Supplementary materials for:

Genetic circuit design automation

Alec A.K. Nielsen, Bryan S. Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A. Strychalski, David Ross, Douglas Densmore, and Christopher A. Voigt

I. Design and characterization of insulated gates 3
I.A. *Insulators of promoter context: design of ribozymes and spacers 3*
I.B. *Terminator selection for transcriptional insulation 6*
I.C. *RBS selection to tune the response threshold 7*
I.D. *Response functions* equation *and cytometry data for insulated gates 8*

II. Simple gate combinations 10
II.A. *Non-insulated gates: predicted and measured outputs 10*
II.B. *Characterization of error modes 11*
II.C. *Insulated gates: predicted and measured outputs 15*

III. Complete circuit data 17
III.A. *Circuits with 1 failed output state 17*
III.B. *Circuits with 2 failed output states 18*
III.C. *Circuits with ≥3 failed output states 18*
III.D. *Majority circuit variants 19*
III.E. *Alternative repressor assignments 21*
III.F. *Replicates of circuit library 22*

IV. Debugging genetic circuits 24

V. Cello Software 26
IV.A. *Specification: Verilog hardware description language 27*
IV.B. *Parsing Verilog to generate a truth table 32*
IV.C. *Logic synthesis 34*
IV.D. *Repressor assignment 36*
IV.E. *Combinatorial design of circuit layouts 39*
IV.F. *Predictions of circuit performance 45*

VI. Characterization of sensors and gates for use with Cello 48
V.A. *Measurement of RPU standard 48*
V.A. *Sensor characterization 51*
V.B. *Characterization of gates to be included in the UCF 54*

VII. Format of the User Constraint File (UCF) 58

VIII. Materials and Methods 76
VIII.A. *Circuit induction and measurement guide 76*
VIII.B. *Circuits library measurement and time-courses 76*
VIII.C. *Circuit analysis 77*
VIII.D. *Strain, media, and inducers 77*
VIII.E. *Design and assembly of 2-input circuits 77*
VIII.F. *Ribozyme cleavage assay 77*
VIII.G. *In vivo ribozyme insulation assay 78*
VIII.H. *Construction and screening of RBS libraries 78*
VIII.I. *Gate construction and characterization 79*
VIII.J. *Characterization of gate impact on cell growth 79*
VIII.K. *Flow cytometry analysis 79*
VIII.L. *Conversion of fluorescence to RPU 80*
VIII.M. *Genetic circuit assembly 80*
VIII.N. *Hexadecimal and Wolfram Rule naming conventions 81*
VIII.O. *Software tools 81*
VIII.P. *Precomputing 3-input 1-ouput NOR circuit diagrams 82*
VIII.Q. *RPU plasmid characterization using smRNA-FISH 83*

IX. Plasmid maps and DNA sequences 86
VIII.A. *Plasmid maps 86*
VIII.B. *DNA sequences 89*

X. Supplementary References 99

I. Design and characterization of insulated gates

I.A. Insulators of promoter context: design of ribozymes and spacers

The function of a genetic part can depend on its local genetic context; that is, the identity of up- and downstream parts (*1*). Previously, we found that the inclusion of the RiboJ insulator ensured that the response function of a gate would not be impacted by the identity of the input promoter(*2*). RiboJ is composed of two elements: (i) a hammerhead ribozyme derived from the satellite RNA of tobacco ringspot virus (sTRSV) that cleaves the 5'-UTR at a defined point and thereby removes upstream sequences that derive from the promoter, and (ii) an additional hairpin at the 3'-end of the ribozyme that helps expose the Shine-Dalgarno sequence of the RBS (Figure S1b). The entire RiboJ DNA sequence is 75 base pairs (bp), which is large enough to undergo homologous recombination if used more than once in a genetic circuit (*3–6*). Thus, each gate needs its own insulator with the same functionality of RiboJ but with a sequence that is different enough to prevent homologous recombination. To address this, we built and tested natural and engineered RiboJ variants and characterized both their cleavage activity and insulator functionality.

Two approaches were taken to identify ribozyme sequences that have diverse sequences but still function as insulators. First, "part mining" was performed to identify other hammerhead ribozymes derived from plant viroids and plant virus satellite RNAs. We built and tested sixteen hammerhead ribozymes (*7*, *8*) (including RiboJ) and others that had been previously tested as insulators (*2*). This approach ultimately led to the characterization of nine functional natural ribozyme-based insulators (Fig. S1a, Table S1).

A second approach to library expansion was taken by diversifying the sTSRV scaffold. This was aided by a number of structural studies detailing ribozyme function (*7–11*). Three design rules were implemented (Figure S1b). First, the sequences of the catalytic core residues (CTGATGA and GAAA) and two loops (GTGC and GTGA) were conserved (*7*, *8*). Second, the total number of nucleotides and the hammerhead secondary structure were kept intact. This was achieved by only mutating three stem regions: 5 bp of stem 1, 4 bp of stem 2, and 3 bp of stem 3. These mutated sequences were generated using the Random DNA Generator (http://www.faculty.ucr.edu/~mmaduro/random.htm; 50% GC-content). RNA secondary structures were predicted using mFold(*12*) and were found to maintain their hammerhead structure when simulated in isolation from flanking sequences (conditions: 37°C, 1M NaCl). We built and tested 45 engineered ribozymes, of which seven were functional and used to insulate gates (Table S1). For both the natural and engineered RiboJ variants, the downstream hairpin sequence was held constant due to its short size (23 nucleotides), which is short enough that it should not lead to homologous recombination.

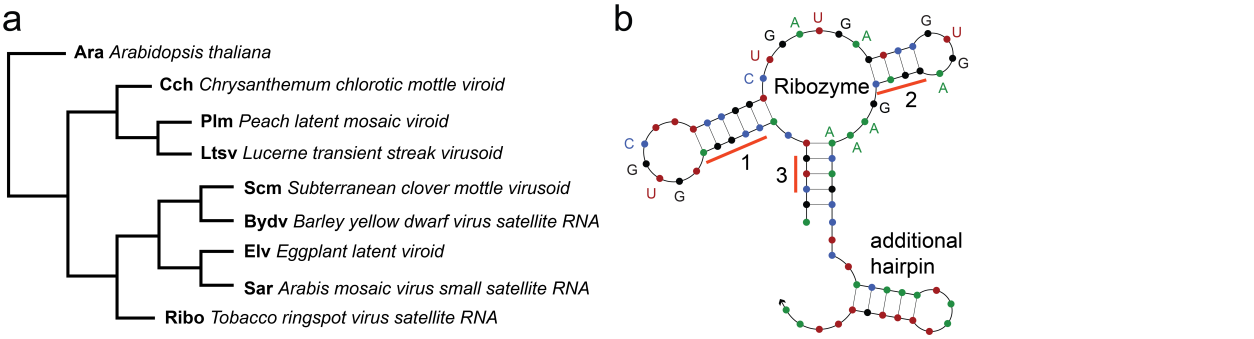

Figure S1: ***Expanding a library of hammerhead ribozymes.*** ***(A)*** *Phylogenetic tree of functional hammerhead ribozyme-based insulators. "Ribo" is the sequence used to build RiboJ.* ***(B)*** *Secondary structure of a hammerhead ribozyme-based insulator including the downstream hairpin. Conserved sequence regions are shown as defined nucleotides and mutable regions are shown adjacent to orange lines (1, 2, and 3).*

The natural and synthetic ribozymes were examined in two assays to measure cleavage activity and functional insulation. To measure cleavage, Rapid Amplification of Complementary DNA End (5'-RACE) was used to generate cDNA from mRNA by reverse-transcription for PCR amplification (Methods). Acrylamide gel analysis shows two bands: one from full-length, uncleaved mRNA and another from cleaved mRNA. The ratio between cleaved and total cDNA is used to calculate the efficiency (Figure S2). Several ribozymes, both engineered and natural, failed to achieve >75% cleavage efficiency. A set of 16 catalytically-active ribozymes is shown in Figure S2c.

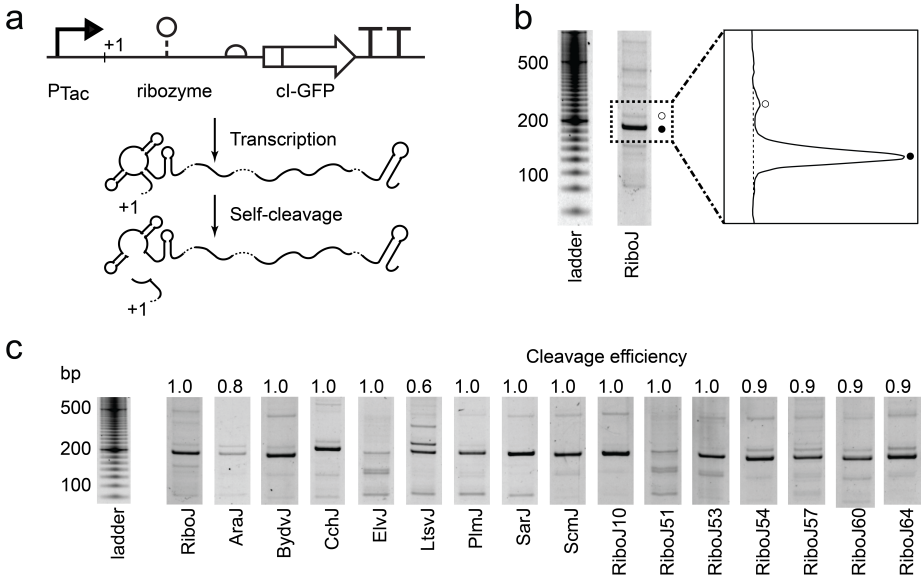

Figure S2: ***Cleavage activity of 16 ribozyme insulators. (A)*** *Schematic of ribozyme activity and measurement using 5'-RACE (Methods). Post-transcription, the hammerhead ribozyme folds and cleaves itself at its 5'-end. The measurement plasmids are pJS1-pJS68 (Supplementary Section IX).* ***(B)*** *Quantifying ribozyme cleavage activity using acrylamide gel electrophoresis and image processing. Full-length and cleaved cDNA products are separated and visualized, and then the band intensities (area under the curve, inset) are quantified using ImageJ. The intensity ratio of cleaved product (shorter band, filled circle) to the full-length product (longer band, empty circle) plus cleaved product yields the cleavage efficiency.* ***(C)*** *Acrylamide gel electrophoresis images and cleavage efficiencies of 16 ribozyme insulators.*

A second assay was performed to determine the insulation functionality of each RiboJ variant. Following an assay developed by Lou *et al*. (*2*), we compared the expression of two genes (*gfp* and *cI-gfp*) from two different inducible promoters, P_{Tac} and $P_{LlacO-1}$ (Figure S3a). The *cI-gfp* fusion gene saturates when induced by pLlacO-1, whereas this saturation is not observed from the pTac promoter (*2*) (Figure S3b). The RiboJ insulator was originally selected because its inclusion between the pLlacO-1 promoter and the RBS ameliorated this saturation and caused the outputs from both promoters to converge onto the same line. Further, the slopes of these lines are approximately constant, indicating that the two genes are expressed proportionally at different promoter activities. Thus, this assay is a direct measurement of insulation; in other words, the context effects that occur for particular promoter-gene combinations are reduced. All 16 RiboJ variants (including the original RiboJ) were tested via this assay and insulation was demonstrated for each (Figure S3c).

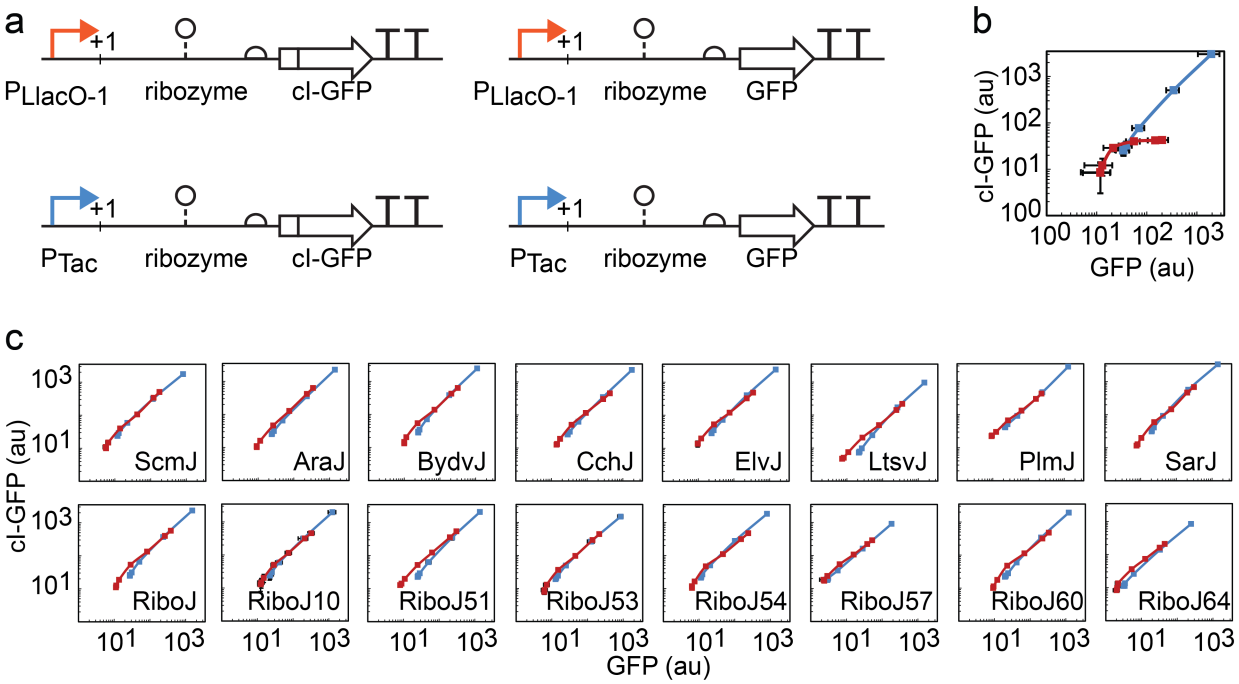

Figure S3: ***Insulating functionality of 16 ribozyme-based insulators. (A)*** *Schematics of genetic constructs used to determine insulating functionality of ribozymes. Two genes (cI-GFP and GFP) are each induced from one of two promoters ($P_{LlacO-1}$ and P_{Tac}) with various IPTG concentrations. The plasmids used for this study are pJS1-pJS68. (Supplementary Section IX).* ***(B)*** *Expression of cI-GFP versus GFP for $P_{LlacO-1}$ (red line) and P_{Tac} (blue line) when no ribozyme insulators are present. This experiment was performed to recapitulate the experiment in ref (2). Plasmids used are pJS1-pJS4 (Supplementary Section IX).* ***(C)*** *Expression of cI-GFP versus GFP for $P_{LlacO-1}$ (red line) and P_{Tac} (blue line) when various ribozyme insulators are used between the promoter and 5'-UTR. The slopes of the $P_{LlacO-1}$ and P_{Tac} lines for each ribozyme are as follows: ScmJ ($P_{LlacO-1}$ = 2.8, P_{Tac} = 2.2); AraJ (1.8, 1.6); BydvJ (2.0, 2.2); CchJ (1.1, 1.3); ElvJ (1.4, 1.6); LtsvJ (0.60, 0.65); PlmJ (1.9, 2.2); SarJ (2.2, 2.3); RiboJ (1.5, 1.5); RiboJ10 (1.4, 1.6); RiboJ51 (1.8, 1.5); RiboJ53 (2.1, 1.8); RiboJ54 (2.0, 2.3); RiboJ57 (5.7, 5.0); RiboJ60 (1.4, 1.6); and RiboJ64 (4.9, 3.5). For panels (B) and (C), error bars are one standard deviation of the median for three experiments performed on different days.*

Table S1: Ribozyme sequences

Name	Sequence[a]
RiboJ(*2*, *7*)	AGCTGTC|ACCGGATGTGCTTTCCGGTCTGATGAGTCCGTGAGGACGAAACAGCCTCTACAAATAATTTTGTTTAA
Natural	
AraJ(*8*)	AGTGGTC|GTGATCTGAAACTCGATCACCTGATGAGCTCAAGGCAGAGCGAAACCACCTCTACAAATAATTTTGTTTAA
BydvJ(*7*)	AGGGTGTC|TCAAGGTGCGTACCTTGACTGATGAGTCCGAAAGGACGAAACACCCCTCTACAAATAATTTTGTTTAA
CchJ(*7*)	AGTTCCAGTC|GAGACCTGAAGTGGGTTTCCTGATGAGGCTGTGGAGAGAGCGAAAGCTTTACTCCCGCACAAGCCGAAACT GGAACCTCTACAAATAATTTTGTTTAA
ElvJ(*8*)	AGCCCCATA|GGGTGGTGTGTACCACCCCTGATGAGTCCAAAAGGACGAAATGGGGCCTCTACAAATAATTTTGTTTAA
LtsvJ(*2*, *7*)	AGTACGTC|TGAGCGTGATACCCGCTCACTGAAGATGGCCCGGTAGGGCCGAAACGTACCTCTACAAATAATTTTGTTTAA
PlmJ(*2*, *7*)	AGTCATAAGTC|TGGGCTAAGCCCACTGATGAGTCGCTGAAATGCGACGAAACTTATGACCTCTACAAATAATTTTGTTTAA
SarJ(*2*, *7*)	AGACTGTC|GCCGGATGTGTATCCGACCTGACGATGGCCCAAAAGGGCCGAAACAGTCCTCTACAAATAATTTTGTTTAA
ScmJ(*7*)	AGCGCTGTC|TGTACTTGTATCAGTACACTGACGAGTCCCTAAAGGACGAAACACCGCCTCTACAAATAATTTTGTTTAA
Engineered	
RiboJ10	AGCGCTC|AACGGGTGTGCTTCCCGTTCTGATGAGTCCGTGAGGACGAAAGCGCCTCTACAAATAATTTTGTTTAA
RiboJ51	AGTAGTC|ACCGGCTGTGCTTGCCGGTCTGATGAGCCTGTGAAGGCGAAACTACCTCTACAAATAATTTTGTTTAA
RiboJ53	AGCGGTC|AACGCATGTGCTTTGCGTTCTGATGAGACAGTGATGTCGAAACCGCCTCTACAAATAATTTTGTTTAA
RiboJ54	AGGGGTC|AGTTGATGTGCTTTCAACTCTGATGAGTCAGTGATGACGAAACCCCCTCTACAAATAATTTTGTTTAA
RiboJ57	AGAAGTC|AATTAATGTGCTTTTAATTCTGATGAGTCGGTGACGACGAAACTTCCTCTACAAATAATTTTGTTTAA
RiboJ60	AGTCGTC|AAGTGCTGTGCTTGCACTTCTGATGAGGCAGTGATGCCGAAACGACCTCTACAAATAATTTTGTTTAA
RiboJ64	AGGAGTC|AATTAATGTGCTTTTAATTCTGATGAGACGGTGACGTCGAAACTCCCTCTACAAATAATTTTGTTTAA

a. Each insulator is annotated for functional sequences: | cleavage site, red catalytic cores, blue loops, underlined 3'-hairpin.

I.B. Terminator selection for transcriptional insulation

Strong terminators are needed for genetic circuits to prevent transcriptional read-through between gates. In addition, these terminators must be sequence-diverse to prevent homologous recombination (*4*, *5*). For the later circuits discussed in this work, we used strong terminators that were measured previously (*6*) (Table S2). These replaced double terminator BBa_B0015 used in earlier circuits.

Table S2: Terminator sequence alignment

Name	Strength[a]	Sequence
Natural		
ECK120033737	310	------GGAA----------ACACAGAAAAAAGCCCGCACCTGACAGTGCGGGCT-TTTTTTTTCGACCAAAGG
ECK120029600	380	TTCAGCCAAAAAACTTAAGACCGCCGGTCTTGTCCACTACCTTGCAGTAATGCGGTGGACAGGATCGGCGGTTTTCTTTTCTCTTCTCAA--
ECK120015440	120	------TCCGGC----------AATTAAAAAAGCGGCTAACCACGCCGCTTTTTTTACGTCTGCA-----
ECK120010876	97	-TAAGGTTGAAA----------AATAAAAACGGCGCTAAAAAGCGCCGTTTTTTTTGACGGTGGTA----
ECK120033736	170	----------AACGCATGA--GAAAGCCCCCGGAAG-ATCACCTTCCGGGGGCTTTTTTATTGCGC---------
ECK120010818	150	----------GTCAGTTTCA--CCTGTTTTACGTAAAAACCCGCTTCGGCGGGTTTTTACTTTTGG----------
ECK120015170	86	----------ACAATTTTCGAAAAAACCCGCTTCGGCGGGTTTTTTTATAGCTAAAA------
Engineered		
L3S3P31[b]	110	----------CCAATTATTGAACACCCTAACGGGTGTTTTTTTTTTTTTGGTCTACC--
L3S3P11[b]	170	----------CCAATTATTGAACACCCTTCGGGGTGTTTTTTTGTTTCTGGTCTACC--
L3S2P24	150	----------CTCGGTACCA---AATTCCAGAAAAGACACC--CGAAAGGGTGTTTTTTCGTTTTGGTCC------
L3S2P11	260	----------CTCGGTACCA---AATTCCAGAAAAGAGACGCTTTCGAGCGTCTTTTTTCGTTTTGGTCC------
L3S2P55	260	----------CTCGGTACCA---AAGACGAACAATAAGACGCTGAAAAGCGTCTTTTTTCGTTTTGGTCC------

a. Strength values reproduced from ref (6).
b. The "C" at nucleotide 45 from was mutated to "A" to eliminate a BsaI recognition site.

I.C. RBS selection to tune the response threshold

The strength of the RBS controlling repressor expression is one determinant of the threshold of a gate. When the ribozyme insulators were added to each gate, this impacted RBS strength and the thresholds shifted (or the response was completely eliminated). To alter the threshold of the insulated gates, we built and screened RBS libraries. For some gates, multiple RBSs were found that generated different thresholds. These were kept and included in the library so that there would be more ways in which the gate could be connected to others in the circuit.

The RBS libraries were built using PCR to amplify the gate plasmid with primers containing degenerate nucleotides in the region in and around the RBS (Methods). The resulting PCR products were ligated and transformed in *E. coli* NEB 10-beta. Individual clones from the gate RBS library were screened by growing them in the presence and absence of inducer. Clones with the largest dynamic range were chosen for further characterization. The full response functions of these gates were measured. Representative cytometry histograms and Hill equation fits to the data are given in Figure S4. The final RBS sequences are given in Table S3, and the response function parameters and toxicity threshold are listed in Table S4.

Table S3: Insulated gate RBS sequences

Repressor	RBS	DNA sequence
AmeR	F1	CTATGGACTATGTTTTCACATACGAGGGGGATTAG
AmtR	A1	AATGTTCCCTAATAATCAGCAAAGAGGTTACTAG
BetI	E1	CCCCCCGAGGAGTAGCAC
BM3R1	B1	CTATGGACTATGTTTTAACTACTAG
	B2	CTATGGACTATGTTTTTCAAAGACGAAAAACTACTAG
	B3	CCAAACGAGGCCGGGAGG
HlyIIR	H1	ACCCCCGAG
IcaRA	I1	ATTGCTATGGACTATGTTTCAAAGTGAGAATACTAG
LitR	L1	GTCCTATGGACTTTTTCATACAGGAGAACCCTCG
LmrA	N1	TACGCTATGGACTATGTTTTCTGCTATGGACTATGTTTTCACACACGAGATGCCTCG
PhlF	P1	CTATGGACTATGTTTGAAAGGGAGAAATACTAG
	P2	GGAGCTATGGACTATGTTTGAAAGGCTGAAATACTAG
	P3	CTTTACGAGGGCGATCCT
PsrA	R1	TTTAATTCGCGGAAGCGCAGAGATAAGGGGTATC
QacR	Q1	GTAAGCCATGCCATTGGCTTTTGATAGAGGATAACTACTAG
	Q2	GCCATGCCATTGGCTTTTGATAGAGGACAACTACTAG
SrpR	S1	GAGTCTATGGACTATGTTTTCACAGAGGAGGTACCAGG
	S2	GAGTCTATGGACTATGTTTTCACATATGAGATACCAGG
	S3	GAGTCTATGGACTATGTTTTCACAAAGGAAGTACCAGG
	S4	CTATGGACTATGTTTTCACACAGGAAATACCAGG

I.D. Response functions and cytometry data for insulated gates

Production of YFP from the insulated gates' outputs were measured at various inducer concentrations by cytometry and converted to output relative expression units (RPU, see Supplementary Section VI.C). For each of these gates, IPTG was used to induce gate expression from the P_{Tac} promoter. Additionally, inducer concentrations were converted to input promoter activity by measuring expression of YFP from P_{Tac} at those inducer concentrations (Figure S4a). The median input and output RPU values were plotted for each inducer concentration to create the experimental response function (Figure S4b).

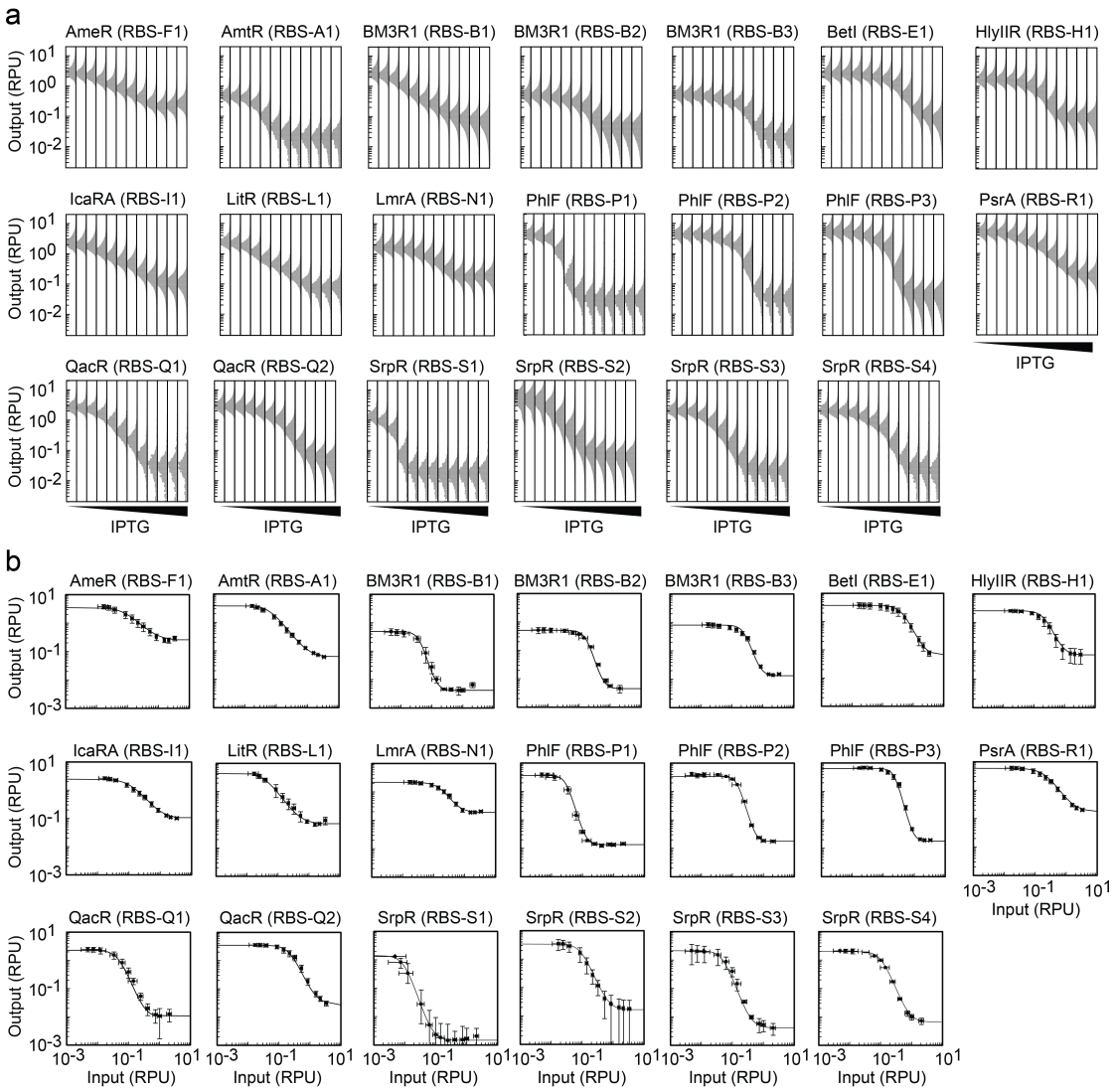

Figure S4: ***Distributions and response functions for insulated gates. (A)*** *Representative YFP fluorescence histograms for each gate are each normalized to RPU (see Supplementary Section VI.C). IPTG concentrations used were: 0, 5, 10, 20, 30, 40, 50, 70, 100, 150, 200, and 1000 μM.* ***(B)*** *The response functions are fit to Equation S1 (black lines). Error bars are one standard deviation of the median for three experiments performed on different days. Hill equation parameters are given in Table S4.*

These empirical response functions were fit to a Hill equation:

$$y = y_{min} + \frac{(y_{max} - y_{min})K^n}{x^n + K^n} \quad \text{(S1)}$$

where y is the output promoter activity, x is the input promoter activity, y_{max} is the maximum observed promoter output value, y_{min} is the minimum observed promoter output value, K is the repression threshold (the input value at which the output is half maximum), and n is the Hill coefficient. The Hill equation is overlayed with the experimental response function (Figure S4b), and the parameters for the Hill equation fits are provided in Table S4.

Table S4: Insulated gate response function parameters

Repressor	RBS	y_{min}[a]	y_{max}[a]	K[a]	n	Toxicity (RPU)[b]
AmeR	F1	0.2	3.8	0.09	1.4	-
AmtR	A1	0.06	3.8	0.07	1.6	4.1
BetI	E1	0.07	3.8	0.41	2.4	-
BM3R1	B1	0.004	0.5	0.04	3.4	-
	B2	0.005	0.5	0.15	2.9	-
	B3	0.01	0.8	0.26	3.4	-
HlyIIR	H1	0.07	2.5	0.19	2.6	-
IcaRA	I1	0.08	2.2	0.10	1.4	1.7
LitR	L1	0.07	4.3	0.05	1.7	0.2
LmrA	N1	0.2	2.2	0.18	2.1	-
PhlF	P1	0.01	3.9	0.03	4.0	-
	P2	0.02	4.1	0.13	3.9	-
	P3	0.02	6.8	0.23	4.2	-
PsrA	R1	0.2	5.9	0.19	1.8	-
QacR	Q1	0.01	2.4	0.05	2.7	N.D.
	Q2	0.03	2.8	0.21	2.4	1.7
SrpR	S1	0.003	1.3	0.01	2.9	-
	S2	0.003	2.1	0.04	2.6	-
	S3	0.004	2.1	0.06	2.8	-
	S4	0.007	2.1	0.10	2.8	-

a. In units of RPU (see Supplementary Section VI.C).
b. Highest input RPU achieved before cell growth was reduced >20% compared to a control (Methods). Dashes indicate no toxicity observed at the highest inducer levels. N.D. means no data collected.

II. Simple gate combinations

II.A. Non-insulated gates: predicted and measured outputs

Originally, the simple circuits (Figure 3a, left data column) were build based on non-insulated gates taken directly from a subset of the repressors previously characterized (*13*). We allowed a library of 16 members, each of which used the same terminator (BBa_B0015). Response functions for these gates were determined as the activity of the output promoter versus the activity of the input promoter (in RPU). The response function of each gate was fit to a Hill equation (Equation S1), the parameters of which are in Table S5.

The non-insulated gates were assembled to form the wiring diagrams shown in Figure S5. The gate assignments differ from those built with the insulated gates (indicated by color). The detailed parts are also different and shown in this figure. The genetic circuits were inserted into the same plasmid backbone as the insulated gates (Figure 1c) and included YFP on the same plasmid as the circuits (no output plasmid).

The assembly strategy used for the non-insulated circuits differed slightly from the insulated circuits. Golden Gate assembly was used to assemble the final circuits, but we used different 4 bp scars than for the insulated circuits. We also used a two-tier assembly where intermediate constructs with 1-4 transcription units were assembled first and then assembled to build the final circuit. The sensor block was also assembled with gate modules into intermediate constructs. This is in contrast to the insulated circuits where the sensor block was cloned into the plasmid before circuit assembly, and then all circuit modules were cloned into the backbone in one step.

Table S5: Non-insulated gate parameters[a]

Name	K[b]	n	y_{max}[b]	y_{min}[b]
AmeR	0.11	1.4	3.9	0.40
AmtR	0.06	1.8	2.2	0.08
BetI	0.05	2.4	3.1	0.09
BM3R1	0.13	4.5	0.61	0.02
ButR	0.30	2.4	2.9	0.44
HlyIIR	0.12	2.7	3.9	0.08
IcaR(A)	0.10	1.8	3.0	0.09
LitR	0.03	1.9	3.9	0.12
LmrA	0.29	3.1	17	0.27
McbR	0.10	1.6	3.8	0.27
PhlF	0.09	4.5	3.8	0.02
PsrA	0.10	2.0	4.7	0.11
QacR	0.11	1.4	5.0	0.05
SrpR	0.07	3.2	6.0	0.03
TarA	0.02	1.8	3.0	0.05

a. The RPU standard in ref (*13*) differs from this manuscript (Figure S32). These values were recalculated based on the new standard.
b. In units of RPU.

