coli*

CRISPRi logic can be used to control synthetic or genomic genes



Applying CRISPRi in *E. coli*

CRISPRi logic can be used to control synthetic or genomic genes

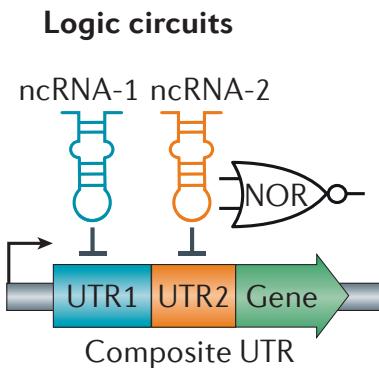


100+ logic gates in a cell?

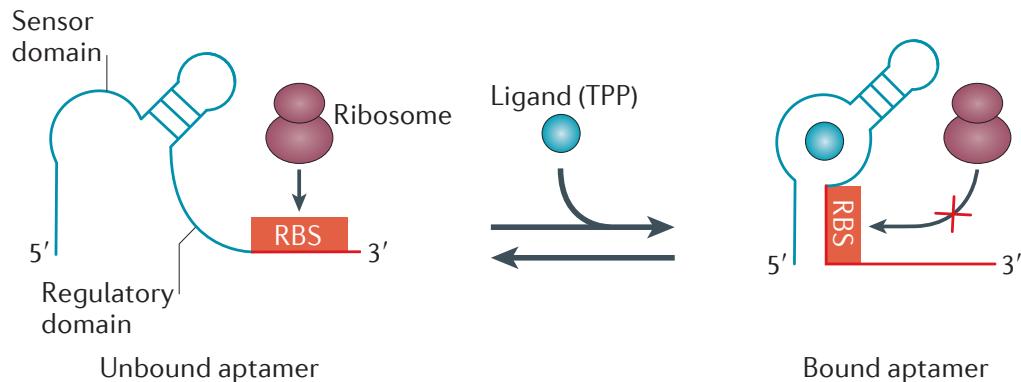
- Scalable parts doesn't fully solve the problem...
- Bacteria are efficient cells optimised for growth and extra DNA programs are a '**burden**' that slows growth and impairs performance
- Two solutions to reduce overloading a cell:
 - I. Use programs that are less 'costly' to the cell
 2. Distribute program among a population:
i.e. multicellular genetic networks

Less ‘costly’ programs: RNA devices

- DNA→RNA is much less work for a cell than DNA→RNA→Protein (translation is a major cost)
- RNA parts can substitute for protein regulators and sensors
- e.g. Riboregulators, Riboswitches (and CRISPRi)



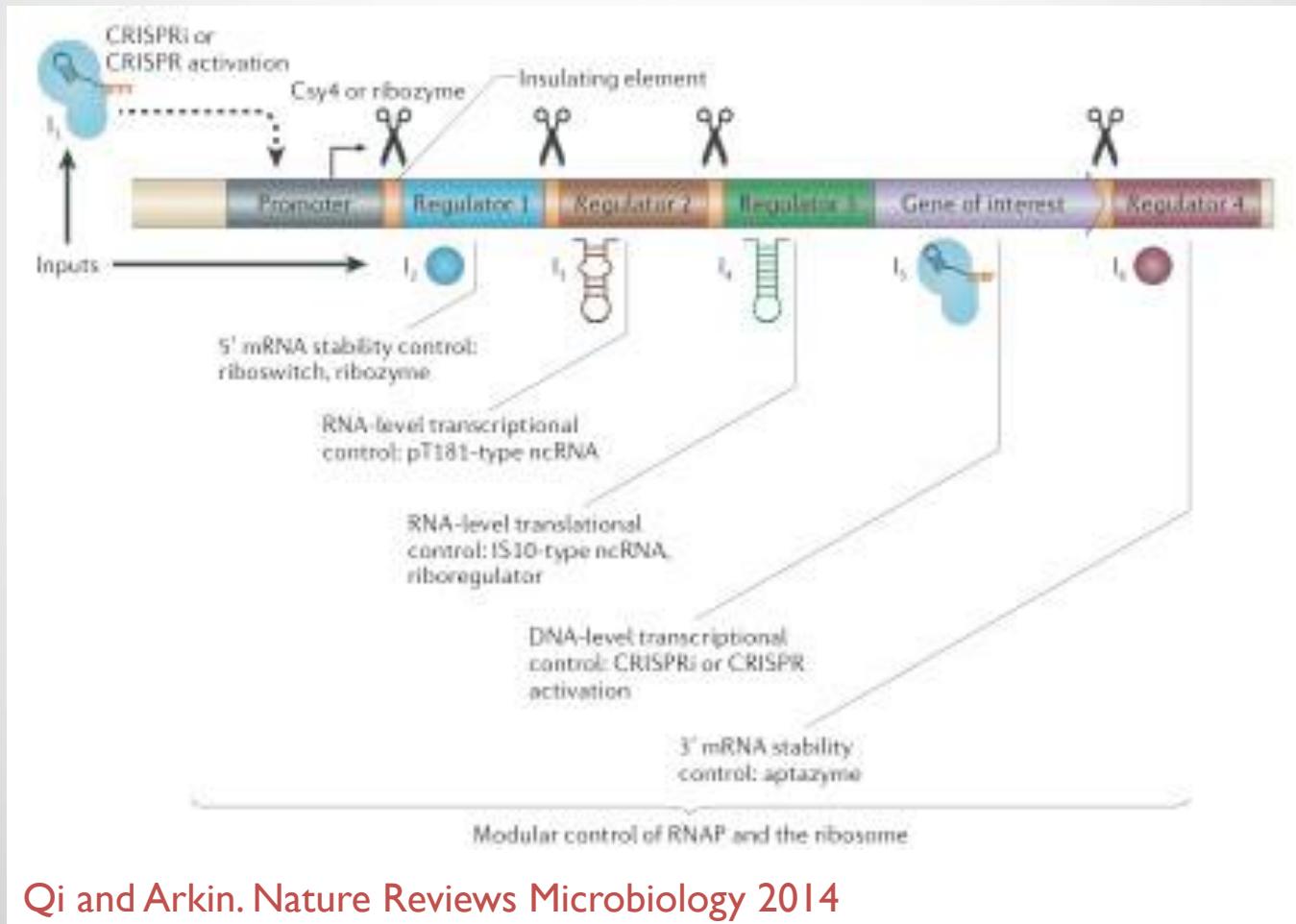
Riboregulators



Riboswitch (also known as an aptamer)