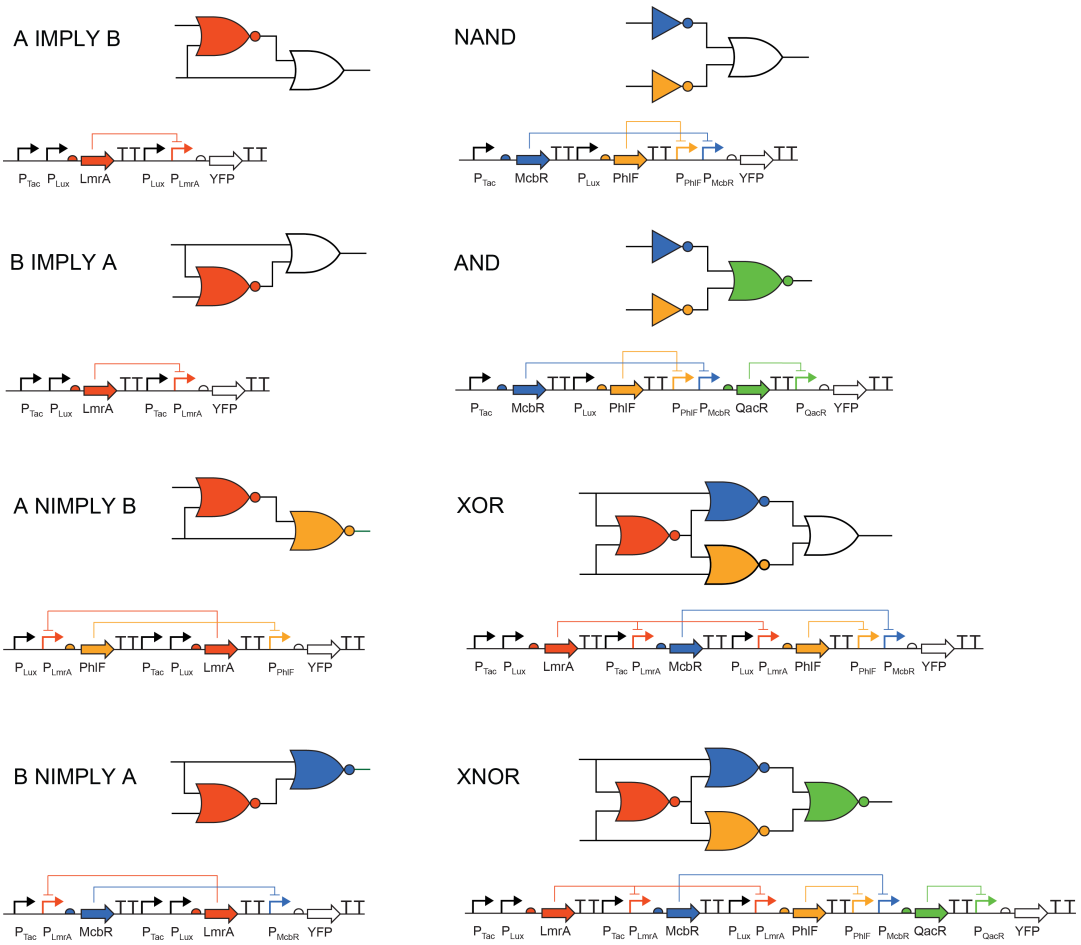

Figure S5: ***Circuit diagrams and genetic schematics for simple circuits built from non-insulated gates.*** *This corresponds to the "non-insulated" data shown in Figure 3a. Gate colors correspond to the repressors in the genetic construct. All the terminators are the same (BBa_B0015) and are shown as a black "TT". Plasmids used were pAN901-908 (Supplementary Section IX).*

II.B. Characterization of error modes

The circuits built from non-insulated gates were almost entirely non-functional. We identified several failure modes in these circuits, which when corrected fixed the circuit function (Supplementary Section II.C). We describe the design solutions for five primary error modes in this section: mismatched response functions, promoter/5'-UTR contextual effects, promoter interference, homologous recombination, and toxicity.

Mismatched response functions. In the construction of the non-insulated gates, several of response functions were mismatched. The outputs from one gate frequently did not map onto either side of the threshold of the downstream gate (Figure S6). For example, in the initial construction of the XNOR circuit from non-insulated gates (Figure S6a), the outputs from the LmrA NOR gate (red gate) is very high, even in its repressed OFF state. These high OFF-state outputs map onto to the downstream response functions to the right of the threshold point. This causes the signal to deteriorate after the first layer. For subsequent circuit designs, the selection of repressor assignments based on the circuit's output dynamic range ensured that the response functions of gates connected to each other in a functional manner. For example, the XNOR circuit built from insulated gates (Figure S6b) has good predicted separation between ON and OFF promoter levels after each gate in the circuit.

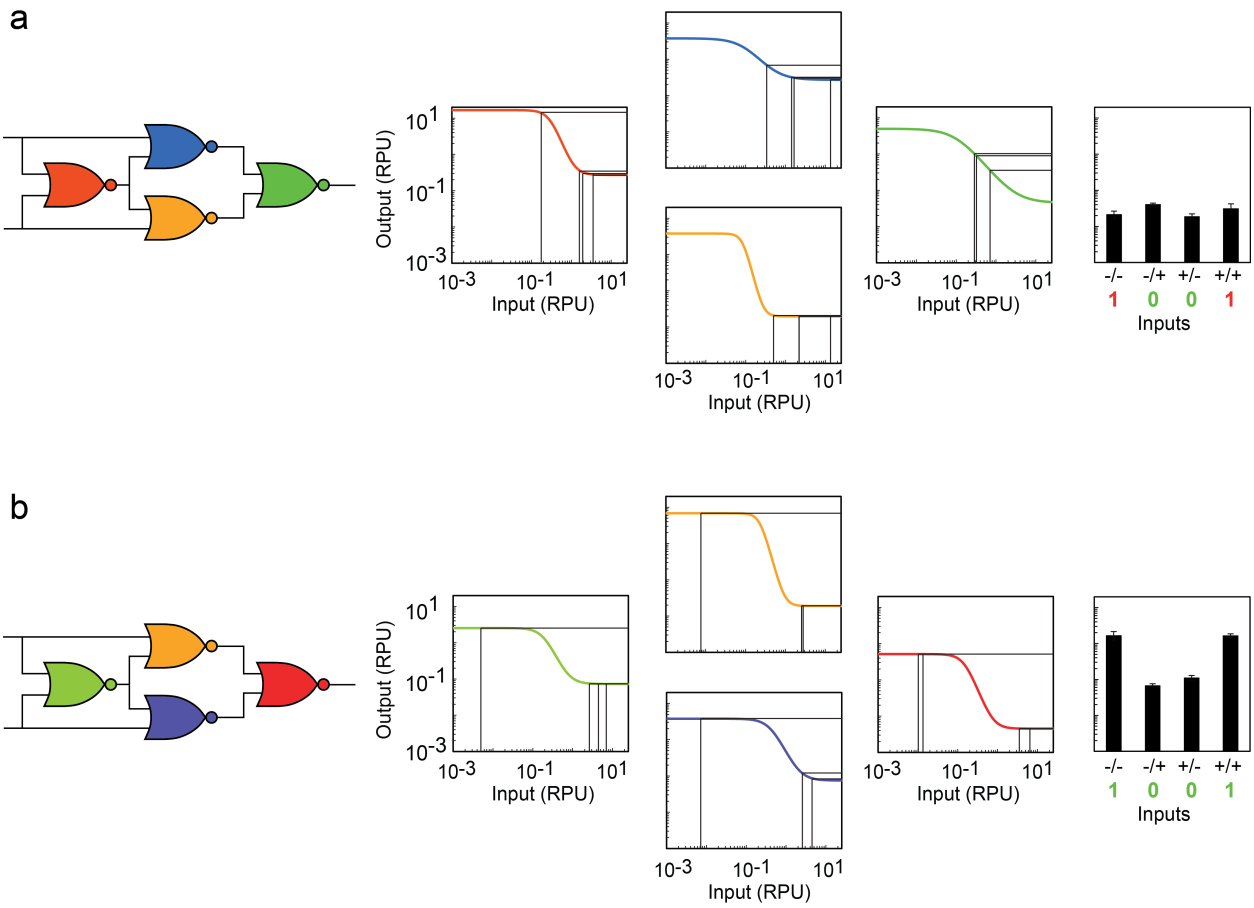

Figure S6: ***Response function matching in circuits. (A)*** *XNOR circuit built from non-insulated gates. Assuming the response functions behave the same in the context of a circuit, the circuit is still predicted to be non-functional because all the output states from the first gate map onto the next gate's response function to the right of the threshold. Experimental data from Figure 3a shown at right. Inputs correspond to the absence or presence of 1 mM IPTG (right -/+) and 20 μM 3OC6HSL (left -/+).* ***(B)*** *XNOR circuit built from insulated gates (Figure 3a). The repressor assignment algorithms cause the outputs from the first gate to span the threshold of each gate. Experimental data from Figure 3a shown at right. Inputs correspond to the absence or presence of 1 mM IPTG (right -/+) and 2 ng/mL aTc (left -/+). For both panels, error bars are one standard deviation of the median for three experiments performed on different days.*

Promoter/5'-UTR context effects. The response function of a NOT gate can change when connected to different input promoters (*2*). In addition, for NOR gates the connection of two promoters in series can lead to contextual effects as they create transcripts of different length (Figure S7), which changes the length of the 5'-UTR—a sensitive region for controlling expression (*14–17*). This manifested as an error mode, where gates that functioned properly as NOT gates fail when converted to NOR gates. The promoter context and 5'-UTR effects can be mitigated through the inclusion of a ribozyme after the promoter, which cleaves the mRNA at a defined nucleotide. This makes the transcripts identical, whether they are produced by the upstream or downstream promoter.

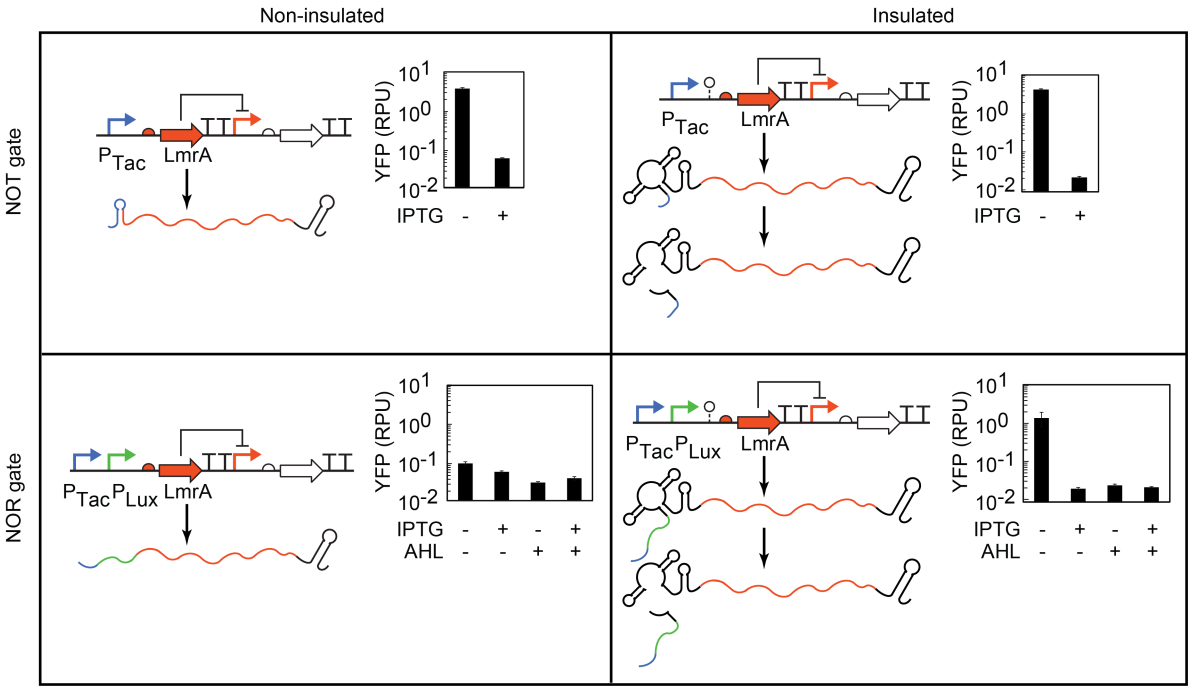

Figure S7: ***Comparison of non-insulated and insulated NOT/NOR gates.*** *NOT and NOR gates without ribozymes contain promoter sequence in the mRNA transcript that can affect translation (left panel). Black bars are expected to be high and gray bars expected to be low. Cells were grown measured with the presence and absence of inducers: 1 mM IPTG and 20 μM 3OC6HSL (Methods). Error bars are one standard deviation of the median for three experiments performed on the same day. The plasmids used are: pAN215 = non-insulated NOT gate, pAN216 = non-insulated NOR gate, pAN412 = insulated NOT gate, and pAN413 = insulated NOR gate (Supplementary Section IX).*

Terminator recombination. The evolutionary stability of a genetic circuit is dependent on several factors. Repeated genetic sequences undergo homologous recombination at a frequency that increases with the repeat length and the number of repetitions (*3*, *18*). Initially, we designed genetic circuits that each contained the 129 bp double terminator BBa_B0015. The largest such circuit, XNOR, underwent rapid homologous recombination that resulted in a non-functional circuit (Figure S8). We sequenced the plasmid and found that the AmtR transcription unit had looped out of the plasmid by homologous recombination between two instances of the BBa_B0015 double terminator. This caused constitutive expression of SrpR and BM3R1 which lead to an always ON output from the circuit. To mitigate homologous recombination, we used a library of sequence-diverse strong terminators (Table S2) to terminate gene expression.

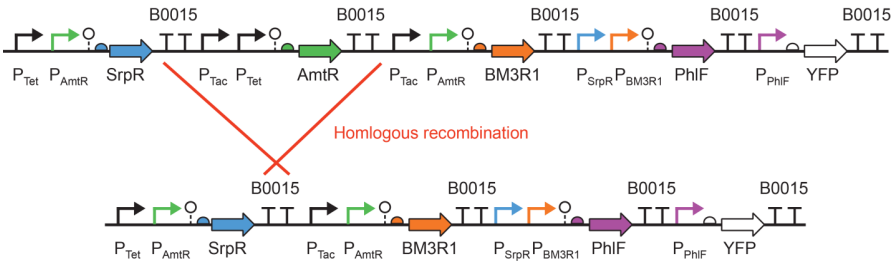

Figure S8: ***Repeated terminators cause high rates of homologous recombination.*** The top construct is the original design. The bottom construct was identified by sequencing, where the AmtR gate was deleted by recombination.

Roadblocking. Genetic NOR gates contain tandem promoters that drive expression of repressors. Our initial assumption was these promoters would function independently, where the activity of a downstream promoter would not impact the activity of an upstream promoter, and vice versa. In practice, we found that some promoters in the downstream position (position 2 of Figure 2C) could interfere with the upstream promoter when in the repressed state. We refer to this effect as "roadblocking" (the name is not intended to imply mechanism). We developed a simple system to measure the propensity of each promoter to roadblock when in the downstream position (Figure S9a). YFP is measured from a single promoter (P_{Tac} or P_{Tet}) by cytometry. Next, we insert a second promoter downstream from this promoter (position 2) and the impact on the upstream promoter was quantified (Figure S9b). We incorporated this roadblocking data into Cello by forbidding P_{BAD}, P_{Tac}, P_{PhlF}, P_{BM3R1}, P_{SrpR}, and P_{QacR} from occupying the second position in a tandem promoter. This appears as Eugene rules in the UCF and impacts the repressor assignment by disallowing multiple gates with output promoters that exhibit roadblocking to both serve as inputs to a downstream gate (Supplementary Section V.E).

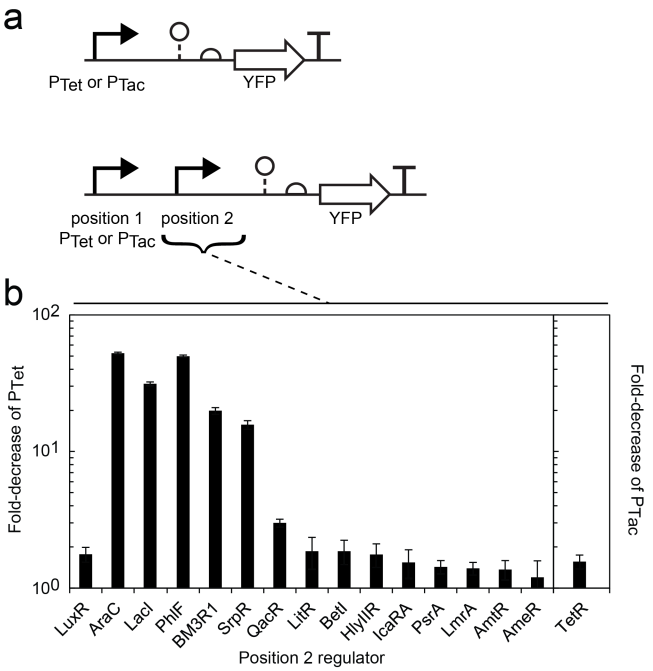

Figure S9: ***Measuring the roadblocking ability of various repressors.*** *The ability of a repressed promoter in position 2 to reduce YFP expression from a promoter in position 1 was tested.* ***(A)*** *Promoter 1 alone (either P_{Tac} or P_{Tet}) was induced to express YFP and was measured by cytometry. Next, a second promoter was inserted downstream from P_{Tac} or P_{Tet}. The upstream promoter was induced, and the downstream promoter was repressed (inactivated in the case of LuxR and AraC). The decrease in YFP expression compared to the P_{Tac}- or P_{Tet}-only case was used to calculate roadblocking. Plasmids used are pAN1250 and pAN1681-pAN1697 (Supplementary Section IX).* ***(B)*** *The fold-decrease caused by each repressed (or unactivated) promoter when in the downstream position. The upstream promoter is P_{Tet} in all cases, except for when the ability of P_{Tet} to roadblock is being measured, in which case the upstream promoter is P_{Tac}. Error bars are one standard deviation of the median for three experiments performed on different days (Methods).*

Toxicity. Certain repressors can be toxic when overexpressed, causing slow cell growth. For example, we constructed an AND gate from the PhlF, BM3R1, and QacR gates, and did not expect to see an impact on growth. However there was a growth defect when the cells were induced with 2 ng/mL aTc (which expresses QacR from P_{Tet}). When the repressors were initially characterized (*13*), their toxicity was measured by inducing their expression from P_{Tac} using various concentrations of IPTG. However, we later found that repressors that were initially not measured to be toxic impacted cellular growth when

expressed from promoters stronger than P_{Tac}, as in the case of QacR being expressed from P_{Tet}. To determine whether genes expressed at higher levels could exhibit toxicity, we cloned the repressors downstream from a tandem inducible promoter (P_{Tac}-P_{Tet}) and measured the impact on growth at various IPTG and aTc concentrations (Figure S10).

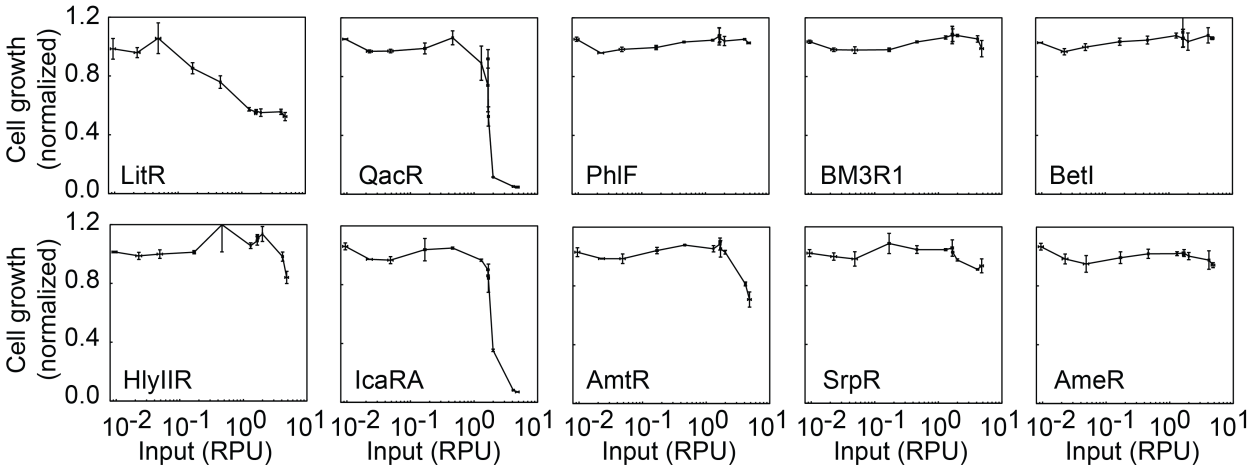

Figure S10: ***Toxic circuits and gate measurements.*** *We induced expression of each repressor from the tandem promoter with seven IPTG concentrations: 0, 9.5, 19, 47.5, 95, 285, and 950 μM; for an additional five samples, we induced with 950 μM IPTG along with aTc at concentrations: 0.0095, 0.095, 0.285, 0.95, and 1.9 ng/mL (Methods). After a period of growth, we measured the cultures' absorbances at 600 nm and normalized the values to the uninduced sample. For x-axis values, YFP was measured from the same tandem promoter at the same inducer concentration and fluorescence was converted to RPU. Error bars are one standard deviation of absorbance (y-axis) and the median (x-axis) for three experiments performed on the same day. Plasmids used were pJS0101-pJS0109.*

II.C. Insulated gates: predicted and measured outputs

The circuits constructed from non-insulated gates (Supplementary Section II.A) were rebuilt using insulated gates and design rules extracted from the previous section. These repressor assignments were found using a MATLAB script that was developed prior to the complete Cello software (Methods). For each circuit, we started with the circuit diagram (Figure 3a) and a subset of repressors from Figure 3b. We enumerated all possible repressor assignments for every gate, with the exception of assignments that would result in a promoter roadblock. For each gate assignment, we propagated sensor input signals through the gate response functions to the circuit output, summing promoter activities at NOR and OR gate inputs. Each circuit assignment was scored as the ratio between the lowest ON state and the highest OFF state. The highest scoring assignment was selected for construction and testing (Methods). For these circuits, the only promoters forbidden from being in position 2 of the NOR gates were P_{SrpR} and P_{Tac}. Figure S11 shows the experimental data from Figure 3a alongside the simulated outputs for the circuits.

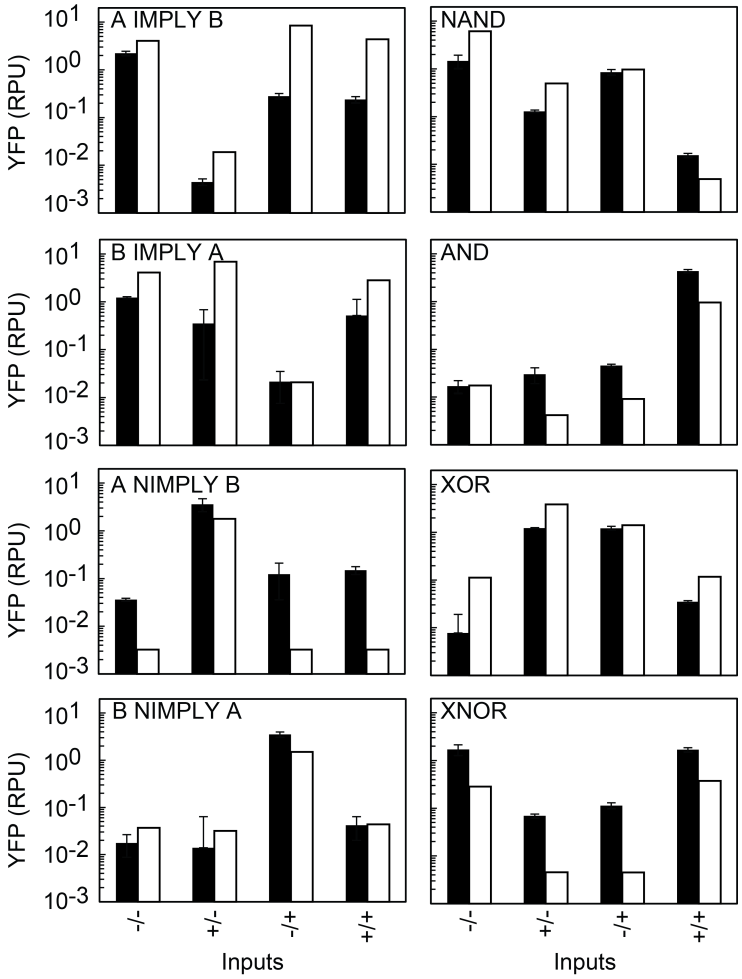

Figure S11: ***Predicted and measured outputs for simple circuits built from insulated gates.*** *Experimentally measured outputs (black bars) for the circuits in Figure 3a, alongside predicted outputs (white bars) generated from sensor input levels and gate response functions. Inputs are the absence or presence of 1 mM IPTG (bottom -/+) and 2 ng/mL aTc (top -/+). Error bars are one standard deviation of the median for three experiments performed on different days. Plasmids used were pAN901-pAN908 (Supplementary Section IX).*

III. Complete circuit data

We constructed a library of 3-input, 1-output genetic circuits using Cello. All the fully functional circuits are shown with data and predictions in Figure 4. The remaining circuits are shown in Figures S12 – S14. Experimental/predicted distributions and replicates for the Majority circuit variants are shown in Figures S15 and S16. Experimental/predicted distributions and replicates for three alternate circuit assignments are shown in Figure S17 and S18. Replicates for the circuit library are shown in Figures S19 and S20.

III.A. Circuits with 1 failed output state

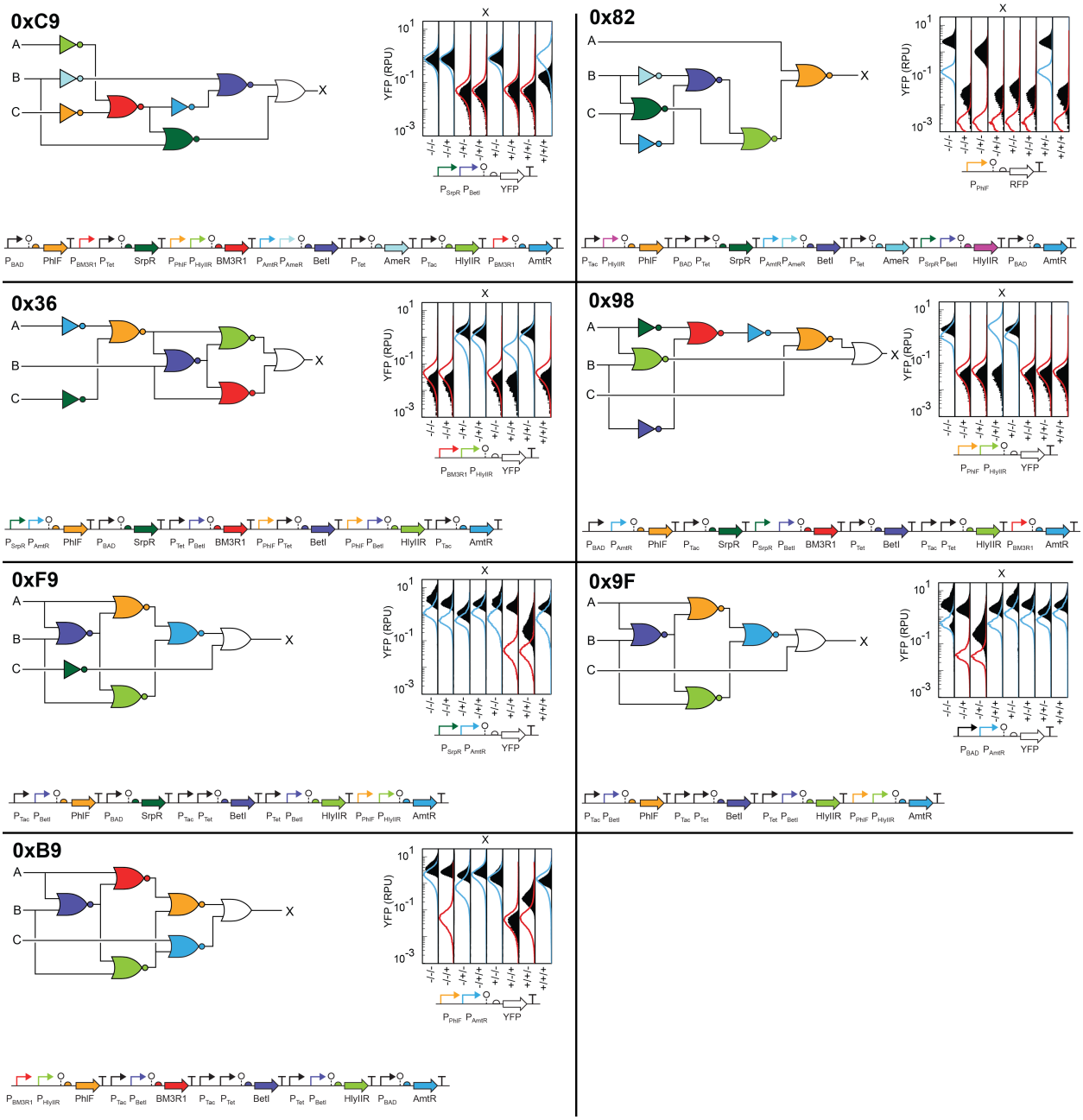

Figure S12: ***Circuits with 1 failed output state.*** *Representative experimentally measured fluorescence histograms (black) and predicted distributions (blue and red lines) are shown for circuits with a single failed output state. Inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+).*

III.B. *Circuits with 2 failed output states*

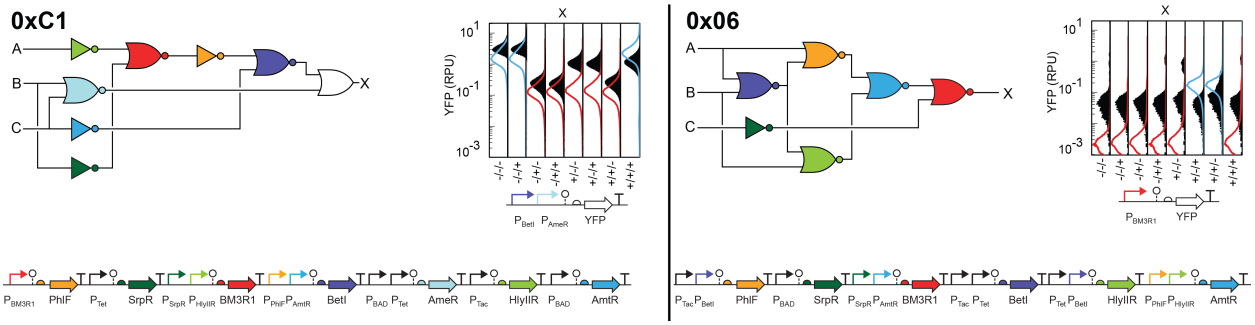

Figure S13: ***Circuits with 2 failed output states.*** *Representative experimentally measured fluorescence histograms (black) and predicted distributions (blue and red lines) are shown for circuits with two failed output states. Inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+).*

III.C. *Circuits with ≥3 failed output states*

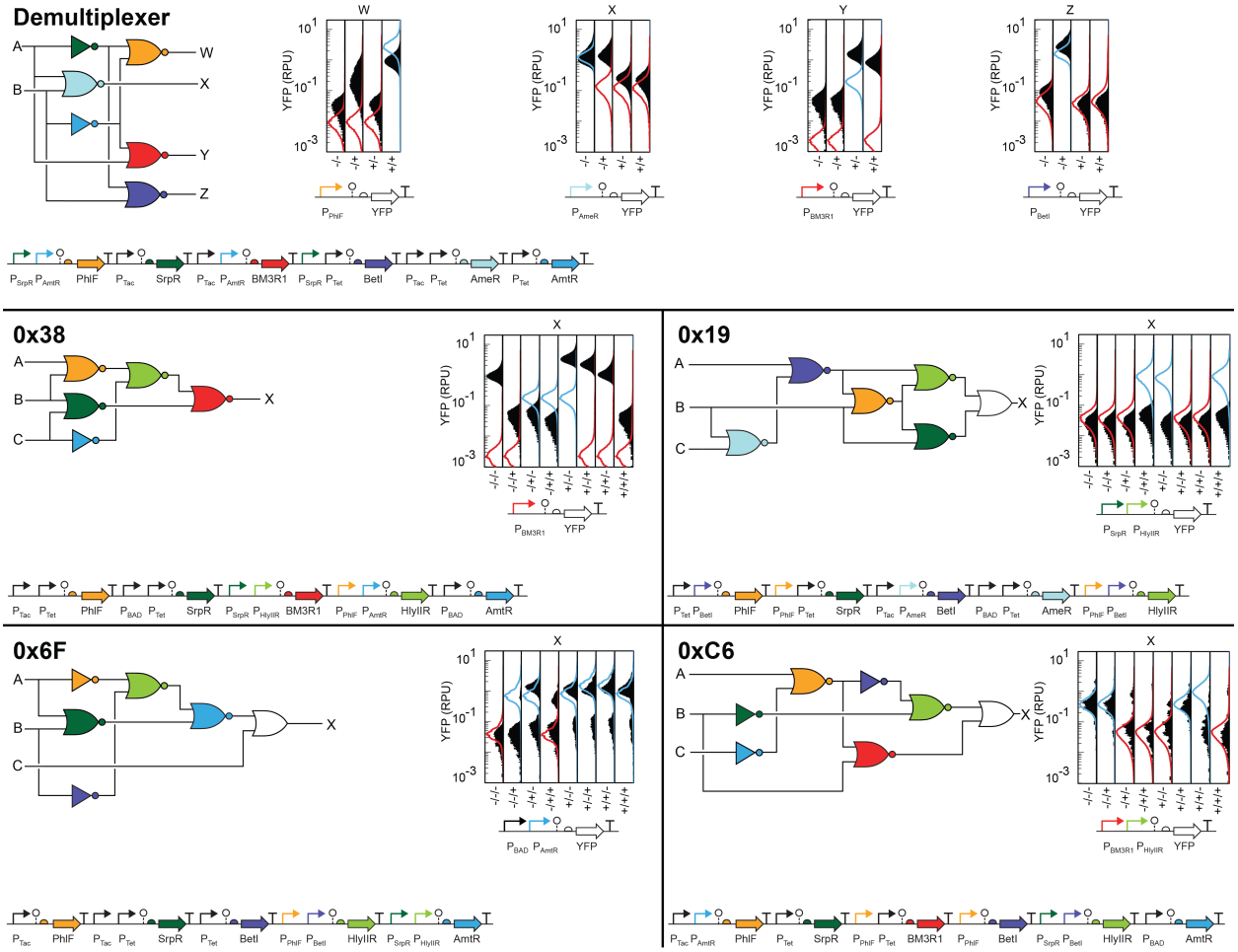

Figure S14: ***Circuits with 3 or more failed output states.*** *Representative experimentally measured fluorescence histograms (black) and predicted distributions (blue and red lines) are shown for circuits with three or more failed output states. For the demultiplexer circuit, inputs correspond to the absence or presence of 1 mM IPTG (top -/+) and 2 ng/mL aTc (bottom -/+). For all other circuits, inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+).*

III.D. Majority circuit variants

The original design for the Majority circuit produced an output that was higher than expected for input state 2 (+IPTG, —aTc, and —arabinose). We constructed an additional five layouts that retained the same repressor assignments to test whether we could fix that output state (Figure 5e). We hypothesized that subtle contextual effects might arise in different layouts (terminator read-through, part interference, cryptic promoters, etc.), and that these effects could improve the circuit's performance.

For the original circuit (Design #1), we used the default Cello layout where all transcription units point in the forward orientation and the repressors have a defined order (PhlF, SrpR, BM3R1, BetI, HlyIIR, AmtR). Design #2 reverses the order of all transcription units, keeping them pointed in the forward orientation. Design #3 clusters the three NOT gates together in the first half of the DNA sequence, and the three NOR gates together in the second half. Design #4 clusters one three gate sub-circuit (AmtR, BetI, SrpR) in the first half of the DNA sequence, and a second three gate sub-circuit (BM3R1, PhlF, HlyIIR) in the second half. Design #5 scrambles the order of the transcription units, keeping them pointed in the forward orientation. Design #6 uses the default transcription unit positions, but alternates their orientation so that the first transcription unit points backward, the second points forward, the third points backward, and so on. Each of these genetic layouts was physically constructed by simply changing the 4 bp Golden Gate scars that occur between transcription units.

The precise rules that govern the layout of these circuits were converted to Eugene code (Supplementary File 3). Each variant used a different Eugene file, and the only differences are a small set of rules in the "circuit device". The different rule sets only use three different Eugene keywords, but in different combinations: ALL_FORWARD, ALTERNATE_ORIENTATION, and BEFORE (e.g. gate_PhlF BEFORE gate_SrpR). The promoter order was also fixed due to roadblocking constraints in "gate devices".

In principle, if all the gates were perfectly modular and exhibited no contextual behavior differences, then all six layouts should function identically. Instead, we observed slight shifts in the output distributions for each circuit. The alternating orientation layout (Design #6) produced the greatest fold-change between the lowest ON state and the highest OFF state; furthermore, the high OFF state from Design #1 was decreased to more closely match the predictions. Representative experimentally-measured histograms and the predicted outputs are shown in Figure S15.

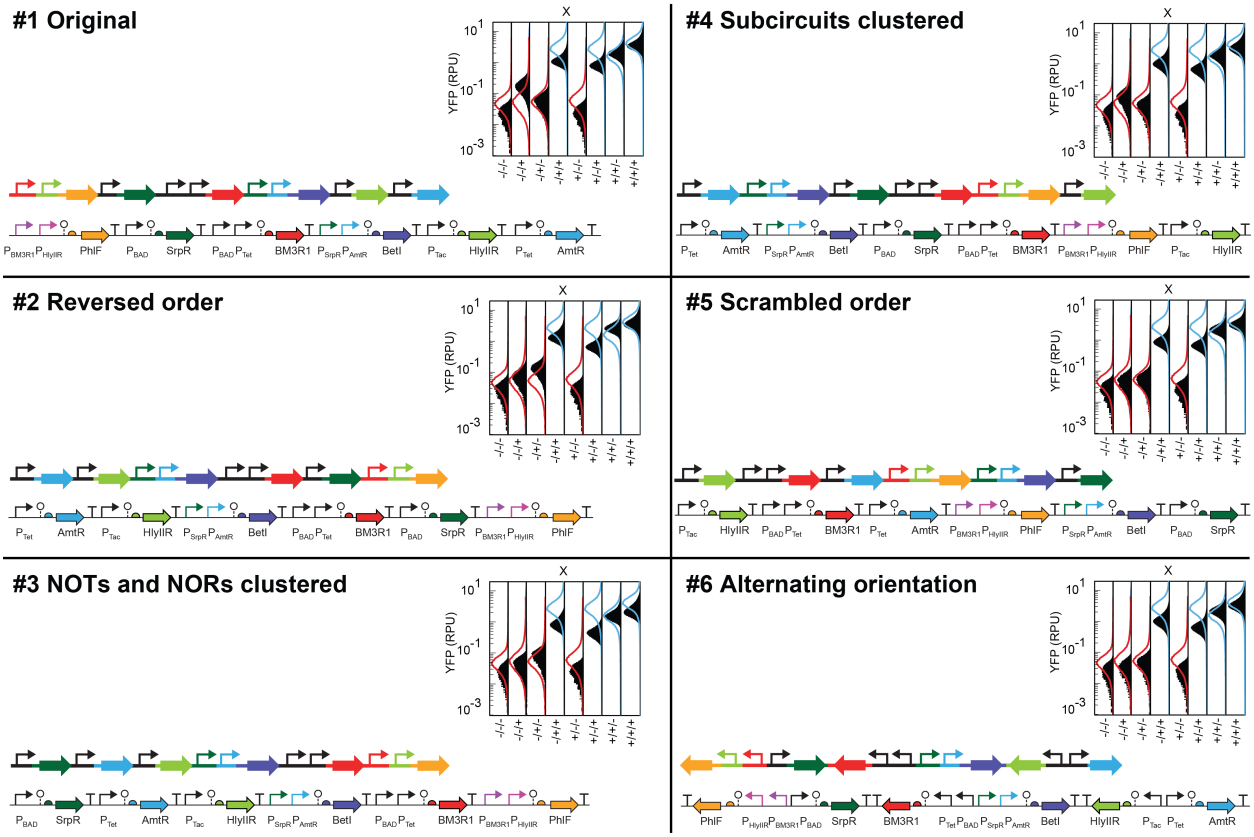

Figure S15: ***Majority circuit variants.*** *Representative experimentally measured fluorescence histograms (RPU, black) and predicted distributions (blue and red lines) are shown for the Majority circuit variants in Figure 5.The simplified genetic schematics from Figure 5e are shown above the full, labeled schematics. Inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+).*

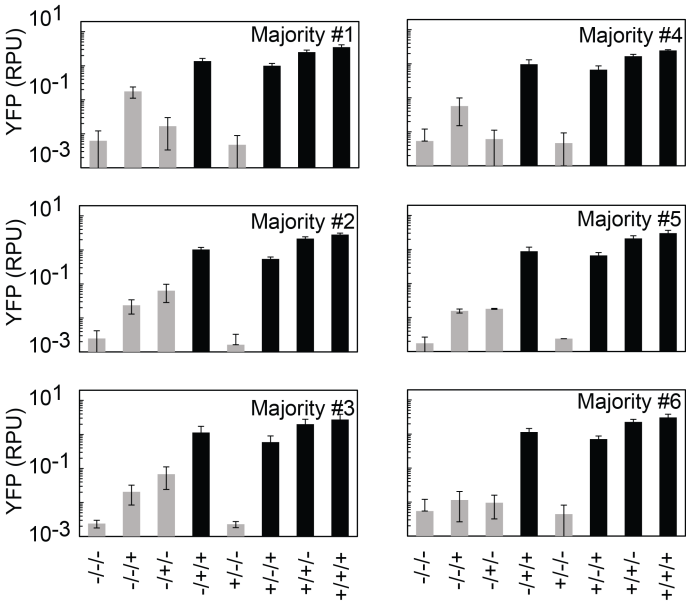

Figure S16: ***Replicates of majority circuit variants.*** *Average output (RPU) for alternate repressor assignments circuits (Figure S15). Outputs are predicted to be high (black bars) or low (gray bars). Error bars are one standard deviation of the median for two experiments performed on different days. Inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+).*

III.E. Alternate repressor assignments

In addition to testing different layouts for a single gate assignment (Supplementary Section III.D), we constructed three of the circuits (Multiplexer, Consensus, and Majority) using alternate repressor assignments predicted by Cello (Figure S17). The alternate Multiplexer circuit replaces only the terminal PhlF NOR gate from the original assignment (Figure 4a) with a BM3R1 NOR gate. The outputs are correct for all of the assignments. The alternate Consensus circuit swaps two gate assignments (HlyIIR and PhlF) from the original circuit. This swap results in two failed output states, whereas the original version had all states correct (Figure 4a). The alternate Majority circuit changes the assignment for every gate, except for the HlyIIR NOT gate connected to the P_{Tac} input. While most of the repressors are present in the same circuit layer in both circuits, BM3R1 is absent in the alternate assignment and AmeR is present. The alternate Majority circuit's output behaves correctly for every input state.

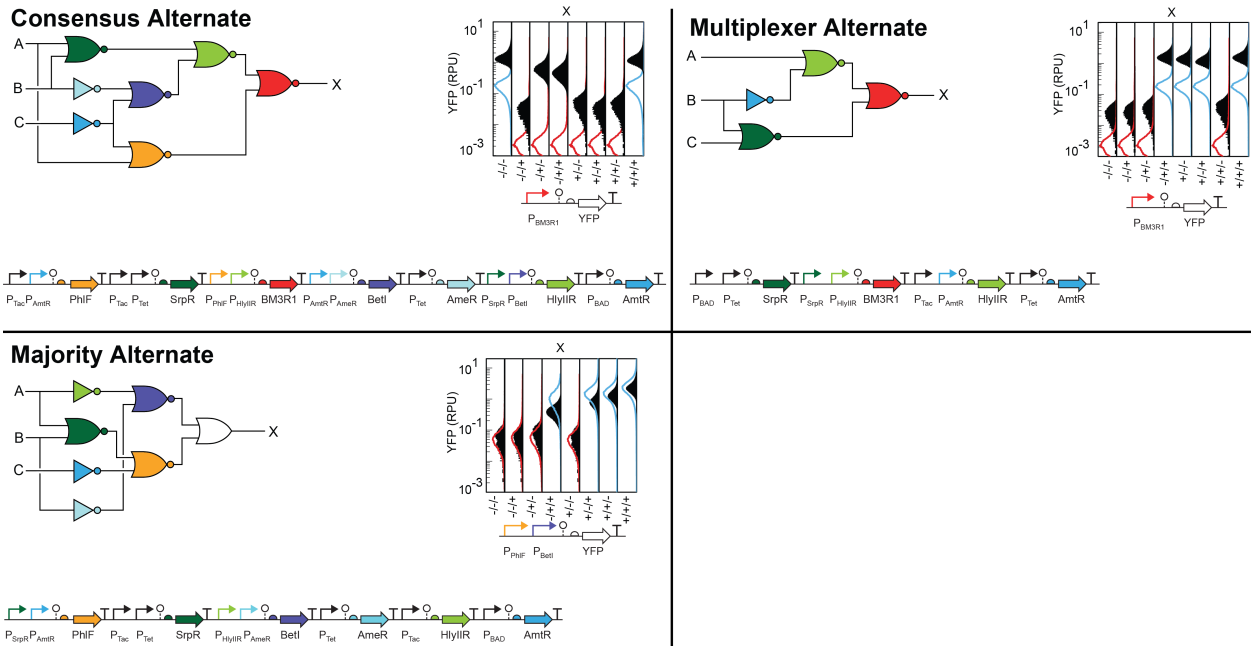

Figure S17: ***Alternate repressor assignments.*** *Representative experimentally measured fluorescence histograms (black) and predicted distributions (blue and red lines) are shown. Inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+).*

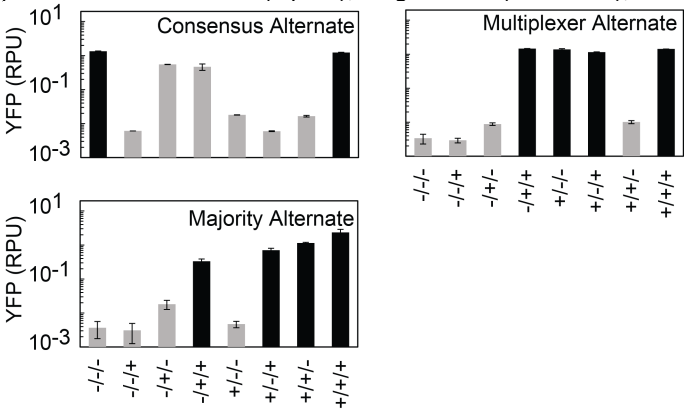

Figure S18: ***Replicates of alternate assignment circuits.*** *Average output (RPU) for alternate repressor assignments circuits (Figure S17). Outputs are predicted to be high (black bars) or low (gray bars). Error bars are one standard deviation of the median for two experiments performed on different days. Inputs correspond to the absence or presence of 1mM IPTG (to -/+), 2ng/mL aTc (middle-/+), and 5mM L-arabinose (bottom -/+).*

III.F. Replicates of circuit library

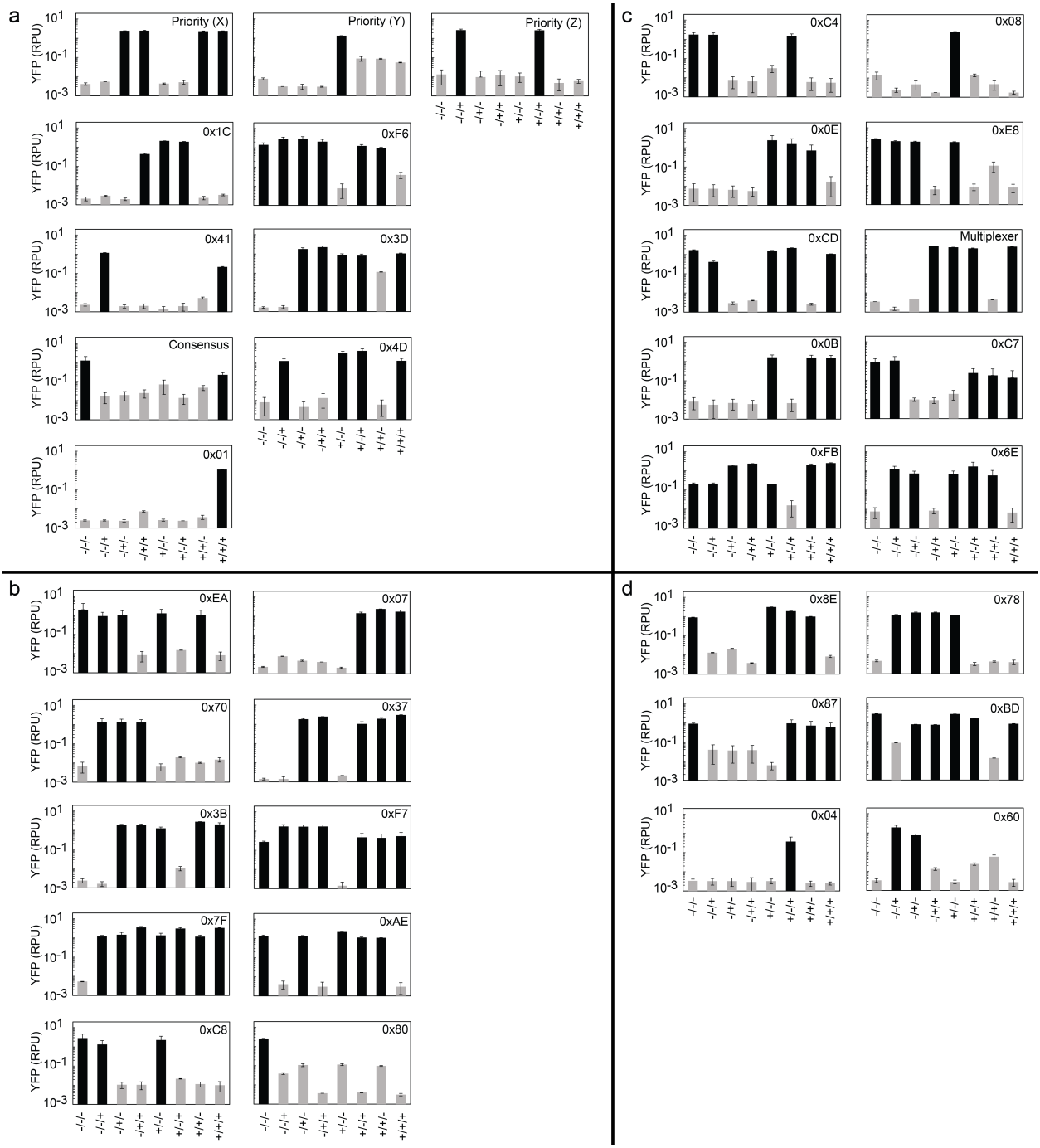

Figure S19: ***Replicates of functional circuits.*** *Average output (RPU) for all functional 3-input circuits from the circuit library (Figure 4). The ordering of the circuits in panels (A)-(D) matches the four pages of circuits in Figure 4. Outputs (X, Y, and Z) correspond to YFP driven from output promoters in separate experiments, and are predicted to be high (black bars) or low (gray bars). Error bars represent one standard deviation of the median for two experiments performed on different days. Inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+).*

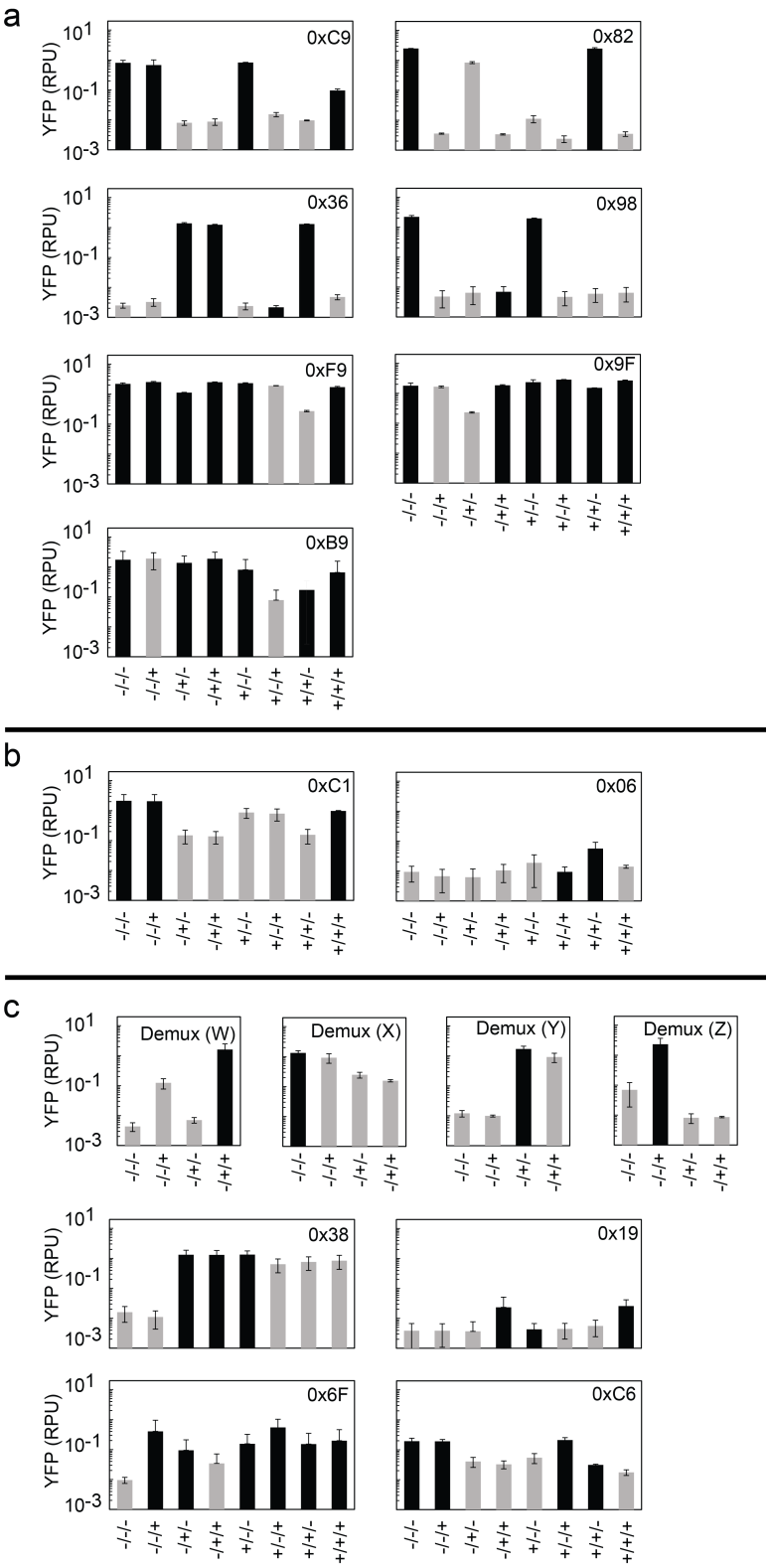

Figure S20: ***Replicates of circuits with failed output states.*** *Average output for circuits with* ***(A)*** *1 failed output state (Figure S12),* ***(B)*** *2 failed output states (Figure S13), or* ***(C)*** *3 or more failed output states (Figure S14). The ordering of the circuits matches the corresponding Supplementary Figures. Outputs (W, X, Y, and Z) correspond to YFP driven from output promoters in separate experiments, and are predicted to be high (black bars) or low (gray bars). Error bars are one standard deviation of the median for two experiments performed on different days. For 3-input circuits, inputs correspond to the absence or presence of 1 mM IPTG (top -/+), 2 ng/mL aTc (middle -/+), and 5 mM L-arabinose (bottom -/+). For the Demultiplexer circuit (Demux), inputs correspond to the absence or presence of 1 mM IPTG (top -/+) and 2 ng/mL aTc (bottom -/+).*

IV. Debugging genetic circuits

We developed a strategy for "debugging" a malfunctioning circuit to determine which gate is causing the failure. This was done by creating a series of plasmids that transcriptionally fuse the output promoter of each gate to *yfp*. These constructs are carried on a plasmid with a pSC101 origin of replication and a spectinomycin resistance marker. This is co-transformed with the circuit plasmid (the plasmid containing the output promoter of the circuit is not included) and the cells are grown and assayed in the same way that the circuit is characterized (Methods). This is done in two steps. First, a single screen is performed on all gates and all combinations of input conditions. From this, it can be seen which gates are failing to respond as expected. Then, the screen is followed up with more detailed measurements including replicates that focus on the failed gate. A summary of these experiments is shown in Fig. S21, which shows the subset of data highlighting the failure discovered. From these data, the gate where the problem originates can be deduced and the impact as the error propagates through the circuit observed. The most prevalent failure mode appears linked to the P_{Tet} promoter, an effect we observed in seven cases (0xF9, 0x06, 0x9F, 0xB9, 0x19, 0x36, and 0xC1). We also saw a repeated failure associated with the use of the AmtR gate (Figure S21: 0x98 and Figure 5c: 0xC9).

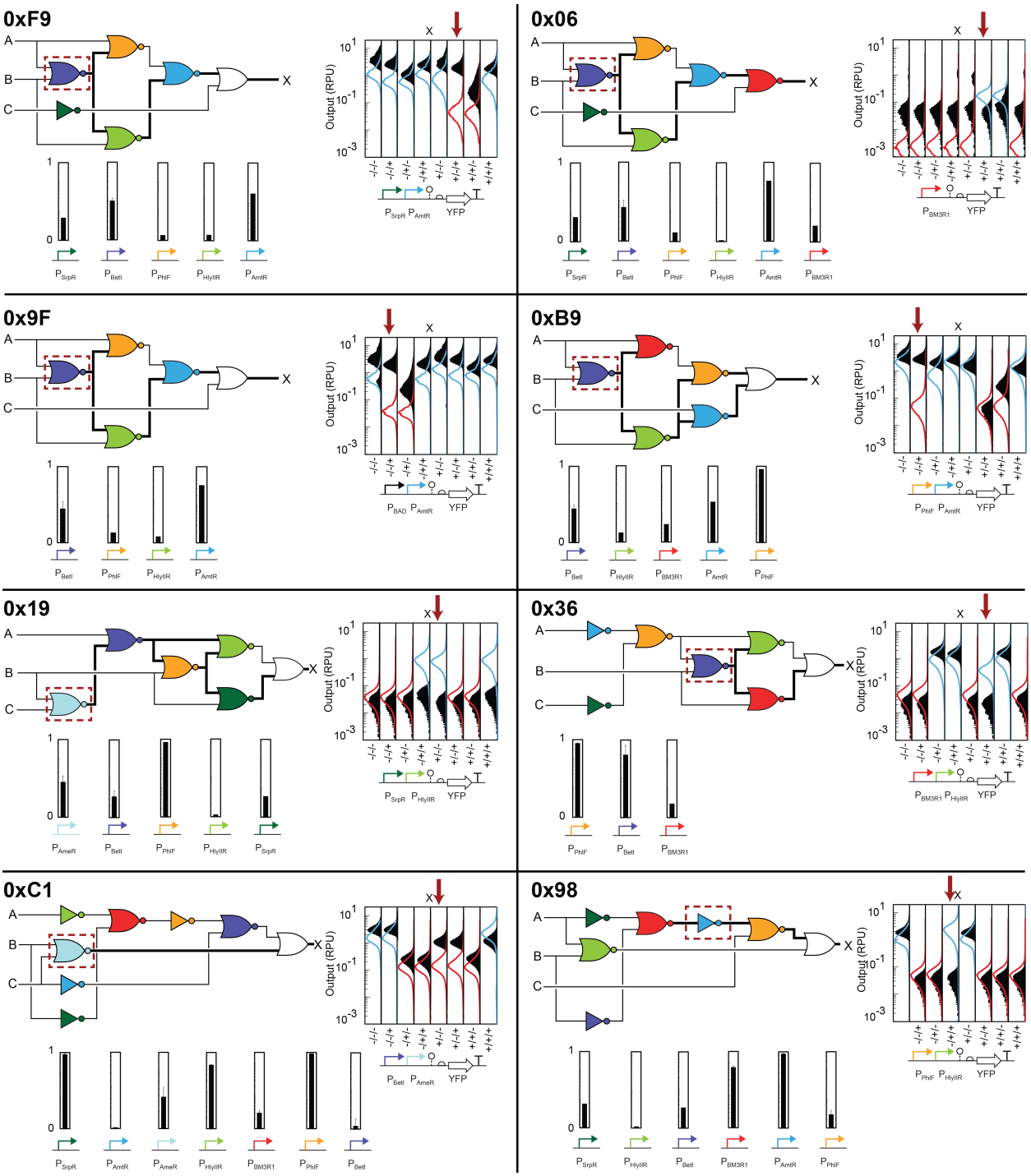

Figure S21: ***Experiments to determine gate failures internal to a circuit.*** *Data are shown for 8 of the non-toxic circuits that show at least one failed state. In each case, an initial screen was performed where the debugging plasmids are substituted for the output plastid (shown under the cytometry plot) and screened under all eight combinations of inputs. The bar graphs correspond to the activity of each promoter for the failed state under investigation (e.g., +/-/- for 0xC1, red arrow). To account for the dynamic range differences of the gates, the fluorescence measured is normalized by the minimum and maximum fluorescence observed for the debugging plasmid across all circuits and states. This allows the reporting of the gate activity in the range [0,1]. The dashed red box shows where the error initiates and the thick black line shows how it propagates to the circuit output.*

V. Cello Software

This section has two purposes: (1) to describe each algorithm used by Cello in detail, and (2) to provide a "user manual" for how to write code describing a circuit and interpret the software output. The Cello software provides a design automation environment whose input is a high-level specification from a hardware description language (Verilog) and the output is a DNA sequence that implements the circuit (Figure S22). Alternatively, the output can be a Eugene file that is used by combinatorial design algorithms to build a library of constructs.

The first step is to parse the Verilog code to compute the truth table. The truth table is the starting point for logic synthesis, which generates the circuit diagram. An assignment algorithm then selects gates from the UCF to be used in the circuit. Combinatorial design then generates the linear DNA sequence and places it into the physical context, defined in the UCF. Simulations are run to predict the performance of the circuit. Figure S22 shows an overview of each step of Cello described in this section. Details regarding the organization of the UCF are provided in Supplementary Section VII.

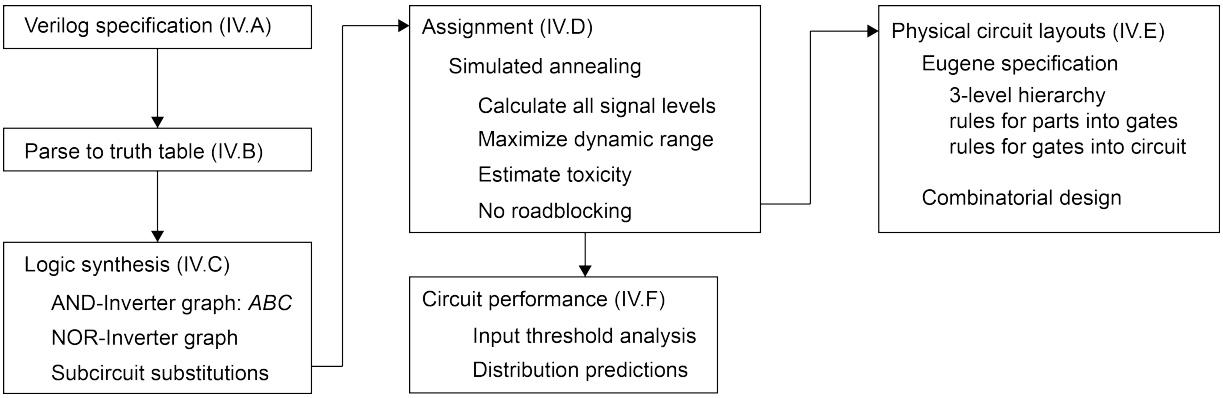

Figure S22: ***Overview of the Cello software.*** *The Cello input is a high-level logic specification written in Verilog, a hardware description language. The code is parsed to generate a truth table, and logic synthesis produces a circuit diagram with the genetically available gate types to implement the truth table. The gates in the circuit are assigned using experimentally characterized genetic gates. In assignment, a predicted circuit score guides a Monte Carlo simulated annealing search. The assignment with the highest score is chosen, and this assignment can be physically implemented in a combinatorial number of different layouts. The Eugene language is used for rule-based constrained combinatorial design of one or more final DNA sequence(s) for the designed circuit.*

V.A. Specification: Verilog hardware description language

A subset of Verilog is synthesizable, meaning the program can be directly mapped to a physical implementation in hardware. Synthesizable Verilog is transformed to a netlist (a list of connected primitive gates that can be mapped to a hardware technology), which is functionally equivalent to the Verilog code. The subset of Verilog used in Cello is described in this section.