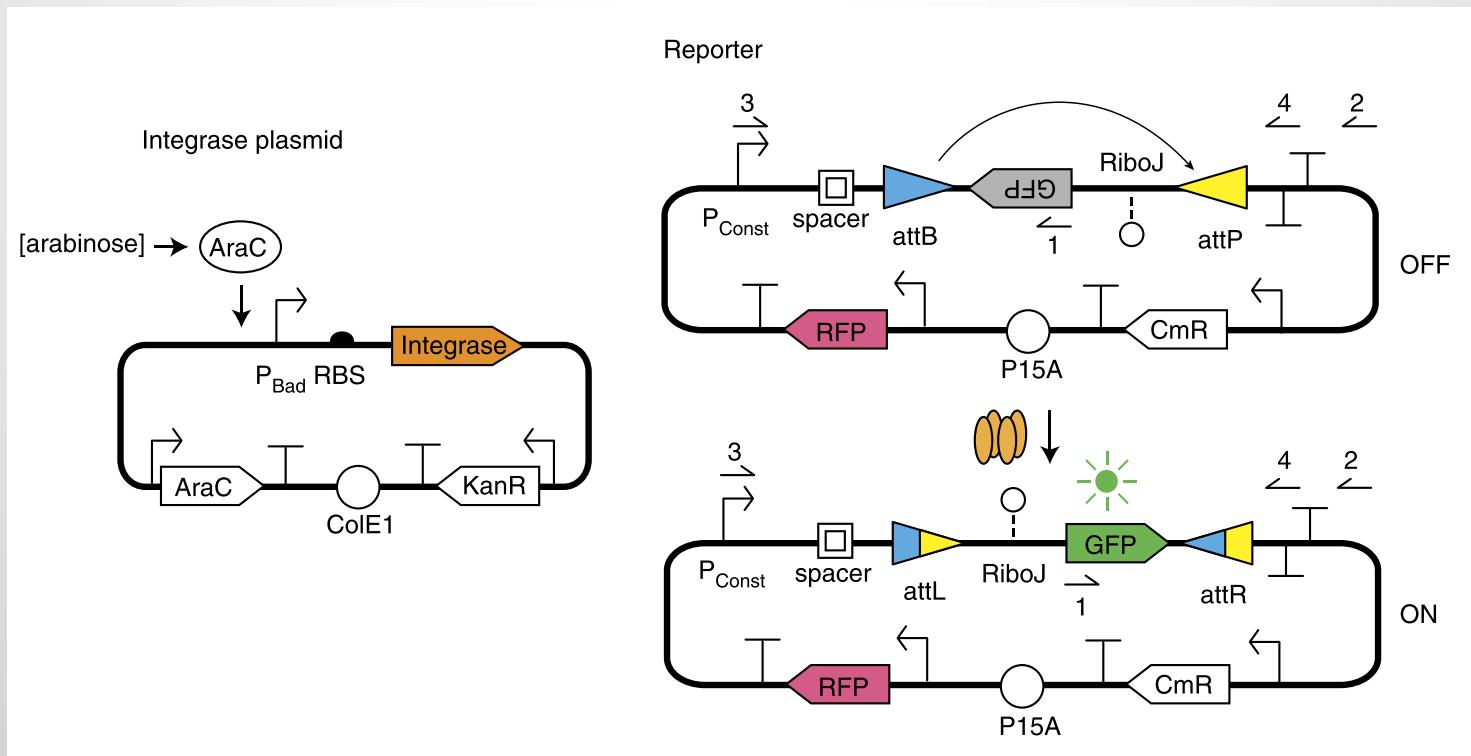
Less ‘costly’ programs: RNA devices

Many different ways to tune gene expression with RNA



Less ‘costly’ programs: DNA switches

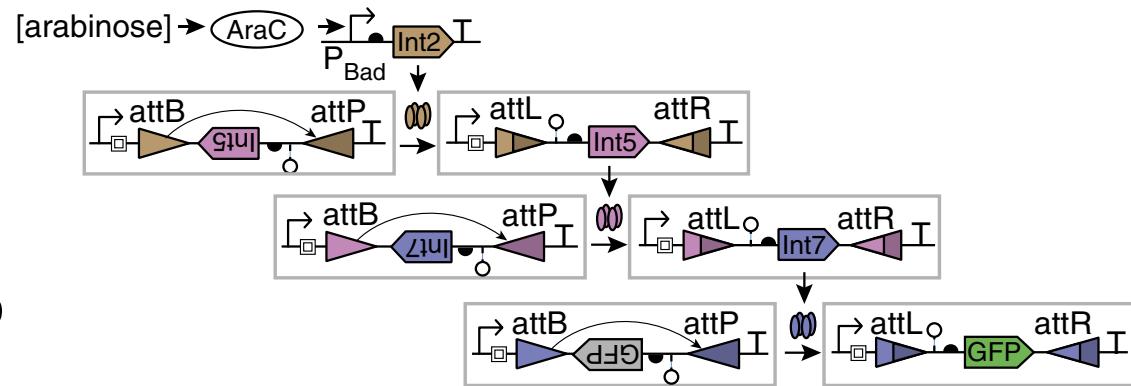
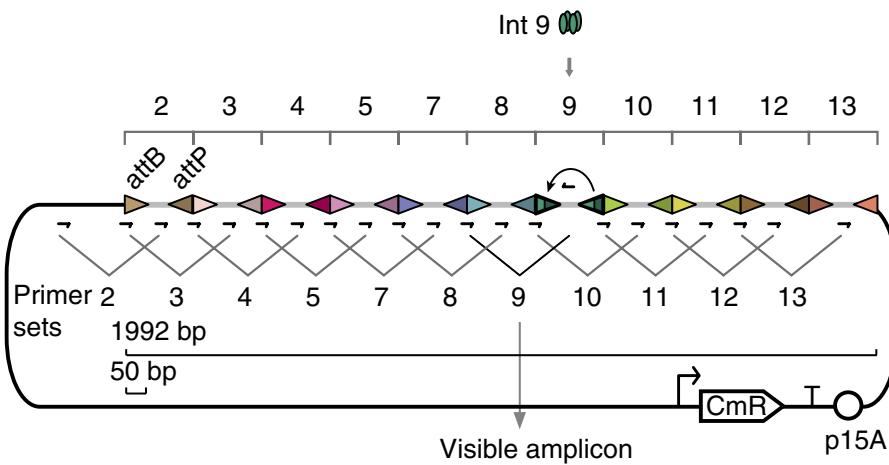
- DNA recombination can be used to write memory physically into the DNA sequence
- DNA region inverts when an *integrase* is induced



Less ‘costly’ programs: DNA switches

With 11 different integrases you can encode >1 byte of digital information (2^{11} different events)

Nesting of integrase genes between recombination sites



Final layout of DNA is a record of what inducers came in which order

Yang, Nielsen et al. Nature Methods 2014

Summary

- Bacterial logic is possible at promoters
- NOR gate logic in multicellular systems and now also within *E. coli*
- Logic scales if well-characterised and insulated
- Cello means design is fully-automated.
- CRISPRi is a game-changer for regulation
- As things get more complex, non-protein parts are needed to allow lower-cost

Example Exam Questions

- Give examples of logic functions implemented in *E. coli* by synthetic biologists
- How does Cello work?
- Outline the need for insulators and characterisation for automating logic circuit design?
- Describe the challenges of making a 12-gate logic network in *E. coli*

Key References

Original Research

Genetic Circuit Design Automation

Nielsen *et al.* Science 2016

Robust multicellular computing using genetically encoded NOR gates and chemical ‘wires’

Tamsir, Tabor and Voigt Nature 2011

CRISPRi Logic Gates

Nielsen and Voigt. Molecular Systems Biology 2014

Further References

Principles of Gene Circuit Design (REVIEW)

Brophy and Voigt. *Nature Methods* 2014

RNA-based parts and devices (REVIEW)

Qi and Arkin. *Nature Reviews Microbiology* 2014

Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology

Wang *et al.* *Nature Communications* 2011

DNA recombinases for >1 byte of memory

Yang, Nielsen *et al.* *Nature Methods* 2014