Verilog module. Verilog is written using modules, where each module has a name, and the line defining the module name also requires input definitions and output definitions. The following box defines a module named "example" with output "x", and inputs "a" and "b". Keywords are shown in blue.

```
module example(output x, input a, b);

endmodule
```

Assign statement. Within a Verilog module, Cello accepts and parses assign statements, case statements, and structural statements. An assignment provides a concise way to specify a combinational logic function. Assign statements use an = operator to set the value of a wire on the left-hand side based on the wire values and logic operators on the right-hand side.

```
module example(output x, input a, b);

  assign x = a & b;

endmodule
```

Additional Verilog operators that can be used in assign statements are:

a & b	a AND b
a | b	a OR b
~a	NOT a

The following statement uses multiple operators.

```
module example(output x, input a, b, c);

  assign x = a & b | ~c;

endmodule
```

The order of operations proceeds from left to right. Parenthesis can be used to specify a different order of operations to implement a different function.

```
module example(output x, input a, b, c);

  assign x = a & (b | ~c);

endmodule
```

More complex assign statements can use internal wires to carry values within the module. To use internal wires, the names must be defined, and they must be assigned (appearing on the left-hand side of the equation) before they can be used as an operand on the right-hand side. The function above can also be implemented using internal wires.

```
module example(output x, input a, b, c);

  wire w1, w2;

  assign w1 = ~c;

  assign w2 = b | w1;

  assign x = w1 & w2;

endmodule
```

Case statement. A case statement provides a way to specify a truth table in Verilog. Since all combinational logic functions can be represented as a truth table, the case statement can be used to specify any combinational logic function as input to Cello. A case statement is placed within an "always" block. An always block contains a "sensitive list", meaning the always block executes the code within the begin/end keywords whenever a value changes for a member of the sensitive list. The sensitive list below contains in1 and in2.

```
module example(output out, input in1, in2);

always@(in1, in2)

    begin

    end

endmodule
```

The case statement is placed within the begin/end lines within the always block. The line case({in1,in2}) indicates that the argument of the case statement is {in1,in2}. In Verilog, the brackets indicate concatenation, meaning the argument for the case statement is one value that is the concatenation of in1 and in2. If in1 is 0 and in2 is 1, the argument would be 01.

```
module example(output out, input in1, in2);

always@(in1, in2)

    begin

      case({in1,in2})

      endcase

    end

endmodule
```

The actual cases within the case statement are specified using a bit-wise numbering system: 2'b01: {out} = 1'b0. This individual case executes when the argument is a 2-bit number in binary notation (2'b) equal to 01. When this case executes, the value 0 for a 1-bit number in binary notation (1'b) is assigned to the wire named "out". By specifying all combinations of input values as individual cases, a complete truth table can be specified. The following example specifies the truth table for a 2-input AND operation.

```
module example(output out, input in1, in2);

always@(in1, in2)

    begin

      case({in1,in2})

        2'b00: {out} = 1'b0;
        2'b01: {out} = 1'b0;
        2'b10: {out} = 1'b0;
        2'b11: {out} = 1'b1;

      endcase

    end

endmodule
```

More complex case statements can be used to specify n-input m-output truth tables, such as the multiple output truth table for the priority circuit (Figure 4A).

```
module example (output x, y, z, input a, b, c);

always@(a, b, c)

    begin

      case({a,b,c})

        3'b000: {x,y,z} = 3'b000;
        3'b001: {x,y,z} = 3'b001;
        3'b010: {x,y,z} = 3'b010;
        3'b011: {x,y,z} = 3'b010;
        3'b100: {x,y,z} = 3'b100;
        3'b101: {x,y,z} = 3'b100;
        3'b110: {x,y,z} = 3'b100;
        3'b111: {x,y,z} = 3'b100;

      endcase

    end

endmodule
```

The names of multiple output wires are concatenated within brackets, so the concatenated value of xyz equals 000 in the first case, equals 001 in the second case, and so on. Due to concatenation within brackets, the order of names matters in {a,b,c} and {x,y,z}. However, the order of names in the sensitive list does not matter.

always@(a, b, c) is the same as always@(c, b, a)
{x, y, z} is different than {z, y, x}
case({a, b, c}) is different than case({c, b, a})

Structural statement. Assign statements and case statements are forms of "behavioral" Verilog, meaning that the function is specified without considering a gate-level schematic. Structural Verilog can be used to directly specify the desired circuit topology using the same form as a netlist, which specifies a gate type, the output wire name, followed by the input wire names.

```
nor (x, a, b);
```

The above example specifies the function x = a nor b. Note that gate types are specified in lowercase in Verilog. Each gate can only have a single output, but can have multiple inputs. Allowed gate types include: not, or, nor, and, nand, xor, xnor, buf. For example, the following specifies the function x = a and b and c.

```
and (x, a, b, c);
```

To use structural elements within a Verilog module, the above lines just need to be written within a Verilog module:

```
module example (output x, input a, b, c);

   and (x, a, b, c);

endmodule
```

Internal wires can be used to build up more complex structural statements. The next example also implements 3-input AND logic, but uses a combination of four NOT gates and two NOR gates:

```
module example (output x, input a, b, c);

  wire w1, w2, w3, w4, w5;
  not (w1, c);
  not (w5, b);
  not (w4, a);
  nor (w3, w4, w5);
  not (w2, w3);
  nor (x, w1, w2);

endmodule
```

Even though structural Verilog can be used to specify a wiring diagram, logic synthesis is used to convert certain primitive gate types might not be available in the genetic gates library and to minimize number of gates in the circuit, if possible.

Combining Verilog statements. Explanations and examples of Verilog case statements, assign statements, and structural statements provided above were limited to one type of statement per module. However, these forms can also be combined in a module to build more complex programs. An example is provided below that combines the following commands:

Define a module name, input wire names, and output wire names:
    module example(output out, input a, b, c);
Initialize the internal wire names that will be required to carry values within the module:
    wire w1, w2, w3, w4;
Assign: Let w1 carry the value of the logical operation a AND c:
    assign w1 = a & c;
Assign: Let w2 carry the value of the logical operation (NOT a) AND (NOT c):
    assign w2 = ~a & ~c;
Structural: Define a NOR gate with output wire w3 and input wires w1 and w2:
    nor (w3, w1, w2);
Structural: Define a NOT gate with output wire w4 and input wire w3:
    not (w4, w3);
Case: Use a case statement to define a truth table for a 2-input AND function with inputs w4 and b, and output out. Members of the sensitive list are w4 and b, so the begin/end block will execute when w4 or b changes value. The argument for the case statement is the concatenated value of w4 and b. Only set the value of wire out to 1 when the concatenated value of w4 and b equals 11.
End the Verilog module:
    endmodule

```
module example(output out, input a, b, c);

wire w1, w2, w3, w4;
assign w1 = a & c;
assign w2 = ~a & ~c;
nor (w3, w1, w2);
not (w4, w3);
always@(w4, b)
    begin
      case({w4,b})
        2'b00: {out} = 1'b0;
        2'b01: {out} = 1'b0;
        2'b10: {out} = 1'b0;
        2'b11: {out} = 1'b1;
      endcase
    end

endmodule
```

V.B. Parsing Verilog to generate a truth table

Section V.A explained the syntax for writing Verilog code. All combinational logic functions can be expressed in the form of a truth table, which is the entry point to logic synthesis. In this section, we describe how the Verilog program is parsed to a naïve netlist (list of connected gates), and how the naïve netlist is used to generate a truth table.

The first Verilog line is the module definition, which is parsed to obtain the input names and output name(s). From there, individual assign, structural, and case statements are parsed from the Verilog file. Each individual statement is converted to a logic node that can contain a single gate, multiple gates, or a truth table.

Assign. A line starting with the **assign** keyword indicates an assign statement, which is parsed to a tree data structure in which input wire names are the leaf nodes, the output wire name is the root node, logic operators (~, |, &) are the internal nodes, and parentheses inform the branching. This tree is functionally equivalent to a circuit diagram, which is used as a logic node with one or more logic gates.

Structural. A line starting with the lowercase name of a gate type (**not, nor, or, and, nand, xor, xnor**) indicates a structural statement, which is parsed to a single-gate logic node of that type, where the first argument indicates the node's output wire name, and all subsequent arguments indicate input wire names.

Case. An always block containing a **case** keyword is parsed to a truth table, where the wire name within curly brackets (for example, {out}) is the output wire name of the node, and the case argument (for example, {w4,B}) indicates the input wire names. Gate types are not used in the logic node parsed from a case statement; instead, the truth table is used to relate output values to the input values of the node.

Connecting all nodes according to the input/output wire names results in a graph that can be used to propagate logic through each node to calculate the truth table specified by the input Verilog (Figure S23).

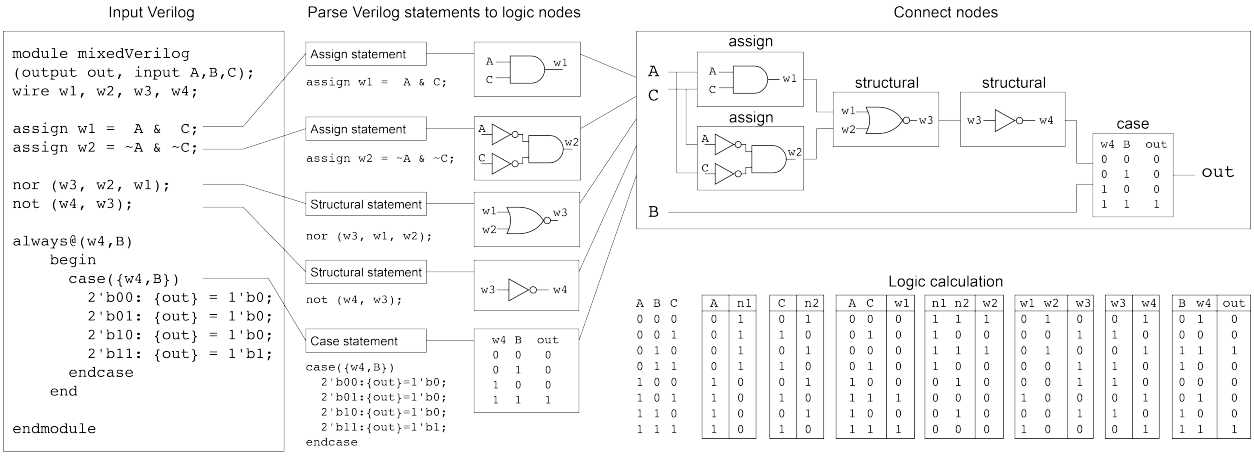

Figure S23: ***Flow of Verilog parsed to a truth table.*** *A Verilog file is parsed into individual assign, structural, and case statements. Each statement is converted into a logic node, which can contain one or more gates, or a truth table. Logic nodes are connected by matching input/output wire names, and Boolean logic is propagated through each node to compute the truth table of the circuit output.*

Nested Verilog modules. The above example used different types of Verilog statements in the same module. However, Verilog modules can also be nested to form more complex programs. In the module hierarchy, the referencing module is called the parent module, and the referenced modules are called child modules (Figure S24). This nesting implements the reuse of previously written modules, which is helpful when scaling up to larger logic programs.

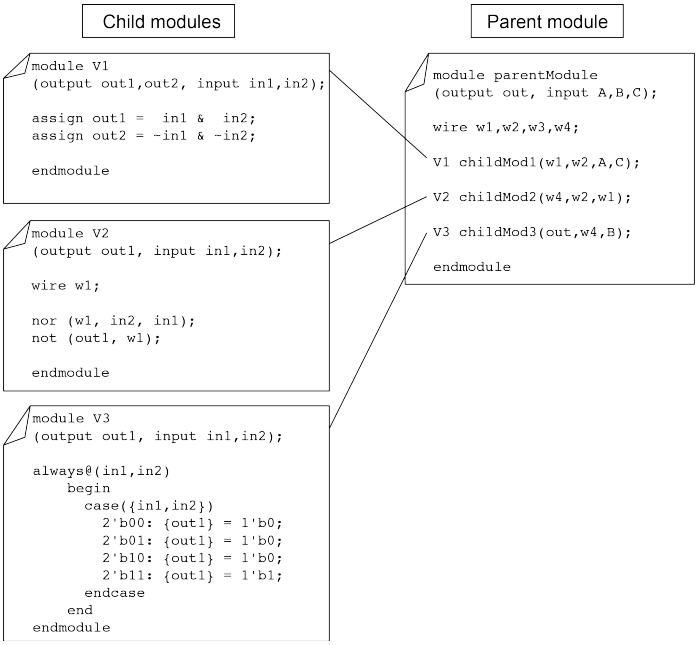

Figure S24: ***Nested Verilog modules for module reuse.*** *The same logic function shown in Figure S23 is rewritten using a parent module that references child modules. In the same flow, each statement is converted to logic nodes, which are used to generate the truth table specified by the full program. In Cello, the parent and child modules would appear as a single long file.*

V.C. Logic synthesis

The previous section describes how the Verilog code is parsed to create a truth table. This section focuses on the next step, which is to convert the truth table to a circuit diagram. This is a process known as logic synthesis and our approach relies on algorithms that are typically applied to electronic circuits, with additional steps to incorporate constraints that arise from working with a limited set of genetic gates.

Logic synthesis is performed in several steps (Figure S25). First, the truth table is converted to a NOR-Inverter Graph (NIG). Second, logic motifs can be swapped for equivalent subcircuits to reduce circuit size. Logic motifs can be stored and retrieved from the UCF to biasing the circuit toward particular motifs that are desirable given the biochemistries of the gates in the library.

To convert a truth table to a NOR-Inverter Graph (NIG), an intermediate step uses the logic synthesis tool ABC(*19*) to generate an AND-Inverter Graph (AIG) built exclusively from 2-input AND and NOT gates. ABC minimizes the number of gates (nodes) and layers (longest path) in the AIG. The AIG is converted to an NIG containing 2-input NOR and NOT gates. This conversion is done by replacing (A AND B) with the equivalent (NOT A) NOR (NOT B) according to DeMorgan's rule.

As an alternative to ABC, we also developed a path to the circuit diagram using Espresso(*20*), another commonly used tool for logic synthesis. This approach differs in that it first converts a truth table to a minimized Product of Sums (POS), which we then convert to an NIG. Both the ABC and Espresso routes are implemented in Cello and the one that produces the circuit diagram with the minimum number of gates is selected. In approximately 95% of the cases, the number of gates after the ABC route is less than or equal to the number of gates after the Espresso route.

The user may have preferred logic motifs that they would like to have in the circuit diagram, if possible. These could represent optimized combinations of gates (both ABC and Espresso are not guaranteed to find the global minimum) or motifs that are particularly robust for a given biochemistry. For example, the UCF we developed has a list of small 3-input 1-output motifs generated from brute-force enumeration (Methods). Additionally, this is a simple mechanism to introduce non-NOR logic functions for which genetic gates may be available. The Eco1C1G1T1 UCF motif library contains: (1) a 2-input OUTPUT_OR motif to replace a NOR-NOT subcircuit at an output, (2) a 2-input 1-output optimal XNOR motif, and (3) small 3-input 1-output NOR/NOT motifs.

An attempt to incorporate the user-defined circuit architecture motifs into circuit diagrams occurs during the final step of logic synthesis. Starting with an initial NOR/NOT circuit diagram, subcircuits are replaced with a set of user-defined motifs, if possible. This is performed by the following steps. First, all possible subcircuits in the initial circuit diagram with ≤ 4 input wires and 1 output wire are enumerated. This is done by visiting each gate's output wire, then performing a breadth-first search on the incoming wires and gates, proceeding until the circuit inputs are reached. During this search, unique subcircuits are added to a list. Second, the truth table for each subcircuit and each user-defined motif is evaluated. If a subcircuit and a motif have Boolean equivalence (also checking permuted input wire order), then the motif is substituted in place of the subcircuit. If multiple subcircuit/motif matches are found, the motif that reduces the number of circuit gates the most is used. Finally, each time a motif replacement is made, the replacement algorithm is performed again until no more replacements can be made.

Motifs in the library can use gate types other than NOR/NOT, such as AND, NAND, OR, XOR, or XNOR. To constrain the logic gates according to the number and types of gates available in the genetic gates library, a cost function is used during subcircuit substitution. The cost is the total number of gates in the circuit that exceed the gates available in the library. For example, if there are 6 NOR/NOT gates and 1 AND gate in the library, and the circuit has 7 NOR/NOT gates and 2 AND gates, the cost would evaluate to (7-6) + (2-1) = 2. If there are enough available gates in the library to cover the gates in the

circuit, the cost is 0. A substitution is rejected if the cost increases, and is accepted if the cost decreases or does not change. This cost evaluation guides logic synthesis to produce a circuit that can be covered by the gates library. However, after subcircuit substitution converges and no more substitutions are possible, if the cost is still greater than 0, the circuit is reported as "not synthesizable".

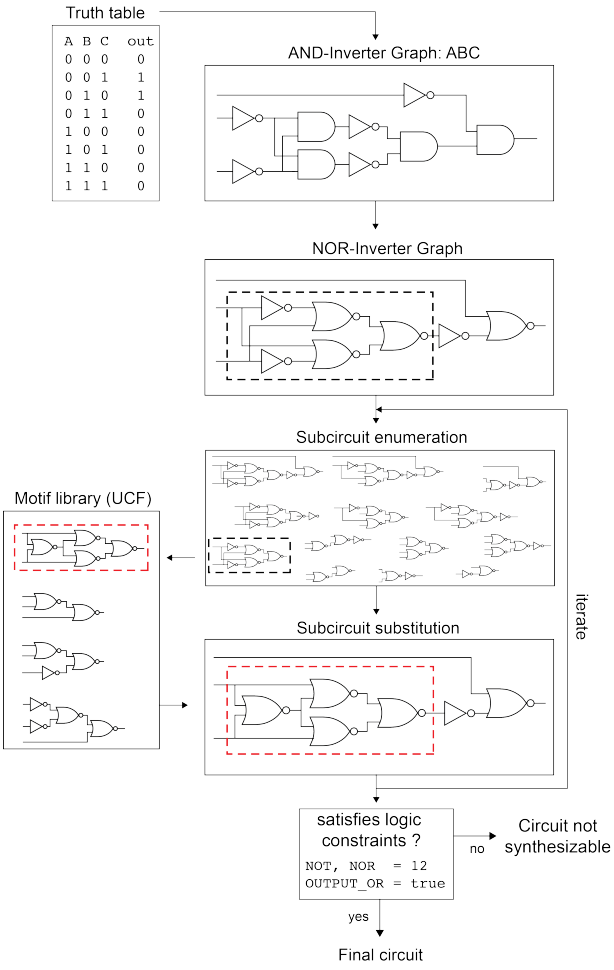

Figure S25: ***Logic synthesis workflow.*** *The starting point is a truth table. The AIG is converted to an NIG using DeMorgan's rule: (A AND B) equals (NOT A) NOR (NOT B), and removing double NOT gates. Subcircuits in the initial circuit diagram can be substituted for user-defined logic motifs specified in the UCF. The black dashed box highlights one subcircuit from the initial circuit, and the red dashed box indicates a functionally equivalent motif from the library, which is substituted into the circuit. This process is done iteratively until no more substitutions are identified. The logic constraints are determined by the gate types available in the UCF, and the number of instances of each gate type in the UCF. In this example, there are a maximum of 12 NOR/NOT gates, and any number of OUTPUT_OR gates.*

V.D. Repressor assignment

The previous section describes how the circuit diagram is generated. The next step is to assign genetic regulators to the gates in the diagram. Each gate is based on a unique biochemistry and thus generates a different response function. The assignment problem is to identify the optimal way to select and connect these gates to generate the maximum overall dynamic range for the circuit. In this section, we first describe how we score a particular repressor assignment. Next, the search algorithm is described that optimizes the assignment.

One approach to the assignment problem would be to permute all possible combinations of gates and identify the one that generates the best circuit. This would guarantee the identification of the global optimum. However this method becomes intractable as circuit size and library size grow. The number of unique assignments (with a single RBS variant per gate) is given by $r!/(r-g)!$, where g is the number of gates in the circuit and r is the number of repressors in the library. With our library of repressors (including RBS variants), a 9-gate circuit has ~10^{11} permutations. A search algorithm needs to be implemented to scale to larger circuits and libraries, but often comes with the tradeoff of introducing stochasticity into the search and can converge on local optima.

Calculating the circuit score. The circuit score S captures how closely the logic function generated by a repressor assignment matches the desired truth table for the circuit. Because the output of genetic circuits is not digital, the ON and OFF states have numerical values and a larger difference between these values (the dynamic range) is desirable. Calculating S requires two steps. First, the output is calculated for all combinations of input states. An example is shown in Figure S26, where there are two sensors and four input states. The activity of the sensors feeds into the gates and their response functions (Equation S1) are used to calculate how the signal propagates through the circuit. Then, S is calculated by comparing the lowest output for a state that should be ON and the highest output for a state that should be OFF:

$$S = \frac{\min(ON)}{\max(OFF)} \tag{S2}$$

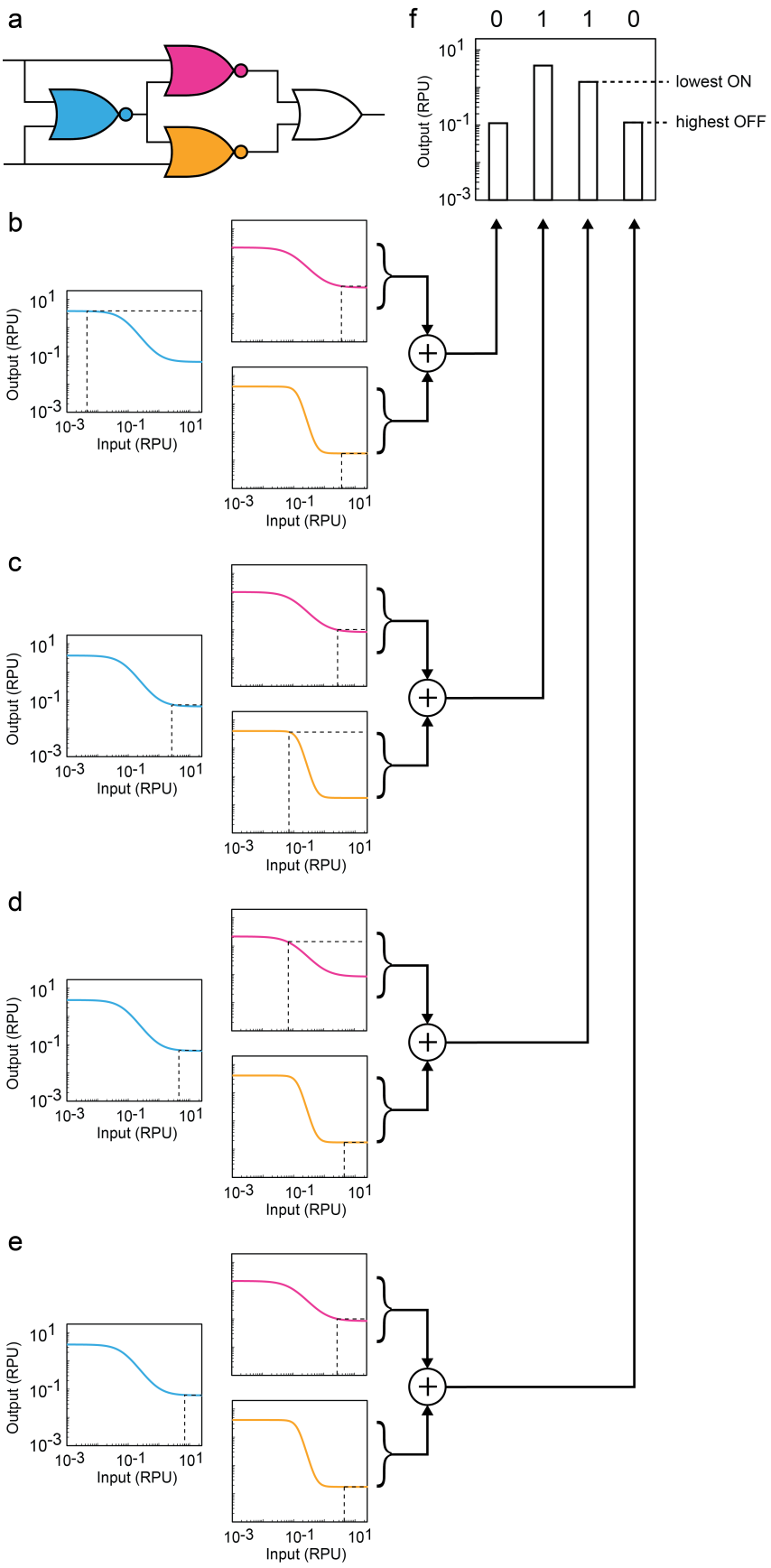

Figure S26*: Circuit score calculation.** ***(A) *Circuit diagram for an XOR circuit with gate assignments AmtR (blue), IcaRA (magenta), and PhlF (orange).* ***(B)*** *Visualization of signal propagation for each of four input states. Colored curves are gate response functions (Equation S1) with the same coloring scheme from (A). Dashed vertical lines represent promoter input levels for the gate. Dashed horizontal lines represent promoter output levels. The "+" symbol indicates promoter outputs from the IcaRA and PhlF gate are summed at the terminal OR gate.* ***(C)*** *Predicted output levels for each of the four input combinations. The 0s and 1s at the top of the graph indicate the desired truth table behavior for each output. The lowest ON state and highest OFF state are marked, and the ratio of these values is the circuit score, S.*

Calculation of predicted circuit toxicity. For each gate, normalized cell growth is measured as a function of input promoter activity (Figure S10). For a circuit, certain input states can lead to the expression of multiple repressors and this can lead to toxicity. For each gate in a circuit, the input RPU is calculated, and the cell growth value is interpolated from the two nearest experimentally-measured normalized cell growth values from the UCF. The toxicity of the whole circuit for a particular input combination is calculated as the product of normalized cell growth for each of the individual gates. There is no theoretical basis for this; rather, it was chosen to strongly bias against circuits where any repressors are expressed beyond their empirical toxicity threshold. After the toxicities of all the input states are calculated, the toxicity of the circuit as a whole ("growth score") is taken as the worst input state.

As shown in Figure S27 for the Majority circuit, there is a trade-off between the circuits with the highest circuit score (*S*) and those that are at risk of reducing growth, creating a Pareto-optimal curve. The current algorithm applies a cutoff (0.75) with respect to the growth score and only allows circuit assignments that fall above the cutoff.

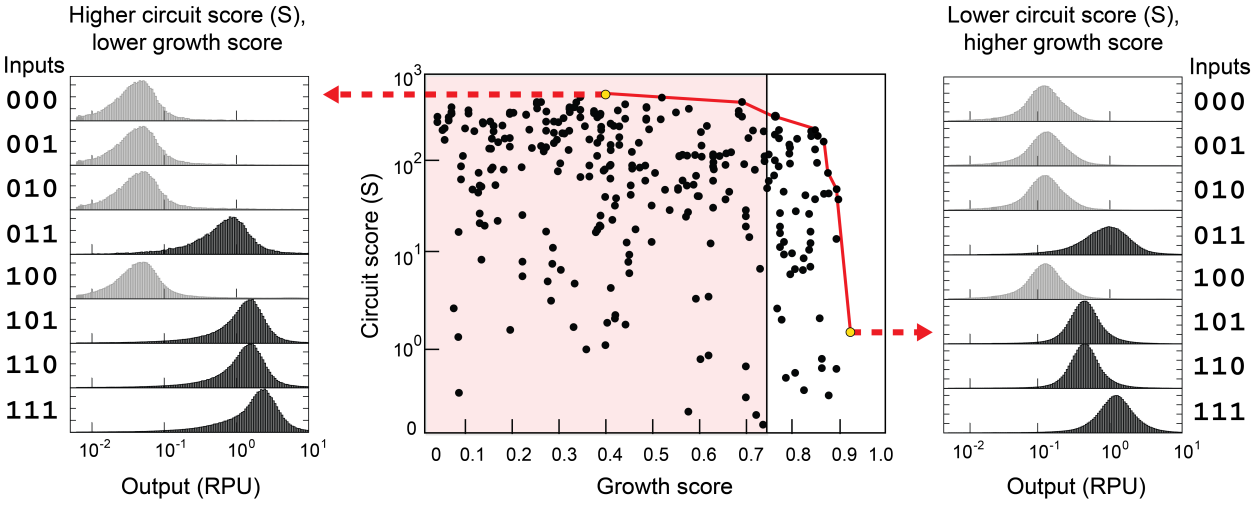

Figure S27: ***Tradeoff between circuit score and predicted cell growth.*** *For the Majority circuit (Figure 5), each assignment has a circuit score (*S*) and a growth score (represented as a point on the scatter plot). The Pareto frontier is shown as a red line. A threshold is defined to eliminate toxic assignments from consideration (shaded region in center plot). Left (assignment highlighted yellow in center plot): Prediction of assignment with high S but toxic expression of IcaRA. Assigned gates: P2-PhlF, H2-HlyIIR, A2-AmtR, B3-BM3R1, I1-IcaRA, S4-SrpR. Right (assignment highlighted yellow in center plot): Prediction of assignment with normal cell growth but low S. Assigned gates: B3-BM3R1, F1-AmeR, S4-SrpR, A2-AmtR, P2-PhlF, H2-HlyIIR.*

Simulated Annealing Assignment Algorithm. The goal of repressor assignment is to find the combination of gates that maximizes the circuit score, *S* (Equation S2). The repressor assignment problem has a large discrete search space for which we implemented a Monte Carlo simulated annealing algorithm(*21, 22*) to identify an optimum assignment. The search initializes with gates from the library being randomly chosen and assigned to a gate in the circuit. Any gate can be assigned to any position in the circuit. Each iteration of the Monte Carlo algorithm swaps the assignments of two gates. This is done by randomly selecting one gate in the circuit, randomly selecting a second gate either in the circuit or in the gate library, and then performing the swap. After the swap, the circuit score for the new assignment *S'* is calculated and the move is accepted with a probability based on the score change and the temperature factor *T*:

$$P = e^{-\left(\frac{S-S'}{T}\right)} \qquad \text{(S3)}$$

After calculating the probability, a random number *R* between 0 and 1 is generated: if *R* < *P*, the swap is accepted, and if *R* > *P*, the swap is rejected. If the swap improved *S*, then *P* > *1*, and the move is always accepted. After the first assignment is initialized, the probability of accepting a move decreases as the temperature anneals with exponential decay:

$$T_i = T_{max} \cdot e^{-Ci} \quad \text{(S4)}$$

where *i* is the current iteration, T_{max} is the starting temperature, and *C* is a constant that determines the rate of cooling. After reaching the end of annealing, the run continues at *T* = 0 until 10,000 steps progress with no additional improvement. The simulated annealing results in Figure S28 show convergent solutions for the circuits ranging from 5 to 9 gates, where T_{max} = 100, and *C* is 5 x 10^{-5}.

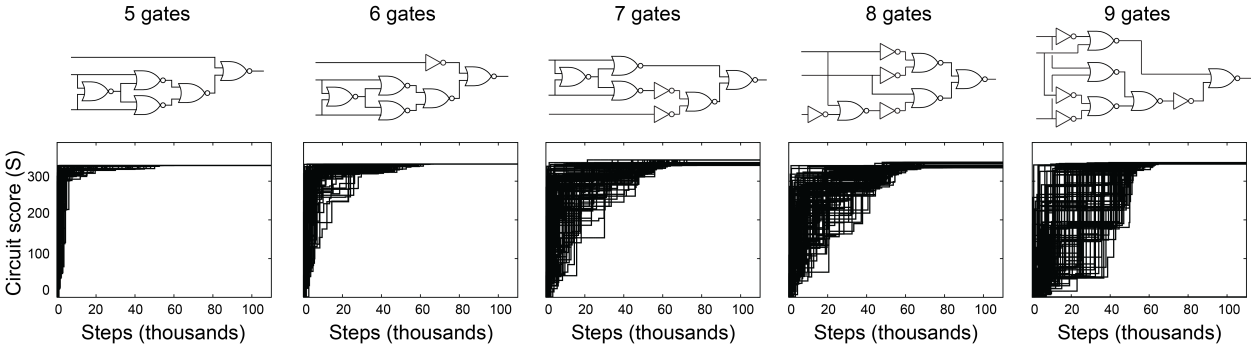

Figure S28: ***Simulated annealing search algorithm for repressor assignment.*** *Each plot shows 100 trajectories, and as the number of steps increases for each trajectory, the highest S up to that point is plotted (black lines). The temperature factor annealed according to the schedule listed above.*

Several modifications were made to the basic algorithm described above to allow for additional constraints that do not appear in the *S* calculation. Some gates have multiple RBS options, but a repressor cannot be used more than once in a circuit. To prevent illegal swaps that reuse a repressor, a list of gates that can be legally swapped is generated. Gates with the same repressor as the selected gate are allowed in the list, because this swap simply replaces the RBS for the gate. Gates with a different repressor are only allowed in the list if another gate from the circuit does not use the same repressor. To avoid repressor reuse, gate group names are also specified in the UCF (Supplementary Section VII, *gates* collection). The group name will typically be the repressor name, but different repressors can also be grouped if they exhibit cross-talk.

Additional constraints can be applied to reject assignments that would otherwise be accepted based on *S*. For example, we have implemented the rejection of assignments whose growth score is below a threshold (previous sub-section) or when two "roadblocking" promoters have to be connected as inputs to a gate (Supplementary Section II.B).

V.E. Combinatorial design of circuit layouts

After a gate assignment has been found for a circuit diagram, a linear DNA sequence that contains the complete circuit is generated. This is done using combinatorial design(*23*, *24*), which has been applied to build DNA sequences using a set of parts, constraints between parts, and organizational rules. The assignment algorithm leads to a list of parts in the circuit as well as constraints between parts (*e.g.*, due

to roadblocking). The UCF can also contain additional organization rules, such that the repressors have to appear in a specified order and orientation. After the assignment algorithm, the parts and rules for a circuit are automatically used to build a Eugene file. From this Eugene file, combinatorial algorithms described previously(*23*, *24*) are used to build the DNA sequence, which is the output of Cello. The Eugene file itself is also an output of Cello so that it can be run at a later time to generate additional constructs (https://cidar.bu.edu/EugeneLab/).

One of the advantages of using combinatorial design is that many constructs can be built that preserve the same underlying rules–and therefore produce the same circuit function–but unconstrained aspects of the design are allowed to vary. Before the user runs Cello, an option is available to specify the number of desired constructs. Building and testing a library of designs instead of a single design can help identify a functional variant. Additionally, identifying failed and successful designs provides a data set for learning new organizational rules(*23*). An example of this are the Majority circuits in Figure 5e, where multiple constructs are shown.

This section describes how gates and their component parts are organized in Eugene as well as the impact of adding organizational constraints to the UCF. A hierarchical design is used to describe circuits in Eugene (Level 1: Parts, Level 2: Gates, and Level 3: Circuit).

In Level 1, part types are defined, and individual parts with those types are defined. Parts in Eugene require a type and a name, while other attributes such as DNA sequence can be added optionally. Below are examples of part definitions for a promoter, ribozyme, RBS, CDS and terminator that make up a gate.

Level 1: Part type definitions

```
PartType Promoter;
PartType Ribozyme;
PartType RBS;
PartType CDS;
PartType Terminator;
```

Level 1: Part definitions

```
Promoter pTac;
Ribozyme RiboJ53;
RBS P3;
CDS PhlF;
Terminator ECK120033737;
```

In Level 2, we assemble parts into gate devices (a device is defined as a collection of parts). The gate device contains the ribozyme insulator, RBS (sometimes multiple variants are allowed), repressor, and terminator. For a NOR gate, there can be two additional undefined promoters. For example, the device for the PhlF gate (with the P3 RBS) is as follows.

Level 2: Gate device

```
Device PhlF_device(
       Promoter, Promoter, RiboJ53, P3, PhlF, ECK120033737
);
```

We then define a set of rules that act on the P3_PhlF device. These rules define the promoters that drive PhlF according to the circuit diagram, and an enforced order of those promoters (e.g., to avoid

roadblocking). The ALL_FORWARD rule just orients all parts in the forward direction (this gate will be allowed in the reverse direction in a later step). Note that rules on different lines must be joined with the AND keyword.

Level 2: Gate rules

```
Rule PhlF_rules
(ON PhlF_device:
      CONTAINS pBM3R1 AND
      CONTAINS pHlyIIR AND
      pBM3R1 BEFORE pHlyIIR AND
      ALL_FORWARD
);
```

Given the devices and rules for each gate, an enumeration of variants for each device is performed using the 'product' function. The PhlF device shown above only allows a single device variant.

Level 2: Design of gate variants

```
PhlF_devices   = product(PhlF_device);
SrpR_devices   = product(SrpR_device);
BM3R1_devices  = product(BM3R1_device);
HlyIIR_devices = product(HlyIIR_device);
BetI_devices   = product(BetI_device);
AmtR_devices   = product(AmtR_device);
```

In Level 3, gate device variants will be combined into the circuit device. First, we must initialize the circuit device and each gate device, where each gate is named by the repressor to allow rules from the UCF to be applied according to that name.

Level 3: Initializing the circuit device, and its component devices.

```
Device circuit();

Device gate_PhlF();
Device gate_SrpR();
Device gate_BM3R1();
Device gate_HlyIIR();
Device gate_BetI();
Device gate_AmtR();
```

Rules are applied to the circuit device before enumerating circuit variants. The EXACTLY 1 counting rule ensures that each gate appears once and only once in the circuit. Additional rules can also be specified, for example requiring the PhlF gate to be in the first position and in the forward orientation, and requiring each gate to alternate orientation, as follows.

Level 3: Circuit device rules

```
Rule circuit_rules
(ON circuit:
       gate_PhlF   EXACTLY 1 AND
       gate_SrpR   EXACTLY 1 AND
       gate_BM3R1  EXACTLY 1 AND
       gate_HlyIIR EXACTLY 1 AND
       gate_BetI   EXACTLY 1 AND
       gate_AmtR   EXACTLY 1 AND
       STARTSWITH gate_PhlF AND
       FORWARD gate_PhlF AND
       ALTERNATE_ORIENTATION
);
```

Now that we have specified the circuit rules, the combinatorial design step can be performed. Nested for-loops iterate through all gate device variants from Level 2 (there might only be a single variant for each gate, or there might be variants with different promoter orders). In the innermost loop, the current set of gate device variants is used to build the circuit device in Level 3 in the 'permute' function. Each set of designs in the inner loop is appended to an array called 'allResults'.

Level 3: Design of circuit variants

```
Array allResults;

for(num i0=0; i0<sizeof(PhlF_devices);   i0=i0+1) {
for(num i1=0; i1<sizeof(BetI_devices);   i1=i1+1) {
for(num i2=0; i2<sizeof(SrpR_devices);   i2=i2+1) {
for(num i3=0; i3<sizeof(HlyIIR_devices); i3=i3+1) {
for(num i4=0; i4<sizeof(AmtR_devices);   i4=i4+1) {
for(num i5=0; i5<sizeof(QacR_devices);   i5=i5+1) {

gate_PhlF   = PhlF_devices[i0];
gate_BetI   = BetI_devices[i1];
gate_SrpR   = SrpR_devices[i2];
gate_HlyIIR = HlyIIR_devices[i3];
gate_AmtR   = AmtR_devices[i4];
gate_QacR   = QacR_devices[i5];

Device circuit(
   gate_PhlF,
   gate_BetI,
   gate_SrpR,
   gate_HlyIIR,
   gate_AmtR,
   gate_QacR
);

result = permute(circuit);

allResults = allResults + result;

}}}}}}
```

Next, we explain how Eugene rules specify the design space of allowed circuit layouts (Figure S29). At the gate device level (Level 2), the only degree of freedom is promoter order within each gate. NOT gates can only have 1 variant. NOR gates can have two variants, where either promoter order is allowed. This degree of freedom allows 2^N variants, where N is the number of 2-input gates. Enforcing roadblocking rules (STARTSWITH) constrains promoter order, but for tandem non-roadblocking

promoters, either promoter order is still allowed (Figure S29, Gate devices).

In addition to promoter order, gate order/orientation is the other degree of freedom. At the circuit device level (Level 3), if the repressor order is constrained, and all gates are in the forward orientation (as specified in Eco1C1G1T1), then the only degree of freedom is promoter order from Level 2. In the example circuit assignment, repressor order and all forward rules result in 4 solutions (Figure S29, Panel 1).

Additional variants can be generated by removing order/orientation rules. For example, removing the FORWARD gate_PhlF rule allows the reverse orientation of the PhlF gate and results in 8 solutions (Panel 2). Removing a second rule, gate_PhlF BEFORE gate_BetI, now allows the PhlF gate in any position and results in 40 solutions (Panel 3). Removing all remaining gate order rules (a BEFORE b) allows unconstrained shuffling of gate order and results in 960 solutions (Panel 4). Removing all remaining FORWARD rules allows all gate orders and orientations and results in 15,360 solutions (Panel 5).

The Eugene rules specified in the UCF (Supplementary Section VII, *eugene_rules* collection) include roadblocking rules in Level 2 gate devices, and repressor order and all forward rules in the Level 3 circuit device. As described above, these rules can be removed to unconstrain the layout design space. Furthermore, any additional rules from the Eugene language (*25*) can be added to the UCF for user-specified constraints to the design space. The number of desired variants can be specified in the Options tab of the Cello web application. After layout design using Eugene, the Cello output has three forms. The first output is a file containing an ordered list of part names and part orientations (+, -) for each variant. The second output is a file containing an ordered list of gate names and gate orientations for each variant. The third output is a separate plasmid file for each variant in which the circuit module is inserted into the specified genetic location (Section VII, *genetic_locations* collection).

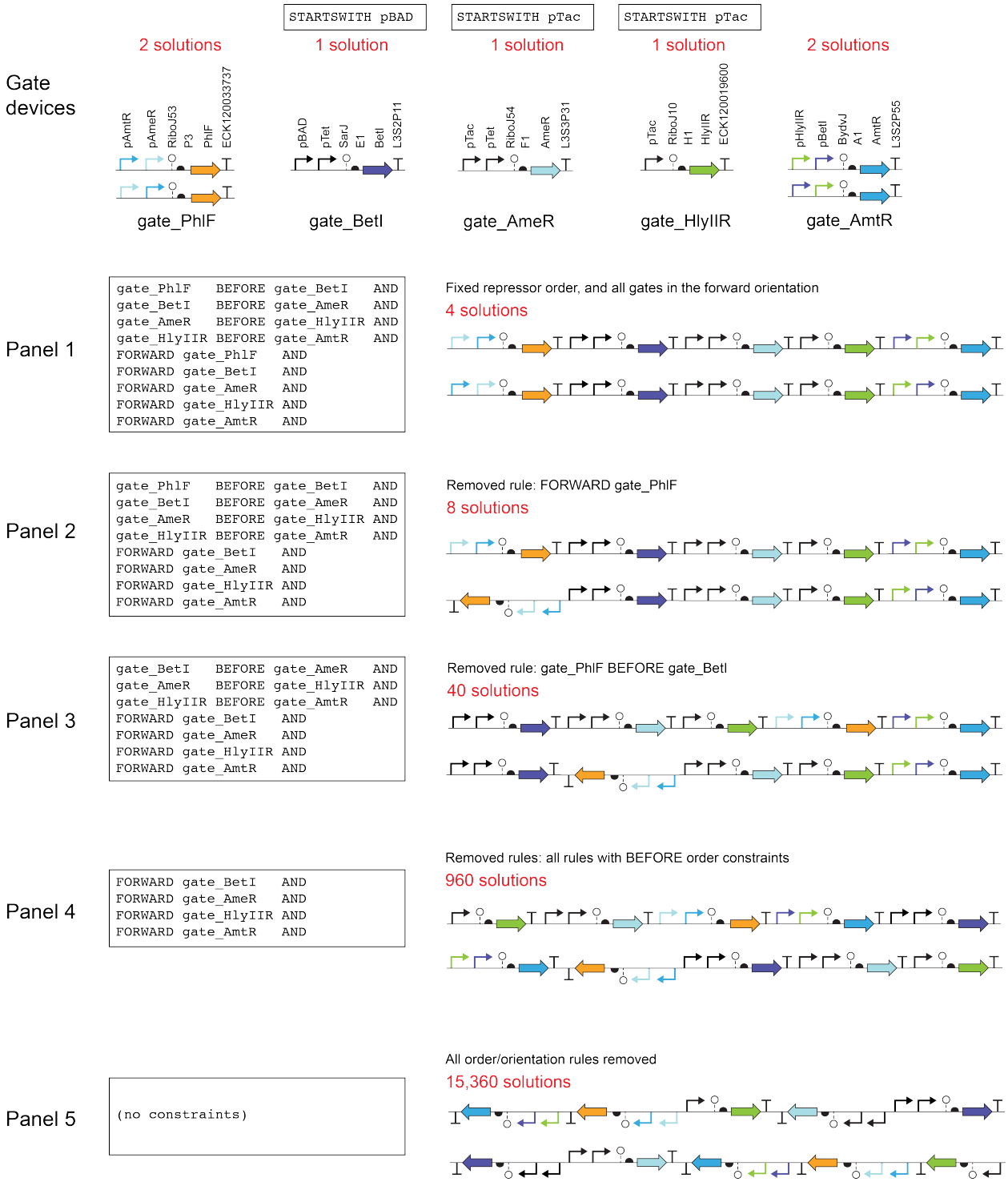

Figure S29: ***Example sets of Eugene rules for the Majority circuit.*** *Parts (Level 1) are used to build gate devices. Promoter order rules are used to disallow roadblocking promoters in the downstream position. The resulting gate devices (Level 2) can be composed in to a circuit device (Level 3). The rules in this figure defined for a circuit device would be specified in "circuit_rules" block in Eugene file, in addition to the EXACTLY 1 gate assignment rules. Depending on the set of rules, the design space for this 5-gate circuit ranges from 4 to 15,360 layouts.*

V.F. Predictions of circuit performance

Qualitative predictions of the circuit output distributions can be computed for each input combination. This is performed as a final step in Cello, after the gate assignment search has converged. In order to perform this step, each gate in the circuit must have experimental cytometry distributions added to the UCF, with fluorescence values converted to RPU (Supplementary Section VI.C).

As a first step, the experimentally-measured gate output RPU histograms are normalized to have a total of 10,000 events in evenly log-spaced bins (one bin every $10^{0.024x}$ RPU). At least 8 experimentally-measured output RPU histograms at various input RPU levels are required (the gates in this work use 12 input levels). Histograms *Y(x)* are generated at intermediate input levels *x* by positioning their medians, *<Y(x)>*, on the gate's response function (Equation S5, with parameters fit according Supplementary Section I.D). Once the medians are in place, the counts *f* for each bin are interpolated from the counts (f_L and f_R) of the experimentally-measured histograms that lie to either side of *x* (Equation S6). The experimentally measured histograms have input values, x_L and x_R. The parameter *m* is the bin relative to the median, *y/<Y>*.

$$\langle Y(x)\rangle = y_{min} + \frac{(y_{max}-y_{min})K^n}{x^n+K^n} \tag{S5}$$

$$f(m) = f_L(m) + (f_R(m) - f_L(m))\frac{x-x_L}{x_R-x_L} \tag{S6}$$

In this way, histograms at intermediate inputs are generated with medians on the response function, and shapes interpolated from the nearest experimental histograms.

Once all the gate distribution response functions are computed, the qualitative predictions for output distributions can be computed. For a particular input combination, sensor values feed into the first layer of gate distribution response functions (dashed vertical lines, Figure S30). This input value takes a vertical "slice" of the distribution response function to create the output histogram. Next, those gate output histograms become input histograms for the second layer of gates.

Input histograms can be viewed as 10,000 individual input events, each of which produces its own output histogram—all of which are averaged to produce a histogram containing 10,000 events. At NOR and OR gates, input histograms are combined by first summing the histogram medians (or summing the histogram median and the sensor input value if a sensor promoter is an input), then shifting both histograms to be centered at that new median, and then averaging the counts in each bin to create a histogram with 10,000 events.

Sensor input signals are propagated through gate distribution response functions in the circuit until the output histograms are produced for each input row in the truth table (Figure S30c).

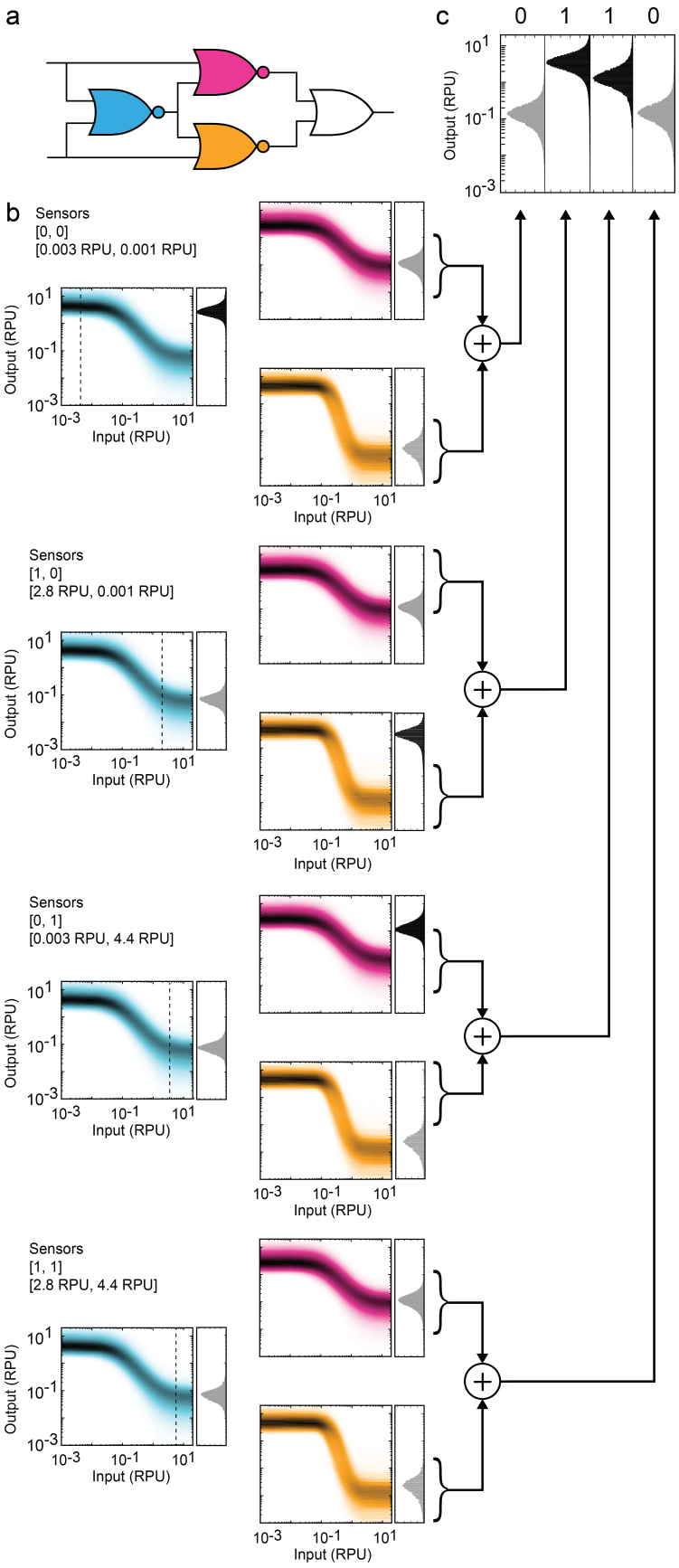

***Figure S30**: **Circuit distribution calculation.** **(A)** Circuit diagram for an XOR circuit with gate assignments AmtR (blue), IcaRA (magenta), and PhlF (orange). **(B)** Visualization of distribution propagation for each of four input combinations. Colored curves are gate distribution response functions with the same coloring scheme from (A). Dashed vertical lines represent sensor input levels for the gate. Vertical histograms to the right of each response function are the output histograms for the gate. The "+" symbol indicates the output histograms from the IcaRA and PhlF gate are summed at the terminal OR gate. **(C)** Predicted output histograms for each of the four input combinations. The 0s and 1s at the top of the graph indicate the desired truth table behavior for each output. Black and gray histograms are expected to be high and low, respectively.*

Input threshold analysis. Genetic gates output a continuous range of values as opposed to digital 0s and 1s. This is similar to electronic systems, where output voltages also take continuous values. For digital abstraction in electronic design, a maximum value for low-inputs and a minimum value for high-inputs specify the input ranges that produce outputs considered to be ON and OFF(*26*). If an input signal falls between a gate's low/high thresholds, this can lead to an intermediate output or an incorrect ON/OFF output if the input is perturbed slightly. This section describes how we define and use input thresholds in Cello.

For two NOT gates connected in series, the low and high output levels of the first gate (*OL* and *OH*) must map onto either side of the low and high input thresholds of the second gate (*IL* and *IH*). The difference between *IL* and *OL* is the low margin (*ML*), and the difference between *OH* and *IH* is the high margin (*MH*). Both margins must be positive to satisfy the input threshold criteria. A negative margin indicates an input that falls in the sensitive intermediate zone, the region between *IL* and *IH* (Figure S31a, diagonal hatching).

In electronic design, the low and high input thresholds are identical for all gates in a circuit. However, each genetic gate has unique thresholds due to their different response functions. Each gate's *IL* and *IH* thresholds are calculated from its response function using the input levels that cause the gate output to be 0.5x the maximum and 2x the minimum output, respectively (Figure S31a, black dots). *ML* and *MH* must be positive values for a gate connection to be valid, and all gate connections must be valid for the circuit as a whole to be valid. As an example, in the Majority circuit (Figure 5) the assignment had a good predicted circuit score, but the assignment did not satisfy all input margins. Specifically, the PhlF gate has a predicted input RPU that falls in the intermediate region (Figure S31b). Experimentally, the initial design for this circuit (Figure S15) had one high OFF state caused by the PhlF gate input falling in the sensitive intermediate region.

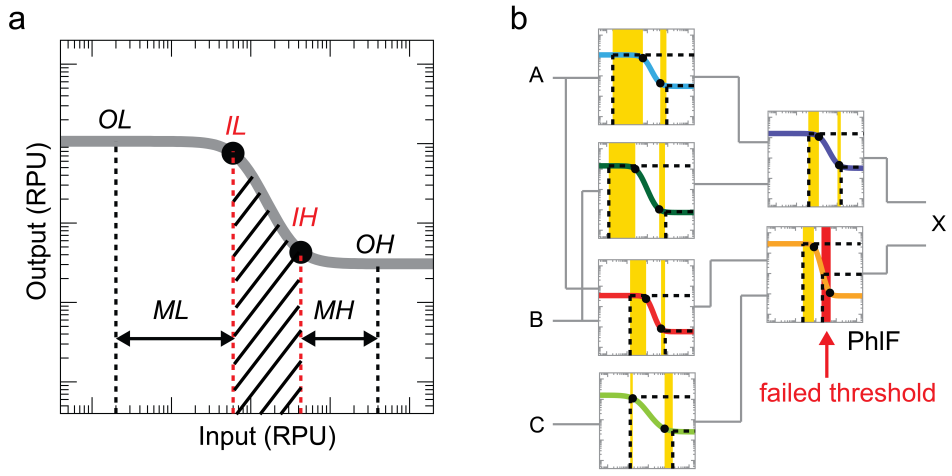

Figure S31: ***Input threshold analysis.*** ***(A)*** *A low threshold and high threshold for a gate (IL, IH) are used to determine valid levels for inputs to that gate. In this example, the outputs from the previous gate (OL, OH) result in positive margins (ML, MH, horizontal arrows). A negative margin would indicate an input level in the forbidden zone (diagonal hatching).* ***(B)*** *Input threshold analysis for the Majority circuit (output of Cello). Yellow regions indicate positive margins that pass input threshold criteria. The red margin for PhlF (P3 RBS) indicates a negative input margin that fails the IH threshold criterion.*

VI. Characterization of sensors and gates for use with Cello

Different sensors and actuators can be connected to the circuits built by Cello. To make use of this, a user has to characterize the output promoter of their sensor(s) in the genetic context defined for the circuit (in the UCF). The sensor is characterized in two states, ON and OFF, under the conditions defined by the user. For example, it could be two different concentrations of a small molecule or the presence and absence of an environmental stimulus. This measurement has to be provided to Cello in standard units (RPUs). The DNA containing any regulators necessary for sensor function (referred to as the "sensor block") could either be uploaded to appear in the circuit plasmid or separately inserted into a different context (*e.g.*, the genome). Sensor blocks are not necessarily required; for example, when the promoter depends only on native regulators. The connection of the output to a new actuator is more straightforward, where it simply requires knowing whether its dynamic range is sufficient to trigger a phenotypic response. A step-by-step guide for the characterization of new sensors such that they can be used with Cello is provided in this section.

Similarly, building new UCFs requires the characterization of new gates and/or gate libraries in different organisms or operating conditions. The procedure is similar to characterizing sensors and is also provided in this section. The details of the data organization for the gate library in the UCF are provided in Supplementary Section VII.

VI.A. Measurement of RPU standard

Sensor and gate fluorescence characterization data must be converted into relative promoter units (RPUs) for incorporation into Cello. To convert characterization data to RPUs, an RPU standard plasmid must first be measured along with an autofluorescence control.

1. ***Transform the RPU plasmid to create an RPU standard strain and a non-YFP plasmid to create an autofluorescence control strain.*** Following work by Kelly et al. (*27*), the promoter activities must be reported to Cello in standard units. The standard plasmid used for the Eco1C1G1T1 UCF is shown in Figure S32. New UCFs may define different standards. This is the same backbone as the circuit plasmid (Figure 1b) and the gate measurement plasmid (Figure S36), and contains the same YFP expression cassette. Note that while the standard constitutive promoter (BBa_J23101 (*28*)) is the same, this plasmid is different from the Kelly standard (*13*, *27*), including an upstream insulating terminator, an upstream promoter spacer, and a different RBS. Our RPU plasmid produces 4.2x the YFP signal as the Kelly standard. The plasmid should be transformed into the strain defined by the UCF, which for the Eco1C1G1T1 UCF is *E. coli* NEB 10-beta (New England Biolabs, MA, C3019).

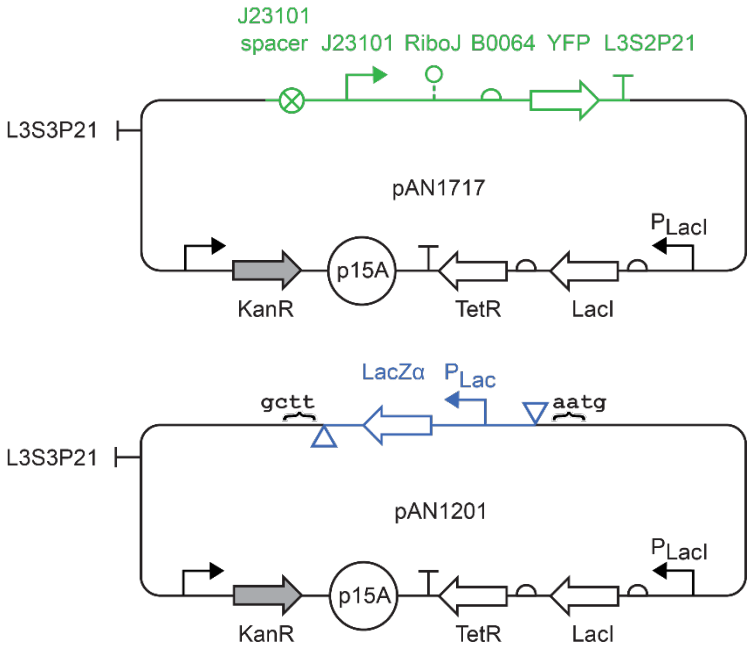

Figure S32: ***RPU standard plasmid and autofluorescence control.*** *Part sequences are provided in Table S9. The RPU plasmid (top) promoter is J23101 (28). The RiboJ sequence is the same as published previously (2), with an additional upstream cloning scar. The RBS is B0064 (29). The YFP is as published previously (30), with three synonymous mutations: C153A, C564A, and G606T. The YFP terminator is L3S2P21 (6). There is a 15 bp spacer upstream from J23101, and the upstream terminator L3S3P21 (6) insulates the YFP cassette from transcriptional readthrough from the plasmid backbone. Cells transformed with pAN1201 (bottom) are used to measure autofluorescence.*

2. ***Measure the fluorescence of the RPU standard strain and the autofluorescence control strain.*** For each set of inducer conditions to be applied to a circuit, gate, or sensor, YFP fluorescence measurements are collected for the RPU strain and the autofluorescence strain. Fluorescence can be measured using flow cytometry, a plate reader, or any instrument capable of measuring YFP fluorescence. The measurements should be made under media and growth conditions that are as close as possible to that defined in the UCF.

We performed additional characterization of the RPU standard to estimate the RNAP flux on the promoter J23101. The RPU standard plasmid was measured using smFISH (*31*) to obtain the rate of mRNA transcription from the P_{Tac} promoter. A background control and a measurement plasmid with an inducible promoter were also measured to generate a standard curve to determine the steady state number of *yfp* mRNAs per cell. We quantified the steady state number of *yfp* mRNA copies per cell at mid-exponential growth using single-molecule fluorescence *in situ* hybridization (smFISH) (*31*) following the method given in (*32*), which we adapted for counting *yfp* mRNA in *E. coli* at single-transcript resolution. Briefly, we designed a set of 25 oligonucleotide probes, fluorescently labeled with TAMRA, each 20 bases in length, against the *yfp* transcript (Table S10) using Stellaris Probe Designer version 4.1. The mean *yfp* mRNA copy number for the RPU standard plasmid (pAN1717) strain was found to be 24.7 ± 5.7 molecules per cell (Figure S33). The half-life for *gfp* mRNA in *E. coli* has been determined to be approximately 2 minutes (*33*), and the average plasmid copy number for the p15A origin of replication has been determined to be approximately 15 per cell (*34*). Therefore, the estimated mRNA production rate is 0.30 ± 0.07 mRNAs per second per cell or 0.02 ± 0.005 mRNAs per second per plasmid for the

pAN1717 J23101 promoter. Stated errors are the standard deviation from replicate measurements and do not include any estimate of systematic bias of the measurement. For comparison, the mRNA production rate for the J23101 promoter was estimated to be approximately 0.03 mRNA per second per DNA copy in (*27*).

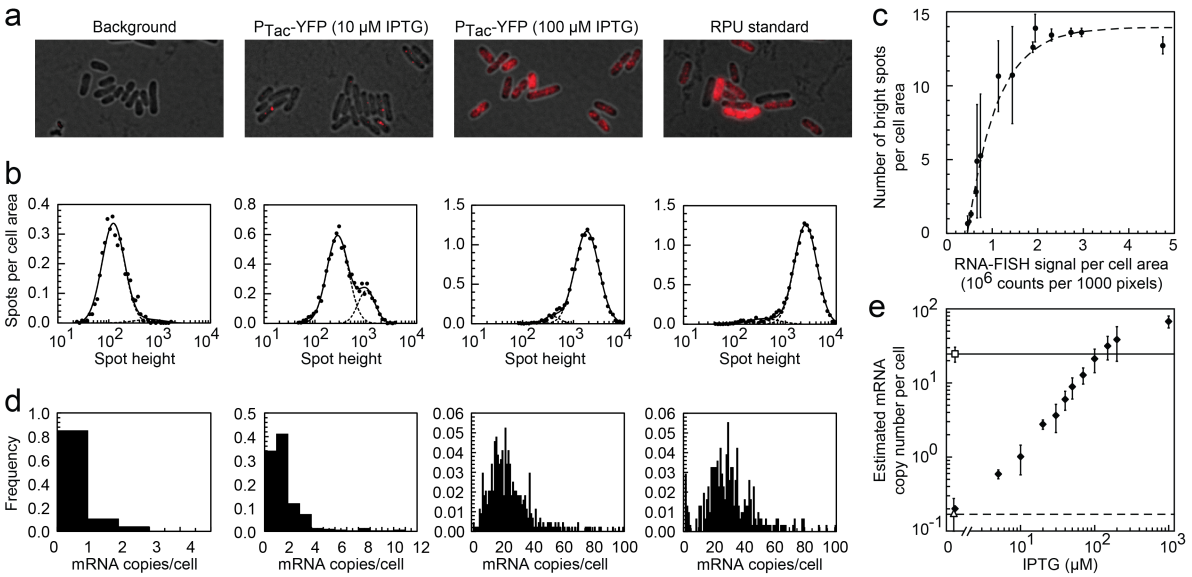

Figure S33: ***smRNA-FISH measurement of the RPU standard. (A)*** *Merged fluorescent and brightfield micrographs of background (pAN1201), P_{Tac}-YFP (10 μM IPTG, pAN1818), P_{Tac}-YFP (100 μM IPTG, pAN1818), and the RPU standard (pAN1717).* ***(B)*** *Histograms of spot heights for the strains shown in (A). Y-axis units are number of spots per 1000 pixels. Histograms were fit to a sum (solid line) of two log-normal distributions (dashed lines).* ***(C)*** *Number of bright spots per cell area vs. FISH signal per cell area for one example replicate experiment. X-axis and y-axis units are both per 1000 pixels. Number of bright spots was estimated from the integrated area of the 2nd (brighter) log-normal distribution fit from (B). The error bars are the standard errors from that estimate. The dashed line is the fit result using a Poisson filling process model (Methods).* ***(D)*** *Histograms of estimated mRNA copy number per cell for the strains shown in (A). Estimates were obtained from the RNA-FISH signal for each cell using the linear extrapolation of the fit curve shown in (C).* ***(E)*** *Estimated mean mRNA copy number per cell vs. IPTG concentration for P_{Tac}-YFP (filled diamonds), background (open triangle), and RPU standard (open square). Plotted values and error bars are the averages and standard deviations of three replicate measurements of the mean mRNA copy number. Dotted and dashed lines represent copy number estimates for background and the RPU standard strains, respectively. The estimated mRNA copy number per cell for background is consistent with zero and the estimated mRNA copy number per cell for the RPU standard is 24.7 ± 5.7.*

VI.B. Sensor characterization

Sensors convert signals (small molecules, light, etc.) into a transcriptional output. This section provides the steps required to characterize a sensor and report its output in standard units (RPUs). First, the output promoter needs to be cloned into the circuit plasmid defined by the UCF. Next, the sensor responses are characterized in the strain of interest and parallel measurements of the RPU standard's fluorescence and the strain's autofluorescence are made (previous section). These data are used to calculate sensor output RPU values, which are input into Cello along with the sequence of the output promoter and sequence of the sensor block.

3. ***Construct the plasmid to measure the sensor output promoter.*** The sensor is characterized in the same backbone as the circuit in Figure 1b (Figure S34, bottom). A version is provided that contains a P_{Lac}-lacZα module used for blue/white screening (Figure S34, top). The sensor output promoter is used to drive a YFP expression cassette, which matches the RPU standard. This cassette includes a ribozyme insulator (RiboJ) and the same RBS-YFP-terminator set as the RPU standard plasmid. The transcriptional fusion between the promoter and expression cassette should be scarless (note that a 4 bp scar is defined at the 5'-end of each ribozyme insulator that can be used for cloning). To clone the entire cassette into the plasmid, BbsI Golden Gate sequences flanking the P_{Lac}-lacZα module simplify cloning, but do not have to be used.

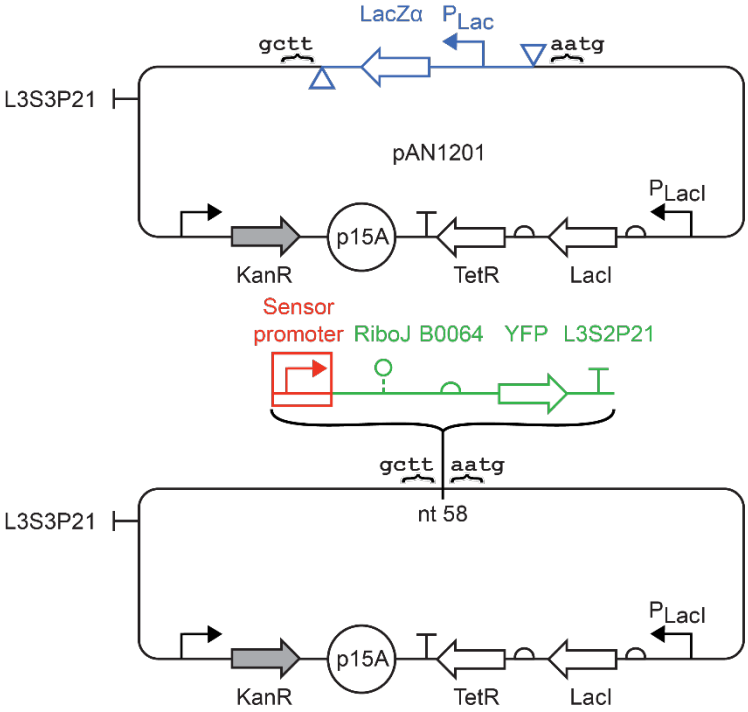

Figure S34: ***Constructing a sensor measurement plasmid.*** *A sensor promoter (red) is positioned in front of the YFP RPU cassette (RiboJ-B0064-YFP-L3S2P21, green) to create a sensor measurement plasmid (bottom). This module is inserted into pAN1201 (top, Supplementary Section IX), optionally using BbsI restriction enzyme recognition sites (triangles) and ligation into the insertion scars "gctt" and "aatg". The entire LacZα module (blue) is replaced upon successful insertion.*

4. ***Transform the sensor plasmid from #1 into the strain defined by the UCF.*** The Eco1C1G1T1 UCF defines the strain as *E. coli* NEB 10-beta (New England Biolabs, MA, C3019).

5. ***Characterize the ON/OFF state of the sensors.*** For each set of conditions, fluorescence measurements are made for three strains containing different plasmids: the sensor, the RPU standard, and empty plasmid for autofluorescence. The measurements should be made under media and growth conditions that are as close as possible to that defined in the UCF. The following equation converts the median YFP fluorescence to RPU:

$$RPU = \frac{\langle YFP \rangle - \langle YFP \rangle_0}{\langle YFP \rangle_{RPU} - \langle YFP \rangle_0} \quad \text{(S7)}$$

where *<YFP>* is the median fluorescence of the cells containing the sensor, $\langle YFP \rangle_{RPU}$ is the median fluorescence of the cells containing the standard plasmid, and $\langle YFP \rangle_0$ is the median autofluorescence.

Example: Characterization of sensors (IPTG, aTc, arabinose)

We demonstrate the characterization of the three sensors used in this manuscript. Three plasmids were constructed (Figure S35) to individually test the output promoters that respond to IPTG (P_{Tac}), aTc (P_{Tet}), and arabinose (P_{BAD}). The same sensor block was used containing the necessary regulators (LacI, TetR, and AraC*). We measured each of these sensors in response to "low" and "high" input signals: the absence or presence of 1 mM IPTG, 2 ng/mL aTc, and 5 mM L-arabinose respectively. The extraction of median fluorescence from the cytometry plots and conversion into RPUs are shown in Table S6.

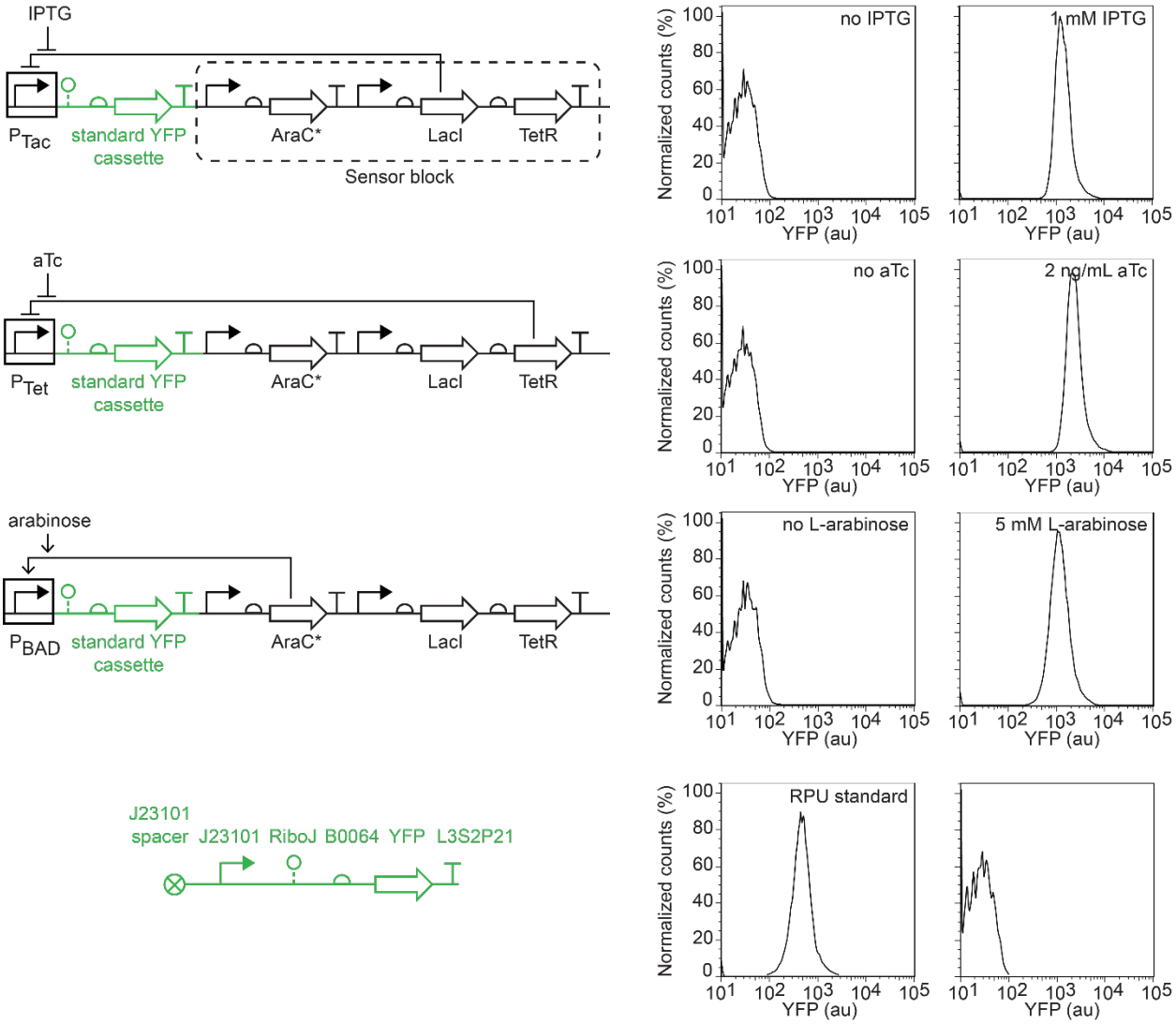

Figure S35: ***Fluorescence data for IPTG-, aTc-, and arabinose-sensors, the RPU standard, and autofluorescence.*** *DNA sequences for these sensor measurement plasmids and RPU standard can be found in Supplementary Section IX (P_{Tac}: pAN1718, P_{Tet}: pAN1719, P_{BAD}: pAN1720, RPU standard: pAN1717). Experimental conditions can be found in Supplementary Section VIII.*

Table S6: Calculation of Sensor OFF/ON activities

		<YFP>	$\langle YFP \rangle_{RPU}$	$\langle YFP \rangle_0$	RPUs
P_{Tac}	OFF	16.6	475	15	0.0034
	ON	1300	475	15	2.8
P_{Tet}	OFF	15.6	475	15	0.0013
	ON	2020	475	15	4.4
P_{BAD}	OFF	18.8	475	15	0.0082
	ON	1170	475	15	2.5

VI.C. Characterization of gates to be included in the UCF

Characterizing gates is similar to sensors, with three differences. First, both the input and output of the gate are promoter activities. This requires a separate characterization of the input promoter that is used to characterize the gate. Second, a full response function is required for the gate, as opposed to simpler ON/OFF values. Finally, in order to qualitatively predict population behavior, Cello requires RPU distributions and these are more complicated to normalize to RPUs. The protocol for gate measurement is below.

1. ***Clone the gate into the measurement plasmid.*** For the Eco1C1G1T1 UCF, the input promoter to the gate is P_{Tac} and a LacI expression cassette is provided in the backbone of the same plasmid used to characterize sensors and the circuits (Figure 1b). The output is the same YFP expression cassette used for the RPU standard plasmid. The gate—which consists of a ribozyme insulator, RBS, gene, terminator, and the output promoter—is cloned between P_{Tac} and the standard YFP expression cassette on the circuit backbone that encodes a constitutively expressed LacI (Figure S36). The resulting construct allows the gate to be induced with IPTG, and the output to be measured by quantifying YFP with cytometry.

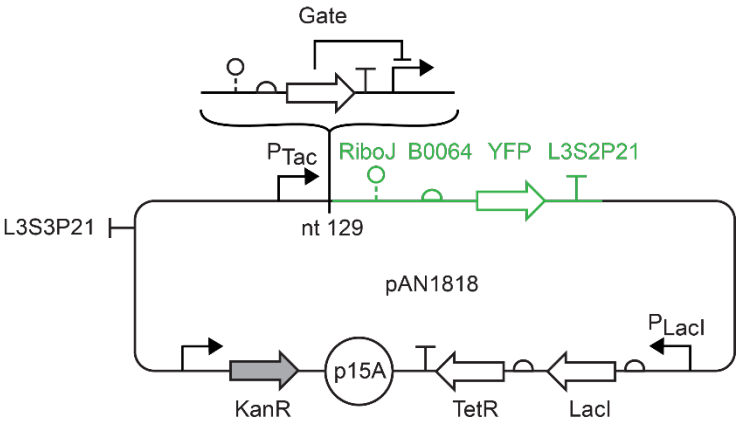

Figure S36: ***Constructing a gate measurement plasmid.*** *A gate comprising a ribozyme insulator, RBS, protein coding sequence, terminator, and output promoter is inserted between P_{Tac} and the YFP expression cassette with constitutively expressed LacI. The gate is inserted at nucleotide position 129 on pAN1818 (DNA sequence in Supplementary Information Section IX).*

2. ***Transform the gate plasmid from #1, the RPU standard plasmid, the autofluorescence measurement plasmid, and input promoter-YFP plasmid.*** Transform the gate plasmid (Figure S36), the same RPU standard plasmid and autofluorescence measurement plasmid for sensor measurement (Figure S32), and the P_{Tac}-YFP plasmid (Figure S37). The IPTG-inducible P_{Tac} promoter is used as the input to the gate. This allows the gate's response to different input promoter activities to be measured. The Eco1C1G1T1 UCF defines the strain as *E. coli* NEB 10-beta.

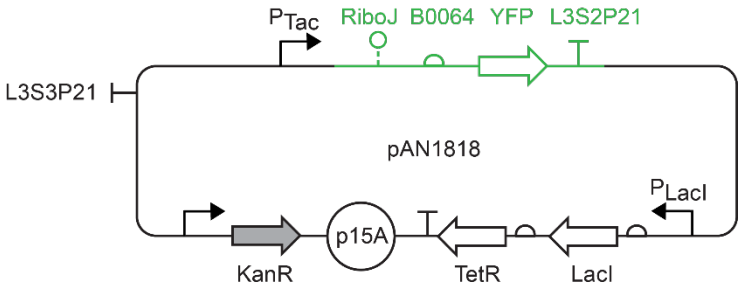

Figure S37: ***Plasmid to characterize the IPTG-inducible P_{Tac} input promoter.*** *The P_{Tac} promoter with symmetric LacO (71 bp, sequence as previous published(2), with an upstream spacer from -51 to -37) is positioned in front the standard YFP cassette with constitutively expressed LacI (pAN1818, Supplementary Section IX). The LacI promoter, RBS, and CDS are the same as in the E. coli genome(35), except the "GTG" start codon is replaced with "ATG". The LacI terminator is the genomic AraC terminator (TaraC) from the E. coli genome(35) (DNA sequences in Supplementary Section IX).*

3. **Characterize the fluorescence of cells carrying the gate plasmid, the input promoter-YFP plasmid, the RPU standard plasmid, and autofluorescence under a series of inducer concentrations.** Grow the cells according to the UCF growth specifications. Induce the input promoter with various concentrations of inducer; at least six inducer concentrations should be used so that the gate output evenly spans the entire output range. For each inducer treatment, calculate the median fluorescence for the sensor, RPU plasmid, and empty cells. Equation S7 in the previous section is used to convert the median YFP fluorescence to RPU.

4. ***Fit the input-output gate data to an equation capturing the response function.*** Step #3 results in a series of data generated by different concentrations of inducer that can be used to generate the response function of the gate. This is done by plotting the activity of the input promoter (using the plasmid in Figure S37) versus the activity of the output promoter of the gate (using the plasmid in Figure S36) at each concentration of inducer. Both of these measurements are first normalized to RPUs. Then, an equation describing the gate response is fit to the data to generate the response function used by Cello. This response function has the form of a Hill equation when the regulator is a repressor,

$$y = y_{min} + (y_{max} - y_{min})\frac{K^n}{x^n + K^n} \quad \text{(S8)}$$

where n is the Hill coefficient, K is the threshold, y_{max} and y_{min} is the maximum and minimum activity of the output promoter, and x is the activity of the input promoter.

5. **(Optional)** ***Convert the response function cytometry distributions to RPU.*** Fluorescence histograms can also be converted from arbitrary fluorescence units to RPU. This can be accomplished in a single step by taking the gate output histogram and multiplying all the fluorescence-axis values by the constant c:

$$c = \frac{\langle YFP \rangle - \langle YFP \rangle_0}{\langle YFP \rangle (\langle YFP \rangle_{RPU} - \langle YFP \rangle_0)} \quad \text{(S9)}$$

Effectively, this rescaling performs two transformations of the data: (1) multiplying the x-axis by $(\langle YFP \rangle - \langle YFP \rangle_0)/\langle YFP \rangle$ shifts the median of the gate's fluorescence distribution down in log-space by the autofluorescence median; and (2) division by $\langle YFP \rangle_{RPU} - \langle YFP \rangle_0$ normalizes the x-axis values to the RPU standard. These operations are visualized in Figure S38.

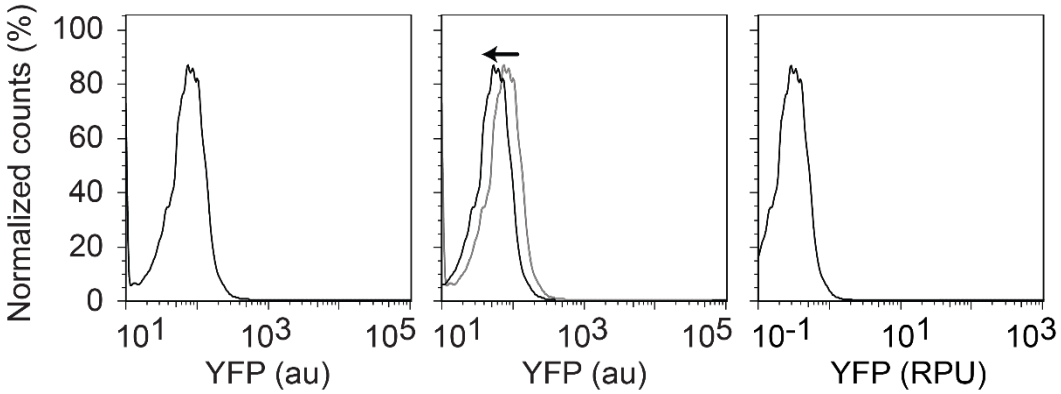

Figure S38: ***Converting a fluorescence histogram to RPU****. Beginning with a raw fluorescence histogram (black histogram, left panel; gray histogram, middle panel), shift it left (black arrow) by the autofluorescence median value <YFP>$_0$ = 15.0 au to generate the autofluorescence-corrected histogram (black histogram, middle panel). Next, divide the x-axis units by the corrected median RPU standard fluorescence <YFP>$_{RPU}$—<YFP>$_0$ = 460 au to convert the x-axis to RPU (right panel). The resulting RPU histogram can then be incorporated into a UCF.*

Example: Characterization of gate PhlF (with RBS-P2)

This section provides an example for the measurement of the PhlF gate with RBS-P2. We measured the YFP expression from the PhlF(RBS-P2) gate and the P_{Tac}-YFP plasmid in the following IPTG concentrations: 0, 5, 10, 20, 30, 40, 50, 70, 100, 150, 200, and 1000 μM. We also measured cellular autofluorescence and YFP expression using the RPU standard (pAN1717, Supplementary Section IX). The median fluorescence value for the RPU standard <YFP>$_{RPU}$ = 1540 and the median autofluorescence value <YFP>$_0$ = 17.4. The median fluorescence values for the PhlF gate and P_{Tac}-YFP were converted to RPU for each concentration of IPTG, and then the PhlF output RPU values were plotted against the P_{Tac}-YFP RPU values to generate the gate's response curve (Figure S39). Using values y_{max} = 4.1 RPU and y_{min} = 0.017 RPU, a Hill equation was fit to the data points and resulted in the fitted parameters K = 0.13 RPU and n = 0.92.

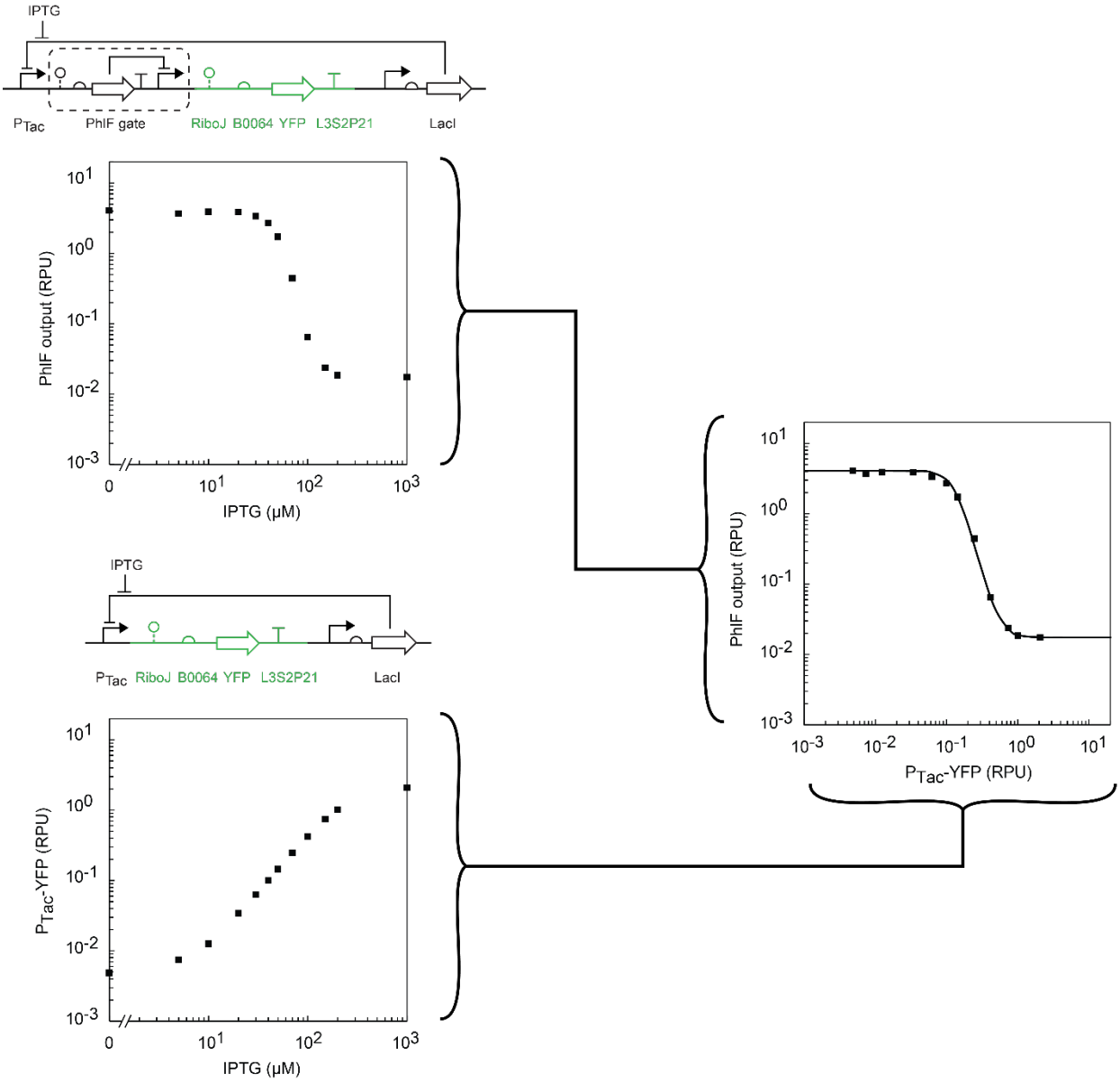

Figure S39: ***Measuring the response function for a gate.*** *RPU measurements at different IPTG concentrations for a gate PhlF(RBS-P2) (upper left) and the input promoter P_{Tac}-YFP (lower left) are plotted against each other to create the response function (right). A Hill equation is fit to the response curve (Equation S1, solid line). IPTG concentrations used were: 0, 5, 10, 20, 30, 40, 50, 70, 100, 150, 200, and 1000 μM. The DNA sequences for the PhlF gate plasmid (pAN1252) and the P_{Tac}-YFP plasmid (pAN1250) can be found in Supplementary Section IX.*

VII. User Constraint File (UCF)

VII.A. File overview

Verilog code is compiled to a circuit architecture that is defined by the user constraint file (Figure S40). This is a highly specified system that defines a particular library of gates as well as rules to be enforced for preferred logic motifs and genetic structure. In addition, it contains the definition of the particular strain and "landing pad" (*e.g.*, a defined set of plasmids or genomic locations) as well as the environmental conditions where the circuit models are valid. New UCFs can be developed for new gate libraries and/or strains and environments. While in practice a particular UCF may be valid for differing genotypes or changes in media/growth rate, out recommendation is that a new UCF file should be built for each end application.

This section describes the format of the UCF, as well as the specific Eco1C1G1T1 file used in this manuscript (and provided as Supplementary Data). Within the genetic gates library category, genetic parts and experimental data for the each gate are specified. The experimental measurements and associated standards for data in the UCF are described in Supplementary Section VI. This section focuses on the data structure of the UCF, with the intention of guiding the composition of files for new gate libraries, organisms, and operating conditions.

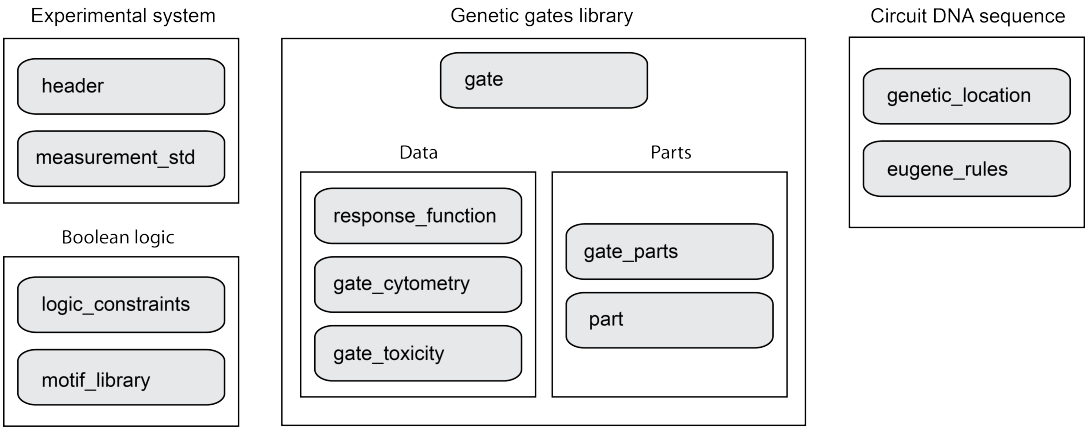

Figure S40: ***Overview of the Eco1C1G1T1 User Constraint File.*** *All data objects in the UCF are organized by collection name (shaded gray).*

VII.A. File format

The UCF is specified using JavaScript Object Notation (JSON). JSON is a widely used and language-independent format based on attribute:value pairs, which is human readable, machine parseable, and can be converted to common data structures in other languages. In a JSON attribute:value pair, the values are restricted to these types: string, number, boolean, null, array (square brackets), and object (curly brackets).

SBOL XML was also considered as a file format for the UCF. SBOL Version 1.1 (*36*) is tailored for specifying parts and composite parts in a genetic design hierarchy, where URIs (uniform resource identifiers) are used to uniquely and globally identify parts via the World Wide Web. While this format would be directly applicable for our *gate_parts* and *parts* UCF collections, the other collections required a more flexible representation. However, the proposed versions of SBOL will have more versatile data model(*37*), have the ability to specify custom objects, and will be able to read/write data in JSON format.

Required collections:

header, measurement_std, logic_constraints, gates, response_functions, gate_parts, parts

These collections specify the experimental system, the available gate types for logic synthesis, response function data for assignment, and gate parts to build the final DNA sequence.

Optional collections:

motif_library: if omitted, subgraph substitution will not occur as an optimization step in logic synthesis.

gate_cytometry: if omitted, the output predictions will be median values, as opposed to cytometry distributions.

gate_toxicity: if omitted, the prediction of growth impact will not be calculated.

eugene_rules: if omitted, unconstrained circuit layout design will occur. This will result in variations in tandem promoter order, variations in gate order, and variations in gate orientation.

genetic_locations: if omitted, the output DNA sequence will contain the circuit components only, and the user will be responsible for deciding and implementing the genetic context of the circuit design.

VII.C. Examples and details for each collection.

Each section describes a different collection with a brief description and a box containing an example object. For all collections, attribute names are parsed in Cello.

VII.C.1. "header"

The header collection specifies the operating conditions, strain, and genetic location where the gate measurements were made and the circuit predictions would be valid. These data do not impact circuit design in Cello. However, it is required to describe the operating conditions for which the circuit designs are valid. Thus, it is required by Cello to accept the file as a valid UCF.

version: This demarcates the iteration of the UCF. Version updates could include larger gate libraries, changes in experimental conditions, more accurate data, etc. Our current numbering system is shown below, but this particular format is not required.

example:

"version": "Eco1C1G1T1"

<string>	Organism identifier (Eco for *E. coli*)
Eco<number>	Strain identifier (counting up from 1; Eco1 = NEB 10-beta)
C<number>	Experimental conditions identifier
G<number>	Genetic gates and insertion location identifier
T<number>	Technology mapping and motifs identifier

required: yes

author: Intended to help document versions and modifications of UCF files.
required: no

```
{
  "collection": "header",

  "version": "Eco1C1G1T1",

  "date": "2015-04-08",

  "author": ["Bryan Der", “Alec Nielsen”, “Prashant Vaidyanathan”],

  "organism": "Escherichia coli NEB 10-beta",

  "genome": "NEB 10-beta Δ(ara-leu) 7697 araD139  fhuA ΔlacX74 galK16 galE15 e14-
ϕ80dlacZΔM15 recA1 relA1 endA1 nupG  rpsL (Str^R) rph spoT1 Δ(mrr-hsdRMS-mcrBC) (New England
Biolabs)",

  “media”: “M9 minimal media composed of M9 media salts (6.78 g/L Na2HPO4, 3 g/L KH2PO4, 1 g/L
NH4Cl, 0.5 g/L NaCl, 0.34 g/L thiamine hydrochloride, 0.4% D-glucose, 0.2% Casamino acids, 2
mM MgSO4, and 0.1 mM CaCl2; kanamycin (50 ug/ml), spectinomycin (50 ug/ml)”,

  "temperature": "37",

  "growth": "Inoculation: Individual colonies into M9 media, 16 hours overnight in plate
shaker.  Dilution: Next day, cells dilute ~200-fold into M9 media with antibiotics, growth for
3 hours.  Induction: Cells diluted ~650-fold into M9 media with antibiotics.  Growth: shaking
incubator for 5 hours.  Arrest protein production: PBS and 2mg/ml kanamycin.  Measurement:
flow cytometry, data processing for REU normalization. ",

}
```

organism: Defines the organism, species, and strain for which the circuits are compiled.
example:
“organism”: “Escherichia coli NEB 10-beta”
required: yes
allowed values: should be the full organism name

genome: Specifies the genotype of the organism.
example:
"genome": "NEB 10 Δ(ara-leu) 7697 araD139 fhuA ΔlacX74 galK16 galE15 e14-
ϕ80dlacZΔM15 recA1 relA1 endA1 nupG rpsL (StrR) rph spoT1 Δ(mrr-hsdRMS-mcrBC)"
example:
“genome”: “K-12 MG1655* [F-lambda-ilvG-rfb-50 rph-1 Δ(araCBAD) Δ(LacI)]
required: yes
allowed values: any string, including the entire genome sequence

temperature: Specifies the temperature at which the circuits are expected to perform (and gates measured).
required: yes
allowed values: units of Celsius

growth: Specifies the growth conditions at which the circuits are expected to perform (and gates measures).

required: yes

VII.C.2. "measurement_std"

This collection specifies the unit of measurement that is used for all signals (sensor ON/OFF levels, gate response functions, output levels). The standard unit in UCF Eco1C1G1T1 is RPU (relative expression unit). It also includes a description of the standard plasmid, which contains a constitutively expressed YFP cassette from a standard promoter with ribozyme insulation, the complete plasmid DNA sequence for the plasmid is specified, and instructions for normalization.

required: yes

```
{
  "collection": "measurement_std",

  "signal_carrier_units": "REU",

  "plasmid_description": "p15A plasmid backbone with kanamycin resistance and a YFP expression
cassette. Upstream isulation by terminator L3S3P21 and a 5'-promoter spacer. Promoter
BBa_J23101, ribozyme RiboJ, RBS BBa_B0064 drives constitutive YFP expression, with
transcriptional termination by L3S3P21.",

  "plasmid_sequence": [all lines of the Genbank file (not shown) for the measurement standard
plasmid],

  "normalization_instructions": "The following equation converts the median YFP fluorescence
to REU.  REU = (YFP – YFP0)/(YFPREU – YFP0), where YFP is the median fluorescence of the cells
of interest, YFP0 is the median autofluorescence, and YFPREU is the median fluorescence of the
cells containing the measurement standard plasmid",

}
```

signal_carrier_units: Different circuit design frameworks might use different units for quantifying high / low signals. If a different unit is used, the unit should be used for all signal-related fields in a UCF (*response_function, gate_cytometry, gate_toxicity*). The following data should all have the same signal carrier unit: sensor ON/OFF levels and gate response function input/output levels. For example, in Eco1C1G1T1, *response_function* collection is generically written, but the input/output response has units of RPU as specified by this attribute.

required: yes

example: RPU

plasmid_description: Instructions and experimental conditions for normalizing experimental measurements for the standardized units (RPU in Eco1C1G1T1). Instructions should include the full plasmid name, growth, measurement, and other conditions necessary to normalize the data.

required: yes

allowed values: plain text

plasmid_sequence: Plasmid DNA sequence containing the expression cassette that serves as the measurement standard for normalization of all signal levels in the UCF (RPU).

required: yes

allowed values: Genbank file format, with or without annotations before the sequence.

normalization_instructions: A brief set of instructions describing how data is normalized using the measurement standard.

required: yes

VII.C.3. "logic_constraints"

In this collection, the allowed Boolean gate types are specified, and the maximum number of instances is specified for each gate type. These Boolean constraints cannot exceed the genetic gates available in the library, but they can be more restrictive. For example, if 12 NOR gates are in the library, the user could constrain logic synthesis to use a maximum of 9 NOR gates. Furthermore, if there are 20 NOT gates in a library but only 12 unique repressors due to RBS variants, the Boolean constraint would be 12 NOT/NOR gates.

```
{
  "collection": "logic_constraints",
  "available_gates": [
     {
       "type" : "NOR",
       "max_instances" : 12
     },
     {
       "type" : "OUTPUT_OR",
       "max_instances" : null
     }
  ]
}
```

available_gates: Specifies a gate type and maximum number of instances of each gate type for a circuit topology from logic synthesis.

required: yes

Note: A 1-input NOR gate is equivalent to a NOT gate, so a value of 12 for *max_instances* indicates a maximum of 12 NOR + NOT gates in the circuit.

Note: An OUTPUT_OR gate does not require a transcription factor, so a value of null indicates that there are no restrictions on the number of OUTPUT_OR gates in the circuit.

Note: Specifying fan-in constraints, such as allowing a 3-input NOR gate motif, is not done here. This NOR motif would be specified in the *motif_library* collection.

VII.C.4. "motif_library"

In this collection, the user can define logic motifs that can be swapped for logically equivalent subcircuits in the last stage of logic synthesis. Subcircuits are specified using a "netlist" format(*4*), which is standard for specifying the connectivity of a list of gates. Specifying subcircuits in this way is similar to specifying structural Verilog, and the input names and output names must be defined before listing the gate connectivity.

```
{
  "collection": "motif_library",
  “inputs”:[“a”,”b”,”c”],
  “outputs”[“y”]
  "netlist": [
    "NOT(Wire0, b)",
    "NOR(Wire1, Wire0, c)",
    "NOR(Wire2, Wire1, a)",
    "NOR(Wire3, Wire2, a)",
    "NOR(Wire4, Wire2, Wire1)",
    "NOR(y, Wire3, Wire4)"
  ]
}
```

inputs: the names of input wires

example:

“inputs” : [“a”,”b”,”c”]

example:

“inputs” : [“in1”,”in2”,”in3”]

required: yes

allowed values: input names must match the input wire names in the netlist.

outputs: the names of output wires. Allows single- or multiple-output subcircuits to be defined.

example:

“outputs” : [“y”]

example:

“outputs” : [“out1”,”out2”]

allowed values: output names must match the output wire names in the netlist.

netlist: A gate type is listed, followed by the output wire name, followed by the list of input names. The examples show a multi-level (layered) NOR/NOT motif, an OUTPUT_OR motif, a 3-input NOR motif, and a primitive AND motif:

example of a precomputed NOR/NOT motif:

```
“netlist” : [
        "NOT(Wire0, b)",
        "NOR(Wire1, Wire0, c)",
        "NOR(Wire2, Wire1, a)",
        "NOR(Wire3, Wire2, a)",
        "NOR(Wire4, Wire2, Wire1)",
        "NOR(y, Wire3, Wire4)
    ]
```

example of an OUTPUT_OR gate motif:

```
“netlist” : [
        "OUTPUT_OR(y, a, b)"
    ]
```

example of a 3-input NOR gate motif:

```
“netlist” : [
        "NOR(y, a, b, c)"
    ]
```

example of an AND gate motif:

"netlist" : [
"AND(y, a, b)"
]

required: yes

allowed values: gate names are restricted to NOT, NOR, AND, OR, OUTPUT_OR, NAND, XOR, XNOR.

VII.C.5. "gates"

Gates in the library are specified in this collection. This is a concise collection that does not include other gate-related data collections (*gate_parts, response_functions, gate_cytometry, gate_toxicity*). For modularity, these data are stored in other collections, which are linked to the gate through the *gate_name* attribute. The attribute *system* allows other genetic logic systems to be specified in future versions, including zinc fingers, TALEs, CRISPRi, activator-chaperone pairs, etc.

```
{
  "collection": "gates",
  "group_name" : "BM3R1",
  "gate_name": "B3_BM3R1",
  "gate_type": "NOR",
  "system": "TetR",
}
```

group_name: this attribute is used to group variants of gates that cannot be used together in a circuit design. For example, all RBS variants of a certain repressor (B1_BM3R1, B2_BM3R1, B3_BM3R1) would belong to the same group (BM3R1), since a repressor is can only be used once per circuit assignment. Furthermore, if known cross-talking interactions between different repressors are known, these could also be put into the same group. Similarly, if homologous recombination is a concern and two gates have the same large part, then they can be placed in the same group.

required: yes

allowed values: Using the repressor name would be typical for RBS variants, but any name can be used.

gate_name: this attribute is used to link gate data in other collections to the gate object (*response_function, gate_parts*, etc).

required: yes

allowed values: Only alphanumeric and underscore characters are allowed, which complies with allowed names in the Eugene language. The gate name must be identical to the string used in the *gate_name* attribute in other collections

gate_type: used during the assignment algorithm; for example, a genetic NOR gate cannot be assigned to an AND gate in the circuit topology.

required: yes

allowed values: NOR, NOT, OR, AND, NAND, XOR, XNOR. Note: To allow for multiple inputs, NOT gates must be specified here as a NOR gate. If a repressing gate can only have a single input, the gate type can be NOT.

system: used to specify the type of biochemistry from which the gates are built. A single UCF could have gates based on different biochemistries.

required: yes
allowed values: any string, such as “TetR”, “CRISPRi”, or “Activator-chaperone”

VII.C.6. “gate_parts”

This collection specifies the transcription units and output promoters for a genetic gate, which is mapped to a gate through the *gate_name* attribute. A NOR or NOT gate may have a different number of parts compared to, say, an AND gate. Thus, the *transcription_units* attribute is an array instead of a single object for flexibility for different genetic gate types. As with Boolean primitive gates, all genetic primitive gates are restricted to have a single output promoter. The restriction of a single output promoter name is not to be confused with fan-out, where multiple instances of the named promoter are used in the circuit.

```
{
  "collection": "gate_parts",
  "gate_name": "A2_AmtR",
  "transcription_units": [
     [
      "BydvJ", "A2", "AmtR", "L3S2P55"
     ]
  ],
  "promoter": "pAmtR"
}
```

gate_name: The name of the gate.
required: yes
allowed values: the name must be identical to the *gate_name* value in the gate collection.

transcription_units: The part composition of the gate. The regulable promoter is listed separately from the transcription unit, because the promoter driving the transcription unit depends on the circuit diagram.
example:
Some gate types might require two transcription units, such as an activator-chaperone AND gate(*38*). For this reason, the value of this attribute is an array of arrays. The first element of the array is the transcription unit for InvF, and the second element of the array is the transcription unit for SicA:

```
"collection": "gate_parts",
 "gate_name": "SicA-InvF",
 "transcription_units": [  //note that this begins the outer array
   [  //element 1 of the inner array
    "RiboJ11", "RBS-InvF0", "InvF", “M13"
   ],
   [  //element 2 of the inner array
    "RiboJ10", "RBS-SicA0", "SicA", "BBa_B1006U10"
   ]
 ],
 "promoter": "pSicA"
```

required: yes
allowed value: array of arrays. The outer array contains the list of transcription units, excluding promoters, and the inner array contains the list of part names that make up each transcription unit. The part names can be any string, but the string must match a part name in the *part* collection.

promoter: The output promoter of a gate.
required: yes
allowed value: single promoter name (alphanumeric and underscore characters only).

VII.C.7. "parts"

This collection specifies basic genetic parts: promoters, ribozymes, ribosome binding sites, coding sequences, terminators, scars, spacers, etc. This collection specifies the part name, part type, and part DNA sequence. The part names listed in the *gate_parts* collection must match a part name from this collection. For example, all parts used in the A2_AmtR gate from the previous section are specified:

```
{
  "collection": "parts",
  "type": "ribozyme",
  "name": "BydvJ",
  "dnasequence":
"CTGAAGGGTGTCTCAAGGTGCGTACCTTGACTGATGAGTCCGAAAGGACGAAACACCCCTCTACAAATAATTTTGTTTAA"
}
```

```
{
  "collection": "parts",
  "type": "rbs",
  "name": "A2",
  "dnasequence": "AATGTTCCCTAATAATCAGCAAAGAGGTTACTAG"
}
```

```
{
  "collection": "parts",
  "type": "cds",
  "name": "AmtR",
  "dnasequence":
"ATGGCAGGCGCAGTTGGTCGTCCGCGTCGTAGTGCACCGCGTCGTGCAGGTAAAAATCCGCGTGAAGAAATTCTGGATGCAAGCGCAGAACTG
TTTACCCGTCAGGGTTTTGCAACCACCAGTACCCATCAGATTGCAGATGCAGTTGGTATTCGTCAGGCAAGCCTGTATTATCATTTTCCGAGCA
AAACCGAAATCTTTCTGACCCTGCTGAAAAGCACCGTTGAACCGAGCACCGTTCTGGCAGAAGATCTGAGCACCCTGGATGCAGGTCCGGAAAT
GCGTCTGTGGGCAATTGTTGCAAGCGAAGTTCGTCTGCTGCTGAGCACCAAATGGAATGTTGGTCGTCTGTATCAGCTGCCGATTGTTGGTAGC
GAAGAATTTGCAGAATATCATAGCCAGCGTGAAGCACTGACCAATGTTTTTCGTGATCTGGCAACCGAAATTGTTGGTGATGATCCGCGTGCAG
AACTGCCGTTTCATATTACCATGAGCGTTATTGAAATGCGTCGCAATGATGGTAAAATTCCGAGTCCGCTGAGCGCAGATAGCCTGCCGGAAAC
CGCAATTATGCTGGCAGATGCAAGCCTGGCAGTTCTGGGTGCACCGCTGCCTGCAGATCGTGTTGAAAAAACCCTGGAACTGATTAAACAGGCA
GATGCAAAATAATAA"
}
```

```
{
  "collection": "parts",
  "type": "terminator",
  "name": "L3S2P55",
  "dnasequence": "CTCGGTACCAAAGACGAACAATAAGACGCTGAAAAGCGTCTTTTTTCGTTTTGGTCC"
}
```

type: The part class.
required: yes
allowed values: any string, but the string might be used during enumeration of part-based Eugene rules, and will be used for annotation of the output DNA sequence.

name: The name of the part.
required: yes
allowed values: the part name must match the part name specified in the *gate_parts* collection.

dnasequence: The sequence of the part.
required: yes
allowed values: only ATCGatcg characters and the DNA sequence is in the forward orientation.

VII.C.8. "response_functions"

This collection specifies the response function for a gate identified by the gate_name attribute. The response function includes a mathematical equation as well as a set of parameters that map to the variables in that equation. Different gate types may be captured by different mathematical forms and this can be specified in this collection. For NOT/NOR gates, a Hill equation (Equation S1) describes the cooperative and monotonic decrease in output with respect to input, and the parameters y_{max}, y_{min}, K, and n are fitted from experimental data for each gate. Note that y_{max}, y_{min}, and K are specified generically without units; units are described in the header collection, and these units should be used consistently throughout all data related to signal levels.

Cello uses a math evaluator that solves equations expressed as strings. User-defined equations with user-defined parameters can be accommodated, as long as parameter names match the variable names in the equation string. This is an example of a Hill equation for the A2_AmtR gate:

```
{
  "collection": "response_functions",
  "gate_name": "A2_AmtR",
  "variables": ["x"]
  "parameters": [
    {
      "name": "ymax",
      "value": 13.18696
    },
    {
      "name": "ymin",
      "value": 0.316394
    },
    {
      "name": "K",
      "value": 0.169953
    },
    {
      "name": "n",
      "value": 1.319126
    },
  ],
  "equation": "ymin+(ymax-ymin)/(1.0+(x/K)^n)"
}
```

gate_name: The name of the gate.

required: yes

allowed values: name must match the intended gate name from the *gate* collection.

variables: The input to each gate; for example, the activity of the input promoter.

example: if there are multiple inputs (x and y) to the response function:

"variables": ["x", "y"],

required: yes

allowed values: array of strings, where each string must match a variable name in the equation.

parameters: Definition and numerical value of each parameter in the response function.

required: yes

allowed values: array of objects, where each object is a parameter with a name attribute and value attribute. The name must match the equation string.

equation: The mathematical form of the response function.

example: a two input response function, this equation can be used for an AND gate:

"equation": -log(D + ((A-D)/(1+((x/C)^B))) + D + ((A-D)/(1+((y/C)^B)))),

required: yes

allowed values: right-hand side of an equation of interest; the calculated left-hand side value is returned by the evaluate function. It can take any form and is not restricted to a Hill equation. The math evaluator class supports common operators, constants, and functions such as:

```
^       power
*       multiply
(       implicit multiply
/       divide
+       add
-       subtract
LN2     natural log
log10   log base 10
min     minimum of
max     maximum of
sqrt    square root
```

VII.C.9. "gate_cytometry"

This collection specifies histograms that describe the response function of each gate. Note that cytometry data is not required for Cello to run, but including it allows Cello to predict distributions for the simulated circuit output. In its absence, the output of Cello will be predicted values, as opposed to predicted cytometry distributions.

When the response function is characterized, a set of distributions is measured for different activities of the input promoter. Thus, cytometry_data for a gate must be in the form of an array, where each element of the array represents a promoter activity. The data could represent one representative experiment or could be obtained by averaging the distributions from experimental replicates. In the cytometry_data data structure, an input signal level (input) is paired with a histogram representing the measured output level (output_bins, output_counts). Importantly, cytometry data must be consistently binned for all gates; the number of bins (NBINS), minimal value (MIN), and maximum value (MAX) must be used when generating the histogram for a cytometry sample. This consistency is required to propagate distributions through each layer in the circuit. Typical values would be:

```
NBINS = 250.
MAX   = 100.   (RPUs, linear space, not log space)
```

`MIN = 0.001. (RPUs, linear space, not log space)`

To generate the cytometry data for Cello, fluorescence values from flow cytometry must first be converted from arbitrary units to RPUs. This process is described in Supplementary Section VI. Thus, for a single response function, each titration point with a defined input level has a corresponding histogram. These discrete titration points are used to generate a continuous distribution response function by overlaying histograms on the median determined by the Hill equation. As a result, any input RPU value can produce a predicted output histogram (Supplementary Section V.F). In Cello, a single histogram can still be used to generate histograms for the entire distribution response function (if the parameters for the average response function are provided), though histograms from 8 or more titration points are the expected use case.

input: This attribute specifies the input promoter activity of the distribution.

required: yes

allowed values: a single RPU value for the current titration.

output_bins: For the given input level of the current titration, the bins of output levels are listed.

required: yes

allowed values: an array of values (RPU) specifying the histogram bins. The array length (number of bins) must be the same for all histograms specified in this collection. When generating the histogram, binning must be done in log space (log10 RPU), but the bin values must be specified in linear space (RPU). For consistency in the UCF, all RPU values are specified in linear space.

output_counts: counts for each output bin.

required: yes

allowed values: an array of counts for the histogram. Counts can be integers, or fractional counts: Cello will normalize each histogram so the sum of all counts equals 10,000. The array length must be the same as the **output_bins** array, and must be the same for all histograms specified in this collection. Scientific notation (E) is allowed for very low fractional count values.

Thus, the JSON object for each cytometry titration point consists of the attributes *input*, *output_bins*, and *output_counts*. Each titration point is listed in an array representing the full response function titration, and this array of objects is the value of an attribute called *cytometry_data*:

cytometry_data: list of titrations for characterizing a response function using flow cytometry

required: yes

allowed values: a list of JSON objects for each titration point, described above (*input*, *output_bins*, and *output_counts*). Note that the number of titration points can range from 1 to N, where N is any number of titration points used in response function characterization. Unlike the histogram binning, which must be consistent across all gates, the number of titration points can differ across gates.

An example of a JSON object in the gate_cytometry collection is shown below. For readability, only two titrations with a small number of values are shown. Notice that the output_bins are consistent, but the output_counts vary between titrations. See the provided UCF for an example of this organization.

```
{
  "collection": "gate_cytometry",
  "gate_name": "B3_BM3R1",
  "cytometry_data": [
    {
      “input”: 0.018,
      “output_bins":   [ ... 0.129, 0.136, 0.144, 0.152, 0.161, ...],
      “output_counts": [ ... 0.005, 0.002, 0.003, 0.001, 0.0005,...]
    },
    {
      “input”: 0.028,
      “output_bins":   [ ... 0.129, 0.136, 0.144, 0.152, 0.161, ...],
      “output_counts": [ ... 0.002, 0.003, 0.004, 0.004, 0.003, ...],
    }
  (only two titration points shown)
}
```

VII.C.10. “gate_toxicity”

This collection specifies the growth curve for each gate. The object has two data attributes: “input” is the input level (promoter activity in standard units) driving expression of the gate, and “growth” contains growth values normalized by a control cell population. A value of 1.0 indicates full growth, and a value of 0.0 indicates no growth. Growth measurements could take the form of OD_{600} endpoint measurements, cytometry events, growth rates, colony counts, etc. In this manuscript, growth values are OD_{600} endpoints when P_{Tac} is driving the NOT gate, divided by the OD_{600} of cells with P_{Tac} driving YFP. Because titrations are discrete and input levels during circuit simulation are continuous, an input promoter activity requires interpolation of the data to generate a growth value for that specific input level (Supplementary Section V.D). Interpolation is used to compute a growth value that is a weighted average of the two nearest *growth* values, where the weight is determined by proximity of the input level to the two nearest *input* values. If an input level is less than the lowest input in the growth curve, the first growth value is used. If an input level is greater than the highest input in the growth curve, the last growth value is used.

```
{
  "collection": "gate_toxicity",
  "gate_name": "A2_AmtR",
  "input": [
    0.004,
    0.007,
    0.012,
    0.034,
    0.062,
    0.099,
    0.144,
    0.247,
    0.418,
    0.739,
    1.012,
    2.078
],
  "growth": [
    1.03,
    0.99,
    0.99,
    1.04,
    1.08,
    1.05,
    1.08,
    1.1,
    1.05,
    1.03,
    0.81,
    0.71
  ]
}
```


gate_name: The name of the gate.

required: yes

allowed values: gate name must correspond to the correct name in the *gate* collection.

input: promoter activity driving expression of the regulator.

required: yes

allowed values: input RPU values used in the growth curve titration. Values must be ordered from low to high.

growth: normalized cell growth measurement.

required: yes

allowed values: growth values normalized by a control cell population. The length of the array must be equal to the length of the *input* array. After Cello reads the data, values < 0.0 will be set to 0.0, values > 1.0 will be set to 1.0.

VII.C.11. "eugene_rules"

This collection specifies constraints on the physical layout of the circuit, written using Eugene(*25*). These rules are used in tandem with combinatorial design algorithms to build the DNA sequence of the circuit that is the output of Cello. In addition, these rules will be enforced if more than one construct is designed. Our use of the Eugene rules is organized into two attributes in this UCF collection:

eugene_part_rules and eugene_gate_rules. Due to the hierarchical Eugene specification, rules are applied to the parts within a gate device separately from the rules applied to the gates within a circuit device (Section V.E). Note that "device" is a Eugene term for a collection of parts, or a collection of other devices. Rule is also a Eugene term, where a rule is applied to a device to constrain its design.

Part rules. The part rules in the Eco1C1G1T1 UCF enforce the following. Roadblocking requires that a promoter (*e.g.*, P_{SrpR}) be in the upstream position of a NOR gate. In other words, when P_{SrpR} is in the forward orientation, P_{SrpR} must be the first part in the gate device (and this is inverted if the gate is in the reverse orientation). Roadblocking is not the only reason to enforce particular promoter orders and this is a generalized approach to constraining a particular part order.

example:

STARTSWITH pSrpR

In Eco1C1G1T1, we also use rules to constrain a preferred promoter order (Supplementary Section II.B, Figure S8).

example:

pAmtR BEFORE pBetI

For a given gate device in the Eugene file (Section V.E), all part rules are parsed from the UCF, but a part rule will not be applied to the gate device if the gate does not contain a part with that name.

Gate rules. Gate rules can be used to specify the order in which regulators appear in the circuit construct. Because the UCF must accommodate any possible circuit assignment, a combinatorial list of rules is needed to constrain the gate order for any assignment. Given a circuit assignment, Cello scans all of the *eugene_gate_rules*, and if any gate name in a rule is absent from the current assignment, that rule is omitted. For example, the rule below would be omitted if PhlF is not assigned to the current circuit.

example:

gate_PhlF BEFORE gate_BM3R1

Note that some gate rules do not list a gate name, such as ALL_FORWARD. For the ALL_FORWARD rule, no gate names are parsed, so it is not possible for the ALL_FORWARD rule to be omitted based on the gates assigned to the circuit.

examples:

ALL_FORWARD

(all gates will be in the 5' to 3' forward orientation)

ALTERNATE_ORIENTATION

(example: gate 1 forward, gate 2 reverse, gate 3 forward, gate 4 reverse)

(example: gate 1 reverse, gate 2 forward, gate 3 reverse, gate 4 forward)

SOME_REVERSE

(one or more gates will be in the 3' to 5' reverse orientation)

eugene_part_rules: rules that constrain parts within a gate device

required: no

allowed values: Any Eugene rule can be used(*25*). Part names in the Eugene rules must be identical to the part names specified in the *parts* collection of the UCF.

eugene_gate_rules: rules that constrain gate order/orientation within a circuit device

required: no

allowed values: Any Eugene rule can be used(*25*). Gate names must follow a naming convention for correct automatic generation of the Eugene file. This convention uses the "gate_" prefix followed by the repressor name, for example gate_PhlF.

VII.C.12. "genetic_location"

This collection defines the physical location of the sensor module, circuit module, and output module in the context of the plasmid and/or genomic landing pads. The sensor module encodes the transcription factors that are required by the sensors. This could be a transcription factor and its necessary transcription/translation parts (*e.g*., promoter, RBS, AraC, terminator). If all of the machinery is endogenous to the host organism, then there may not be a sensor module. The circuit module encompasses the circuit designed by Cello. The output module contains the circuit regulable promoter(s) assigned to the output gates in the circuit, followed by the actuator gene(s) of interest. Note that the locations for each of the modules may differ from the context in which the gates were characterized. Attributes are provided for correction factors to be included (*e.g*., to correct for copy number or the fluorescent protein used).

locations: This attribute lists genetic locations that will be referred to by name in the sensor, circuit, and output location attributes.

name: the name of the plasmid or genomic landing pad.
required: yes
allowed values: Any string, but the name needs to be internally consistent when referred to in the *sensor_module_location, circuit_module_location,* and *output_module_location* objects.

file: The NCBI sequence file for the plasmid.
required: yes
allowed values: all lines of the GenBank file. The GenBank file can have annotations prior to the first base pair of the DNA sequence, but the sequence should have the following format.

```
ORIGIN

        1 gcttcctcgc tcactgactc gctgcacgag gcagacctca gcgctagcgg agtgtatact
       61 ggcttactat gttggcactg atgagggtgt cagtgaagtg cttcatgtgg caggagaaaa
      121 aaggctgcac cggtgcgtca gcagaatatg tgatacagga tatattccgc ttcctcgctc
      181 actgactcgc tacgctcggt cgttcgactg cggcgagcgg aaatggctta cgaacggggc
      241 ggagatttcc tggaagatgc caggaagata cttaacaggg aagtgagagg gccgcggcaa
      301 agccgttttt ccataggctc cgccccctg acaagcatca cgaaatctga cgctcaaatc
      361 agtggtggcg aaacccgaca ggactataaa gataccaggc gtttcccctg gcggctccct
      421 cgtgcgctct cctgttcctg cctttcggtt taccggtgtc attccgctgt tatggccgcg
      481 tttgtctcat tccacgcctg acactcagtt ccgggtaggc agttcgctcc aagctggact
```

The JSON collection for *genetic_locations* is given below, but DNA sequence files are not shown. Note that each of the locations is structured as an array of objects, where each object is a location. The example only includes one location per array, but there might be designs that require multiple sensor modules. It is also possible to insert sensor modules in series, rather than concatenating individual sensor modules ahead of time. Thus, the UCF is written to accommodate a list of locations for each type of module (sensor, circuit, output).

```
{
  “collection”: “genetic_locations”,

  “locations” : [
    {
      “name” : “pAN1717”,
      “file” : FULL GENBANK FILE NOT SHOWN, SEE BELOW
    },
    {
      “name” : “pAN1201”,
      “file” : FULL GENBANK FILE NOT SHOWN, SEE BELOW
    },
    {
      “name” : “pAN4020”,
      “file” : FULL GENBANK FILE NOT SHOWN, SEE BELOW
    }
  ]

  “sensor_module_location” :
     [
       {“location_name”: “pAN1201”,
        “bp_range” : [58, 556]
       }
     ]

  “circuit_module_location” :
     [
       {“location_name”: “pAN1201”,
        “bp_range” : [58, 556]
       }
     ]

  “output_module_location” :
     [
       {“location_name”: “pAN4020”,
        “bp_range” : [953, 953],
        “unit_conversion” : 0.40
       }
     ]
}
```

sensor_module_location: genetic location where the sensor module (if any) will be inserted. The sensor module encodes transcription factors that regulate the circuit input promoters, and contain all of the necessary parts for expression (constitutive promoter, RBS, CDS, terminator).

required: yes

circuit_module_location: genetic location where the circuit module will be inserted. The circuit module is designed by Cello, and it contains the user-defined input promoters, and parts from the Cello gates library.

required: yes

output_module_location: genetic location where the output module will be inserted. Expression of the output module/modules is/are driven by the promoters assigned by Cello.

required: no

location_name: the name of the GenBank file listed in the *locations* attribute where the module(s) will be incorporated.

bp_range: the starting and ending base pair numbers in the GenBank file where the module will be inserted.

allowed values: to insert without removing any bases, the start, end base pair numbers will be the same. If region of DNA is removed during cloning, for example, a region between two restriction sites, the start, and base pair numbers should span that range.

unit_conversion: If the output plasmid differs from the plasmid used to characterize the circuit, a conversion factor may be necessary. One example would be if it has a different copy number. In the case of the Eco1C1G1T1 UCF, we measured a conversion factor of 0.40 to convert promoter activities on p15A (RPU) to promoter activities on pSC101 (RPU) (Methods). All output levels (RPU) are multiplied by this conversion factor.

The sensor, circuit, and/or output modules could be inserted into the genome, rather than plasmids. Here we provide an example of a possible specification for choosing a genomic site for the circuit output module. This example specifies genomic landing pad that is highly expressed and is amenable to large sequence insertions in any orientation.

```
"output_module_genomic_locations" : [

    "organism": "Escherichia coli str. K-12 substr. DH10B",

    "taxid": 316385,

    "location_name": "atpl-gidB intergenic region",

    "bp_range": [4018174, 4018497],

    "flanking_upstream_sequence": "ATACGGTGCGCCCCCGTGATTTCAAACAATAA",

    "flanking_downstream_sequence": "TTGTGATATTTTCACTAATGACTTATTTTCTGCT"
]
```

VIII. Materials and Methods

VIII.A. ***Circuit induction and measurement guide.*** A step-by-step guide to transforming, inducing, and measuring circuits is provided below. The 0xF6 circuit is used as a specific example.

1. *Co-transform the 0xF6 plasmid (pAN3938) and the P_{PhlF}-P_{AmtR}-YFP output plasmid (pAN4044) into chemically competent NEB 10-beta (New England Biolabs, MA, C3019).* Add 1 µl of each purified plasmid to 50 µl of thawed chemically competent cells. Incubate mixture on ice for half an hour, and then heat shock at 42°C for 30 seconds. Incubate on ice for 2 more minutes, and then add 1 mL room temperature SOC media. Incubate at 37°C for one hour. Plate serial dilutions of recovered cells on LB agar plates with 50 µg/mL kanamycin (Gold Biotechnology, MO, K-120-5) and 50 µg/mL spectinomycin (Gold Biotechnology, MO, S-140-5). Grow plates at 37°C overnight.
2. The day after transformation, pick single colonies and inoculate into 200 µl of M9 glucose with antibiotics in a V-bottom 96-well plate (Nunc, Roskilde, Denmark, 249952). M9 glucose media is composed of M9 media salts (6.78 g/L Na_2HPO_4, 3 g/L KH_2PO_4, 1 g/L NH_4Cl, 0.5 g/L NaCl; Sigma-Aldrich, MO, M6030), 0.34 g/L thiamine hydrochloride (Sigma-Aldrich, MO, T4625), 0.4% D-glucose (Sigma-Aldrich, MO, G8270), 0.2% Casamino acids (Acros, NJ, AC61204-5000), 2 mM $MgSO_4$ (Sigma-Aldrich, MO, 230391), and 0.1 mM $CaCl_2$ (Sigma-Aldrich, MO, 449709). Antibiotic concentrations in M9 glucose media are 50 µg/mL kanamycin and 50 µg/mL spectinomycin.
3. Grow single colonies in V-bottom 96-well plates overnight (16 hours) at 37°C and 1000 RPM in an ELMI Digital Thermos Microplates shaker incubator (Elmi Ltd, Riga, Latvia).
4. The next day, dilute the overnight cultures 178-fold by adding 15 µL of culture into 185 µL of M9 glucose media, and then 15 µL of that dilution into 185 µL of M9 glucose media with 50 µg/mL kanamycin and 50 µg/mL spectinomycin in a V-bottom 96-well plate.
5. Grow the diluted cultures in an ELMI shaker incubator at 37 °C and 1000 RPM for three hours.
6. Dilute the cultures by adding 15 µL of culture into 185 µL of M9 glucose media.
7. Take 3 µL aliquots of that dilution and distribute into eight wells with 145 µL of inducer-containing M9 glucose media with 50 µg/mL kanamycin and 50 µg/mL spectinomycin in a V-bottom 96-well plate. The eight wells correspond to the inducer conditions (—/—/—), (—/—/+), (—/+/—), (—/+/+), (+/—/—), (+/—/+), (+/+/—), and (+/+/+). Each — or + corresponds to the absence or presence of 5 mM L-arabinose (Sigma-Aldrich, MO, A3256), 2 ng/mL aTc (anhydrotetracycline hydrochloride; Sigma-Aldrich, MO, 37919), and 1 mM IPTG (isopropyl β-D-1-thiogalactopyranoside; Sigma-Aldrich, MO, I6758).
8. Grow the cultures containing inducer in an ELMI shaker incubator at 37°C and 1000 RPM for five hours. Note: at the end of five hours, the cultures should still be in exponential-growth phase, and not in stationary phase.
9. Aliquot 10 µL of cell culture into 190 µL of phosphate buffered saline (PBS) containing 2 mg/mL kanamycin to arrest protein production and cell growth. Incubate this mixture for one hour at room temperature.
10. Measure the fluorescence of >1000 cells per inducer condition using flow cytometry (see Flow cytometry analysis).

VIII.B. ***Circuits library measurement and time-courses.*** For 2-input circuits, the protocol was as above except that the four inducer combinations were the presence or absence of 1 mM IPTG and 2 ng/mL aTc. For all 3-input circuits, the protocol was identical to above. For time-course measurements, we took an initial sample for cytometry before dilution into inducer-containing medium. Next, we performed 10 sets of parallel circuit inductions (all eight states) for a given circuit, and removed 50 µL from a consecutive set every 30 minutes for cytometry analysis.

VIII.C. ***Circuit analysis.*** After fluorescence measurement by flow cytometry, fluorescence was measured using flow cytometry (see Flow cytometry analysis), the medians of the YFP histograms were calculated and converted to RPU (Supplementary Section VI). Individual states were deemed "successful" if the experimental distributions were near the predicted distributions, as measured by eye. Because the output plasmid is lower copy than the gate measurement plasmid, the amount of YFP produced is lower. We measured P_{Tac} induction of the YFP RPU cassette on the circuit plasmid and output plasmid, and observed a 2.5-fold decrease in the amount of YFP produced from the output plasmid. We used the conversion factor to downscale all predicted output values; it is stored in the "genetic_locations" collection of the Eco1C1G1T1 UCF, as the "unit_conversion" attribute in "output_module_location". Note:

VIII.D. ***Strain, media, and inducers.*** *E. coli* NEB 10-beta Δ*(ara-leu) 7697 araD139 fhuA* Δ*lacX74 galK16 galE15 e14-*φ*80*d*lacZ*Δ*M15 recA1 relA1 endA1 nupG rpsL* (Str^R) *rph spoT1* Δ*(mrr-hsdRMS-mcrBC), a DH10B derviative*(*35*), was used for cloning and measurements (New England Biolabs, MA, C3019). Cells were grown in LB Miller broth (Difco, MI, 90003-350) for harvesting plasmid. Cells were grown M9 glucose media for measurements. M9 glucose media was composed of M9 media salts (6.78 g/L Na_2HPO_4, 3 g/L KH_2PO_4, 1 g/L NH_4Cl, 0.5 g/L NaCl; Sigma-Aldrich, MO, M6030), 0.34 g/L thiamine hydrochloride (Sigma-Aldrich, MO, T4625), 0.4% D-glucose (Sigma-Aldrich, MO, G8270), 0.2% Casamino acids (Acros, NJ, AC61204-5000), 2 mM $MgSO_4$ (Sigma-Aldrich, MO, 230391), and 0.1 mM $CaCl_2$ (Sigma-Aldrich, MO, 449709). Chemical inducers used as inputs for sensor promoters were isopropyl β-D-1-thiogalactopyranoside (IPTG; Sigma-Aldrich, MO, I6758), anhydrotetracycline hydrochloride (aTc; Sigma-Aldrich, MO, 37919), and L-arabinose (Sigma-Aldrich, MO, A3256). Antibiotics used to select for the presence of plasmids were 100 μg/ml ampicillin (Gold Biotechnology, MO, A-301-5), 50 μg/ml kanamycin (Gold Biotechnology, MO, K-120-5), and 50 μg/ml spectinomycin (Gold Biotechnology, MO, S-140-5).

VIII.E. ***Design and assembly of 2-input circuits*****.** The circuits in Figure 3 that based on non-insulated gates were constructed by using the DNA sequences previously described(*13*) and patterning the promoters in front of the repressors consistent with the desired circuit diagram (Figure S5). The insulated circuits in Figure 3 were constructed automatically, but using software developed in MATLAB that was the precursor to Cello. In this program, all possible gate assignments were exhaustively checked and their performance scored as min(ON)/max(OFF). Promoter activities (in RPU) were propagated through a circuit using the response functions of the insulated gates. The activity of tandem promoters was taken as the sum of the activities of the individual promoters. A first version of roadblocking rules was included to disallow certain promoters in downstream positions. The insulated response functions of the gates in Figure 3b are provided in Supplementary Section I.D.

VIII.F. ***Ribozyme cleavage assay.*** For *in vitro* quantification of cleavage, we performed the Rapid Amplification of cDNA End (RACE) assay(*39*). For each sample, one colony was inoculated into 1 mL LB Miller broth with 20 μg/mL chloramphenicol and then grown for 16 hours at 37 °C shaking at 250 rpm. The next day, the liquid culture was diluted 1000-fold into M9 glucose media (1 μL into 1 mL) with chloramphenicol and 1 mM of IPTG, and then grown until an OD_{600} of 0.2. Cells were then harvested and total mRNA was extracted using the RiboPure bacteria kit (Ambion, CA, AM1924). To ligate a unique RNA adaptor to the 5'-end of the mRNA, three enzymatic steps were performed sequentially. First, 15 μg of purified mRNA was treated with 10 U of T4 polynucleotide kinase (NEB, MA, M0201S) in a total volume of 50 μl 1X T4 DNA ligase buffer and incubated for one hour at 37 °C to phosphorylate the end of the cleaved mRNA. Second, the mRNA was purified by phenol/chloroform extraction (USB, CA, 75831) and ethanol precipitation (VWR, PA, V1016) and then treated with 10 U of Tobacco acid pyrophosphatase

(TAP, Epicenter, T19250) in 50 µl of 1X TAP buffer for two hours at 37 °C to convert the triphosphate of uncleaved mRNA to monophosphate. The treated mRNA was phenol/chloroform extracted and ethanol precipitated once again. Next, 1 µL of 100 µM RNA adaptor (5'-GAGGACUCGAGCUCAAGC-3') was ligated to all extracted mRNA using 15 U of T4 RNA ligase (Ambion, CA, AM2140) in 30 µl of 1X RNA ligase buffer for 2 hours at 37 °C. The mRNA was phenol/chloroform extracted and ethanol precipitated one final time. Next, we reverse transcribed the mRNA using 200 U of SuperScript III (Invitrogen, CA, 18080-044) with a gene specific primer (GSP1, 5'-ATCCCCATCTTGTCTGCGACAG-3') in 20 µl of 1X SuperScript III buffer. In each previous enzymatic step, 20 U of RNasin (Promega, WI, N2611) or 40 U of RNaseOUT (Invitrogen, CA, 100000840) was added to inhibit RNase activity. After reverse transcription, 2 U of RNase H (Invitrogen, CA, 18021-014) was added directly to the 20 µl volume to remove RNA from any RNA/DNA duplex. The cDNA was used as a template for PCR amplification using two primers (the first being the DNA version of the RNA adapter: 5'-GAGGACTCGAGCTCAAGC-3', and the second being the gene-specific primer named GSP2: 5'-TCCTGGGATAAGCCAAGTTC-3'). We performed the PCR using 5 pmol each primer, 2 µl of cDNA template, and Phusion Hi-Fi DNA polymerase (NEB, MA, M0530L) with a 58 °C annealing temperature and 10 second elongation time for 29 cycles. The resulting PCR product comprises multiple sized bands that correspond to cleaved and uncleaved mRNA fragments. These were separated by gel electrophoresis on a 15% acrylamide gel (Bio-Rad, CA, 456-5053) for 1 hour at 100 V. The band corresponding to the cleaved mRNA was excised and placed into 50 µl of water, allowing the DNA to diffuse into the water over 24 hours. This aqueous DNA solution was used as template for a second PCR (performed identically to above). This PCR product was submitted for Sanger sequencing using primer GSP2. To quantitate ribozyme cleavage efficiency, the 15% acrylamide gel with PCR products separated by gel electrophoresis was imaged using a ChemiDoc MP (Biorad, CA, 170-8280). The band intensity of each fragment was integrated using ImageJ 1.47v (National Institute of Health, MD, http://imagej.nih.gov/ij/). The "rectangular selection tool" was used to select the region surrounding both the cleaved and uncleaved bands between 150 bp and 250 bp. Using the "gel analysis tools", band intensity was plotted, and background was subtracted to obtain a single value corresponding to the intensity of the cleaved and uncleaved band. The intensity of cleaved band was divided by the total sum intensity of both bands to obtain the fraction of cleaved mRNA.

VIII.G. ***In vivo ribozyme insulation assay.*** The four ribozyme-insulator constructs (Figure S3) were transformed into separate aliquots of *E. coli* NEB 10-beta (New England Biolabs, MA, C3019). One colony from each transformant was inoculated into 1 mL of LB with 20 µg/mL chloramphenicol in a culture tube and grown for 16 hours at 37 °C shaking at 250 rpm. The next day, the culture was diluted 178-fold (two serial dilutions of 15 µL into 185 µL) into M9 glucose media with chloramphenicol in a V-bottom 96-well plate (Nunc, Roskilde, Denmark, 249952) and grown for three hours at 37 °C shaking at 1000 rpm in an ELMI Digital Thermos Microplates shaker incubator (Elmi Ltd, Riga, Latvia). Next, the cells were diluted 658-fold (two serial dilutions of 15 µL into 185 µL, then 3 µL into 145 µL) into M9 glucose media with chloramphenicol and IPTG. IPTG concentrations used for the P_{Tac} constructs were: 0, 0.12, 0.48, 1.9, 7.6, 30.4, and 121.6 µM; IPTG concentrations used for the $P_{LlacO-1}$ were: 0, 1.9, 7.6, 30, 120, 490, and 1900 µM. Cells were grown in the same shaking-incubator conditions for six hours, and then 40 µL of culture was added to 160 µL of phosphate buffered saline (PBS) with 2 mg/mL kanamycin to halt protein production. Cells were incubated in PBS for one hour to allow YFP to mature, and then flow cytometry was performed. The fluorescence values were white-cell subtracted, and then plots of CI-GFP production versus GFP production were created for both P_{Tac} and $P_{LlacO-1}$ (see Section I.A.).

VIII.H. ***Construction and screening of RBS libraries.*** Mutations in the ribosomal binding site of several gates were introduced to shift the threshold of the gates' response curves. These RBS libraries were created using oligonucleotide primers containing multiple degenerate nucleotides in the 18 bases

immediately upstream from the start codon. These primers were used to amplify the entire gate characterization plasmid using two primers diverging from each other at the gate's RBS. 100 ng of linear dsDNA PCR product was phosphorylated and ligated in a one-pot reaction using 0.5 µL of T4 DNA ligase (New England Biolabs, MA, M0202S) and 0.5 µL of T4 polynucleotide kinase (New England Biolabs, MA, M0201S) in 10 µL 1X T4 ligase buffer, and then transformed into *E. coli* NEB 10-beta (New England Biolabs, MA, C3019). Individual clones from the gate RBS library were screened by growing them in the presence and absence of 1 mM IPTG. Clones with the highest ON/OFF range were chosen for further characterization. The full response functions of these gates were measured (Methods), and a subset of the gates that had unique threshold values were kept (see Section I.C).

VIII.I. ***Gate construction and characterization.*** To characterize gate response functions, the IPTG-inducible promoter P_{Tac} was positioned directly upstream from a gate expression cassette on the circuit backbone (without the 5'-insulating terminator L3S3P21). Following the P_{Tac}-driven gate expression cassette, the cognate promoter for the gate was positioned upstream from the standard RPU cassette. The plasmid backbone also encoded the sensors LacI and TetR in an operon driven from a constitutive promoter. These gate measurement plasmids were transformed into *E. coli* NEB 10-beta, and then a colony was inoculated into 200 µl of M9 glucose media with 50 µg/mL kanamycin in a V-bottom 96-well plate (Nunc, Roskilde, Denmark, 249952) and then grown for 16 hours at 37 °C shaking at 1000 rpm in an ELMI Digital Thermos Microplates shaker incubator (Elmi Ltd, Riga, Latvia). The next day, liquid culture was diluted 178-fold (two serial dilutions of 15 µL into 185 µL) into M9 glucose media with kanamycin and grown for three hours in the same shaking-incubator conditions. Subsequently, the culture was diluted 658-fold (two serial dilutions of 15 µL into 185 µL, then 3 µL into 145 µL) into M9 glucose media with kanamycin and IPTG to induce the gate. The IPTG concentrations used were used were: 0, 5, 10, 20, 30, 40, 50, 70, 100, 150, 200, and 1000 µM. Cells were grown for five hours in the same shaking-incubator conditions, and then 40 µL of culture was added to 160 µL of phosphate buffered saline (PBS) with 2 mg/mL kanamycin to halt protein production. These cells were incubated in PBS for one hour to allow YFP to mature, and then flow cytometry was performed (see Flow cytometry analysis), the data was converted to RPU (see Conversion of fluorescence to RPU, below). In addition to the gate characterization, we also measured a strain containing a similar plasmid that contained P_{Tac} driving the YFP RPU cassette directly; we converted its fluorescent output to RPU. Three independent replicates were performed on three separate days for each measurement, and the average RPU was calculated. We next plotted the average gate output RPU versus the average P_{Tac}-YFP RPU output to visualize the response of the gate output promoter activity as a function of P_{Tac} input promoter activity. This relationship was fit to a Hill equation (Equation S1) using the Solver add-in for Microsoft Excel.

VIII.J. ***Characterization of gate impact on cell growth.*** To quantify how growth is impacted by expression of various repressors, we constructed tandem inducible gate measurement plasmids to achieve higher levels of gate expression (pJS101-109, Supplementary Section IX). These plasmids are identical to the gate measurement plasmids (p[Gate-RBS#], Supplementary Section IX), with the exception that a tandem P_{Tac}-P_{Tet} promoter drives the gate expression cassette. We inoculated and grew these strains in an identical manner to the gate characterization experiments, except we used different inducer concentrations to span the wider inducible range of the tandem promoter. For each construct, we induced with seven IPTG concentrations: 0, 9.5, 19, 48, 95, 290, and 950 µM; for an additional five samples, we induced with 950 µM IPTG along with aTc at concentrations: 0.0095, 0.095, 0.29, 0.95, and 1.9 ng/mL. These induced cells were grown for six hours in the same shaking-incubator conditions. After the induction experiment, 200 µl of cells were added to an optically clear bottom 96 well plate. The optical density of the cultures was measured at 600 nm using a BioTek Synergy H1 Hybrid Microplate Reader. We also measured 200 µl of blank media to determine the background absorbance of the

media. For each gate, the final absorbance measurements were normalized to the absorbance of the first sample that had no inducer added (Figure 3d).

VIII.K. ***Flow cytometry analysis.*** Fluorescence was measured using an LSRII Fortessa flow cytometer (BD Biosciences, San Jose, CA) run by the BD FACSDiva software. An FSC voltage of 437 V, SSC voltage of 289 V, and a green laser (488 nm) voltage of 425 V were used. An SSC and FSC threshold of >200 was used to limit collection to cell-sized particles. Between 10^3 and 10^5 gated events were collected for analysis. To calculate YFP fluorescence values for bar graphs, we used the flow cytometry software FlowJo (TreeStar, Inc., Ashland, OR), and used the median statistical tool. For conversion of the cytometry data to RPU, we used MATLAB to perform the fluorescence-axis transformations and to normalize the distributions.

VIII.L. ***Conversion of fluorescence to RPU.*** The raw fluorescence from measurement of a sensor, gate, or circuit using the measurement protocol must be converted to relative expression units (RPU). The RPU standard used in this study differs from the Kelly standard (*27*) in that we use an upstream insulating terminator, a 5'-promoter spacer, and a ribozyme insulator to reduce contextual variations. We also maintain an identical RBS, YFP, terminator, and plasmid backbone to the circuit measurement constructs (pAN1717, Supplementary Section VI.A). *E. coli* NEB 10-beta (New England Biolabs, MA, C3019) were transformed with the RPU standard plasmid (see Section VI.A.) and plated on LB agar with 50 µM kanamycin. Transformed colonies were inoculated into 1 mL M9 glucose media with kanamycin and grown for 16 hours at 37 °C and shaking at 250 rpm. Next, the cells were diluted 178-fold (two serial dilutions of 15 µL into 185 µL) into M9 glucose media with kanamycin and grown for three hours in the same shaking-incubator conditions. Next, cells were diluted 658-fold (two serial dilutions of 15 µL into 185 µL and then 3 µL into 145 µL) M9 glucose media and grown for six hours in the same shaking-incubator conditions. At this point 40 µL of cells were added to 160 µL of phosphate buffered saline (PBS) with 2 mg/mL kanamycin to halt protein production. Cells were incubated in PBS for an hour to allow YFP to mature, and then flow cytometry was performed. Additionally, un-transformed *E. coli* NEB 10-beta cells ("white cells") were grown in an identical manner alongside the RPU standard-harboring strain, but without any antibiotics. These cells' autofluorescence were measured using flow cytometry as well. After flow cytometry as performed, the median of YFP fluorescence was calculated for the both the RPU standard and the white cells.

The median fluorescence measurements of sensors, gates, and circuits were converted to RPU using the following procedure. The median autofluorescence value from the white cells was first subtracted from all fluorescence values to correct for this non-YFP derived signal. In our experiments with our cytometer settings, the white cell fluorescence was approximately 15 au. The median fluorescence of the RPU standard was also subtracted by the white cell fluorescence. Next, the experimental sample (sensor measurement, etc.) as divided by the median fluorescence of the RPU standard (after autofluorescence correction). In our experiments, our corrected RPU standard fluorescence is 460 au. To return values to corrected arbitrary units, multiply the RPU numbers by the RPU standard's median (460 au for our measurements).

VIII.M. ***Genetic circuit assembly.*** The genetic circuits in this research comprise codon optimized repressors and their cognate promoters (from ref (*13*)) with additional 5'-promoter spacers, hammerhead ribozyme-based insulators (from ref (*2*) and this work), ribosomal binding sites (from this work), and transcriptional terminators (from ref (*6*)). The sensors used include a truncated AraC (AraC-C280*, referred to as AraC* in this work) that has reduced cross-talk with IPTG(*40*) and its output promoter P_{BAD} which is induced using L-arabinose; LacI and its output promoter with a symmetric lac-operator P_{Tac} which is IPTG-inducible(*41*); and TetR with its output promoter P_{Tet}(*42*). LacI and TetR are

transcribed from the native P_{LacI} promoter. AraC* is transcribed from BBa_J23105 and terminated by L3S3P22(*6*). The circuit measurement backbone harbors a medium-copy p15A origin of replication and kanamycin resistance gene (from ref (*42*)). The circuit insertion site is flanked by an upstream insulating terminator L3S3P21(*6*), and a downstream insulating terminator the native AraC terminator TaraC. The actuator used in this research is a variant of yellow fluorescent protein (YFP)(*30*). The output plasmid harbors a pSC101 origin of replication(*42*) and encodes the spectinomycin resistance gene, aadA. The output insertion site is flanked by an upstream insulating terminator L3S2P44(*6*) and a downstream insulating terminator L3S2P21(*6*). Each transcription unit for a circuit was cloned into a submodule plasmid with the ampicillin-resistance gene, ampR, and flanked on either side by 4bp scars and BbsI restriction enzyme recognition sites. To assemble a final circuit plasmid, submodule plasmids and the circuit measurement plasmid (with sensors already inserted) were mini-prepped prior to assembly (Qiagen, Limburg, 27104) and their concentration was measured using a NanoDrop 1000 spectrophotometer (Thermo Fisher Scientific, MA). In one tube, 40 fmol of each DNA plasmid were combined. In addition, 2 µL of ligase buffer, 0.5 µL of T4 DNA ligase HC (Promega, WI, M1794), and 2 µL of BbsI (New England Biolabs, MA, R0539L) were added to the tube (Figure S41). Lastly, filtered, deionized water was added to the tube to a total volume of 20 µL. This mixture was heated and cooled in a thermocycler repeatedly: 37 °C for 2 min, then 16 °C for 5 min, repeated for 10-100 cycles, depending on the number of pieces of DNA being assembled (10 cycles per piece of DNA). After the cycling, the reaction was heated to 50 °C for 10 minutes to inactivate the ligase, and then 80 °C to inactivate the restriction enzyme. Then, 10 µL of assembly mixture was then transformed into 50 µL of NEB 10-beta chemically competent *E. coli*, allowed to recover for an hour, and then plated on agar with antibiotics.

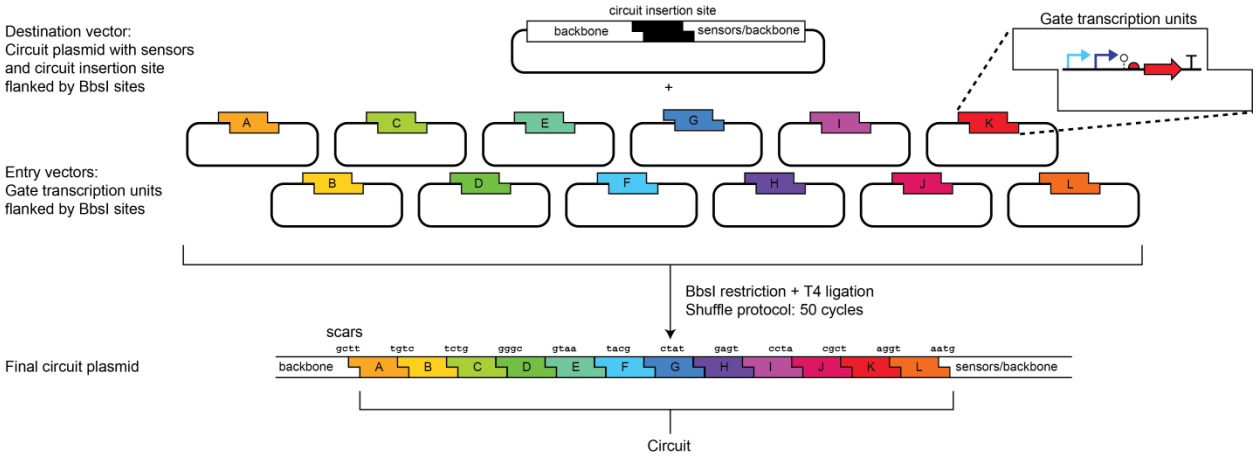

Figure S41: ***Genetic circuit assembly.*** *Final circuit plasmids are constructed from one or more submodule plasmids inserted into the circuit backbone containing the sensor effectors. Submodule plasmids comprise transcriptional units containing different genes (colored shapes correspond to genes). Each overhang is a 4bp sticky end scar generated in the Golden Gate assembly that connects the submodules together. The assembled circuit replaces a P_{Lac}-lacZα cassette (black shape) between the insertion scars "gctt" and "aatg".*

VIII.N. ***Hexadecimal and Wolfram Rule naming conventions.*** The convention for naming 3-input circuits is to first order the input states so that P_{Tac} activity is the least significant input bit, P_{Tet} is the middle significance input bit, and P_{BAD} is the most significant input bit. The corresponding expected output states for all inputs from 000 to 111 are converted to hexadecimal—four binary bits are converted to a single hexadecimal digit. The resulting two-digit hexadecimal number is listed after the hexadecimal indicator "0x" to create a name of the form "0xNN". This is similar to the Wolfram Rule

naming system, where the input rows are arranged 111, 110, ..., 001, 000 and then the binary output vector is converted to decimal (e.g., "Rule 110" has binary output vector *01101110*).

VIII.O. ***Software tools.*** The following software, languages, and libraries were used in this work. The Cello source code is written in Java (version 1.8.0_31) with software project management by Apache Maven (version 3.2.1). Constrained combinatorial designs of genetic architectures are produced using Eugene (version 2.0) (*25*), and the synthetic biology open language library (libSBOLj version 1.1) is used to store circuit designs in a hierarchy of annotated DNA components (*36*). Logic minimization uses Espresso (*20*) (version 2.3) and ABC(*19*) (UC Berkeley, version 1.01 March 2014).

Several figures for data visualization are generated during a Cello design run. Directed graphs are produced using Graphviz (version 2.34.0). Data plots for response function calculations and predicted output distributions are produced using Gnuplot (version 4.6). Part-based circuit representations are produced using Dnaplotlib, which uses the python matplotlib 2D plotting library for programmable rendering of highly customizable genetic diagrams. A static plasmid image is produced using EMBOSS cirdna (version 6.6.0.0).

The Cello web application is hosted using the Amazon Web Service (AWS) and is deployed using the Jetty web server (version 8.1.13). The browser (client-side) sends data to and retrieves data from the Amazon server (server-side) using AJAX (Asynchronous JavaScript and XML). The web interface uses JavaScript, jQuery, and jQuery UI (version 1.10.2) for user-interactive event handling and dynamic interface manipulation. CSS Bootstrap (version 2.3.1) is used to style the content, and CodeMirror (version 3.13) is used for Verilog syntax highlighting.

For parameterizing Hill equations to response functions, the fit was performed by minimizing the sum of relative error magnitudes between the trend line and the data points using the Excel Solver add-in with the GRG Nonlinear solving method. The initial version of Cello that performed gate assignments by simulating input signal propagation through response functions (Supplementary Information Section II.C) was implemented using MATLAB version R2012a (7.14.0.739). Design of the Cello circuits library used the distribution propagation (Supplementary Information Section V.F) to screen for circuits, and used the simulated annealing algorithm with a temperature of 0. Additionally, assignments using QacR, LitR, IcaRA, PsrA, and LmrA repressors were disallowed.

Ribozyme secondary structure was simulated using RNA mFold (*12*) (http://mfold.rna.albany.edu/?q=mfold/RNA-Folding-Form) using the following parameters: 37°C, 1M NaCl, 5 percent suboptimality folds computed, 50 maximum computed foldings, maximum interior bulge/loop size = 30, maximum asymmetry of an interior bulge/loop = 30, no limit for maximum distance between paired bases. Sequence alignments were performed with Clustal Omega (1.2.1) multiple DNA sequence alignment using the default parameters (http://www.ebi.ac.uk/Tools/msa/clustalo/). The ribozyme phylogenetic tree was also generated using Clustal Omega using the default tree format, distance correction off, exclude gaps off, the UPGMA clustering method, and the "real" phylogram branch length setting.

VIII.P. ***Precomputing 3-input 1-output NOR circuit diagrams.*** In the library of user-defined circuit motifs (Section V.C), we used a precomputed list of small 3-input 1-output NOR/NOT circuits. These circuits were found by computationally enumerating all NOR/NOT circuits with ≤ 6 layers, evaluating each circuit's truth table, and then selecting the circuit with the fewest number of gates for each truth table. We computationally enumerated all circuits by constructing them in levels. Level 1 comprises the circuit input wires: A, B, C, and 0, where 0 is a Boolean 'false'. Note that when 0 is one of the inputs to a NOR gate, this results in a NOT gate for the other input. To enumerate all circuits in Level 2, all pairwise combinations of Level 1 output wires (A, B, C, and 0) are input into a NOR gate. For example, (A NOR B)

is a Level 2 circuit. To enumerate all circuits in Level 3, all pairwise combinations of wires containing an output from Level 2 and an output from any level are input into a NOR gate. For example, ((A NOR B) NOR A) is an example of a Level 3 circuit. If the two circuits being joined have a duplicate logic motif (in the previous example, input A was specified twice), a fan-out wire is used and the redundant gates are removed. This process was continued until all Level 6 circuits were enumerated. After each individual circuit construction, the circuit's truth table output was evaluated. If the circuit used fewer gates than all previous circuits implementing that truth table, the circuit was stored until a smaller one was found. This algorithm resulted in small motifs for each 3-input 1-output circuit. We used these motifs for subcircuit replacement in the final step of logic synthesis.

VIII.Q. ***RPU plasmid characterization using smRNA-FISH.*** To determine the steady state number of *yfp* mRNA copies per cell at mid-exponential growth with the RPU standard plasmid (pAN1717), we used smRNA-FISH to label the *yfp* mRNA molecules. We designed a set of 25 oligonucleotide probes, fluorescently labeled with TAMRA, each 20 bases in length, against the *yfp* transcript (Table S10) using Stellaris Probe Designer version 4.1. Three independent replicates were performed on three separate days for each measurement, and the average number of *yfp* mRNA/cell was calculated.

VIII.Q.1 ***Sample preparation.*** The RPU plasmid (pAN1717), the non-YFP plasmid (pAN1201) and the measurement plasmid (pAN1818) were transformed to create an RPU standard, a background control, and a standard curve for mRNA/cell estimates, respectively. The plasmids were transformed in separate reactions into *E. coli* NEB 10-beta (New England Biolabs, MA, C3019), grown on LB + 50 µg/mL kanamycin agar plates, and then a colony was inoculated into 200 µL of M9 minimal media with 50 µg/mL kanamycin in a V-bottom 96-well plate (Nunc, Roskilde, Denmark, 249952). The cultures were grown for 16 hours at 37 °C shaking at 1000 rpm in an ELMI Digital Thermos Microplates shaker incubator (Elmi Ltd, Riga, Latvia). The next day, the liquid culture was diluted 178-fold into M9 minimal media with kanamycin and grown for three hours in the same shaking-incubator conditions. Subsequently, the culture was diluted 658-fold into M9 minimal media with kanamycin and IPTG to induce the YFP production from the pAN1818 plasmid. The IPTG concentrations used were: 0, 5, 10, 20, 30, 40, 50, 70, 100, 150, 200, and 1000 µmol/L. Cells were grown for five hours in the same shaking-incubator conditions, and then 6 mL of culture per sample was pooled together in 15 mL Corning centrifuge tube and the cells were pelleted by centrifugation (10 minutes, 4000×g, 4 °C). The supernatant was removed and the cells were resuspended in 1 mL 1× PBS (diluted from 10× PBS, Ambion, #AM9625) to wash the cells.

The cells were transferred to RNase free microfuge tubes and pelleted by centrifugation (5 minutes, 4500×g, 4 °C). The supernatant was removed and the cells were resuspended in 1 mL freshly prepared 3.7% formaldehyde (Fisher, #BP531) in 1× PBS (diluted from 10× PBS). The cells were then mixed on a nutator at room temperature for 30 minutes. The cells were pelleted by centrifugation (8 minutes, 400×g). The supernatant was removed and the cells were washed in 1 mL 1× PBS twice (i.e. resuspended in 1 mL 1× PBS, centrifuged at 600×g for 3.5 minutes, and supernatant removed). The cells were resuspended in 300 µL water and then 700 µL of 100% ethanol was added and mixed twice to get to a final concentration of 70% ethanol. The cells were left at room temperature with mixing on a nutator for 1 hour to permeabilize the cell membrane.

VIII.Q.2 ***Hybridization procedure.*** After permeabilization, cells were centrifuged (7 minutes, 600×*g*) and the supernatant was removed. The cells were resuspended in 1 mL of 50% formamide wash buffer A (Biosearch Technologies Cat no SMF-WA1-60). Reagents containing formamide were prepared fresh, right before use. The stock formamide was stored at -20 °C in 1.5 mL aliquots and thawed right before use. Next, 50% formamide hybridization buffer (Biosearch Technologies Cat no SMF-HB1-10) was prepared by adding 12.5 µmol/L mixed probe stock to make a final 62.5 µmol/L probes concentration. The cells were then centrifuged (7 minutes, 600×*g*) and the supernatant was removed. The cells were resuspended in 50 µL of the 50% formamide hybridization buffer with probes and left to incubate in the dark at 30 °C overnight. Next, 400 µL of 50% formamide Wash Buffer A was added to the tube and mixed well. Cells were pelleted by centrifugation (7 minutes, 600×*g*) and the supernatant was removed. The cells were washed 3 more times (i.e. resuspended in 200 µL of 50% formamide Wash Buffer A, incubated at 30°C for 30 minutes, centrifuged at 600×*g* for 3.5 minutes, and supernatant removed). 4',6-diamidino-2-phenylindole (DAPI, Fisher Scientific, #PI-46190) was added to the wash solution to a final concentration of 10 µg/mL in the last wash. The cells were resuspended in 500 µL of Wash Buffer B (Biosearch Technologies Cat no SMF-WB1-20), centrifuged at 600×*g* for 3.5 minutes, and supernatant removed. The cells were resuspended in 40 µL to 50 µL of 2×SSC and imaged under the microscope.

*VIII.Q.3.****Microscopy.*** 2 µL of sample was pipetted onto a 45 mm × 50 mm #1 coverslip (Fisher Scientific, #12-544F). A 1 mm thick × 10 mm × 7 mm 1.5% agarose gel pad (in 1× PBS) was laid on the sample. A 22 mm × 22 mm #1 coverslip (Fisher Scientific, #12-545B) was placed on top of the agarose gel pad. The sample was imaged using an inverted epifluorescence microscope (Zeiss Axio Observer.Z1), a 100×, N.A. 1.46 oil immersion objective (Zeiss, alpha-Plan APO), and a cooled digital CMOS camera (Hamamatsu Orca Flash 4.0). The microscope and camera were controlled using the Zen Pro Software (Zeiss). The mRNA labeled by smFISH probes were imaged using a TAMRA filter set (Zeiss, 43 HE), a HXP-200 excitation light source set on 50% intensity, and an integration time of 1 s. DNA stained by DAPI was also imaged using a multi-band filter set (Zeiss, 81 HE), 353 nm excitation with an LED source (Zeiss, Colibri) set to 100%, and an integration time of 50 ms. *Z*-stacks with 9 slices and 200 nm spacing were acquired for bright field and TAMRA images. Each sample was imaged at multiple locations to get a total of at least 300 cells per sample.

*VIII.Q.4.****Image and data analysis.*** Image processing and data analysis were performed using MATLAB and Mathematica. Cell recognition and segmentation was performed on brightfield images of cells using the *Schnitzcells* MATLAB module (*43*). The program applies edge detection and other morphological operations, using the MATLAB Image Processing Toolbox. The output was checked and corrected using the manual interface offered by *Schnitzcells*.

Spot recognition was performed on the segmented TAMRA fluorescence images using the *Spatzcells* MATLAB module (*32*). The *Spatzcells* software detects each fluorescent spot within the segmented cell image stacks, finds its location (x, y, z-slice), and fits it to a 2D-Gaussian function to obtain the height and intensity of the spot.

VIII.Q.5. ***Estimating mRNA copy numbers.*** For cells with low mRNA copy number, the smFISH spots are typically well separated within the cells, and so they can be visualized and counted as individual spots (see example images in Figure S33a). For cells with higher copy number, the spots overlap significantly. Quantitative estimation of the copy number for a full range of expression levels therefore requires a method for extrapolating from the low expression regime to the high expression regime. The mRNA target described in (*32*) was relatively long (~ 3000 base pairs), so that 72 different probes (each ~20 nucleotides long) could be designed to span the length of the target sequence. The mRNA target for the measurement described here (*yfp*) is considerably shorter (720 base pairs) so that only 20 different 20 nucleotide probes could be designed to span the target sequence (see Table S10). Consequently, the spot intensities (the heights of the fitted 2D-Gaussians) for the bright spots corresponding to intact *yfp* mRNAs were only partially resolved from the background, lower-intensity spots. Because of the partial overlap in the typical spot heights for the labeled mRNA and background spots, the thresholding method described in (*32*) for distinguishing the two types of spots did not work reliably. Therefore, a new method for extrapolation to high expression levels was used, based on the assumption that the total FISH fluorescence signal measured for each cell is a linear function of the number of mRNAs in the cell.

Briefly, for each sample, a histogram was constructed of the spot heights for all detected spots (see Figure S33b), and the histogram was fitted to a sum of two log-normal distributions to obtain estimates of the total number of dim spots and bright spots for that sample. For low-expression samples, the two spot populations were partially resolved and the fit was unconstrained (see, for example, Figure S33b, panel 2, "P_{Tac}-YFP (10 μM IPTG)"). The fits to the low expression sample histograms were used to define a constraint value equal to the mid-way point between the locations of the two fitted log-normals.

For higher expression, the location parameters for the 1st log-normal distribution was constrained to be less than the constraint value and the location parameter for the 2nd log-normal was constrained to be greater than the constraint value. The total number of bright spots for each sample was estimated from the integrated area of the 2nd log-normal (the one corresponding to the brighter spots) from each fit. The number of bright spots divided by the total cell area was then plotted vs. the average FISH signal per cell area, and the result was fit to a form assuming a Poisson filling process of the available image area with spots:

$n_{spots} = n_{max}\left(1 - \exp\left(-\frac{x-\beta}{\alpha \cdot n_{max}}\right)\right)$, where n_{spots} is the number of bright spots per cell area, x is the FISH signal per cell area, and the fitting parameters: n_{max} is the maximum number of resolvable spots per cell area, α is the typical FISH signal for a single mRNA, and β is and the background FISH signal per cell area (see Figure S33c). The linear portion of the fit curve was then used to extrapolate to higher expression levels, giving an estimate for the mRNA copy number for each cell: $N_{est.} = \frac{S-\beta \cdot A}{\alpha}$, where S is the total FISH signal for the cell and A is the image area of the cell (pixels). Example histograms of the estimated mRNA copy number are shown in Figure S33d. The mean mRNA copy number per cell was calculated for each sample.

The estimation procedure (including fitting to the spot height histograms, estimation of number of bright spots, fitting with the Poisson filling process model, and estimation of the mean mRNA copy number per cell) was done independently for each replicate experiment. The values obtained from the

replicate measurements were then averaged to produce the final mRNA copy number per cell estimates as shown in Figure S33e.

IX. Plasmid maps and part sequences

IX.A. Plasmid maps

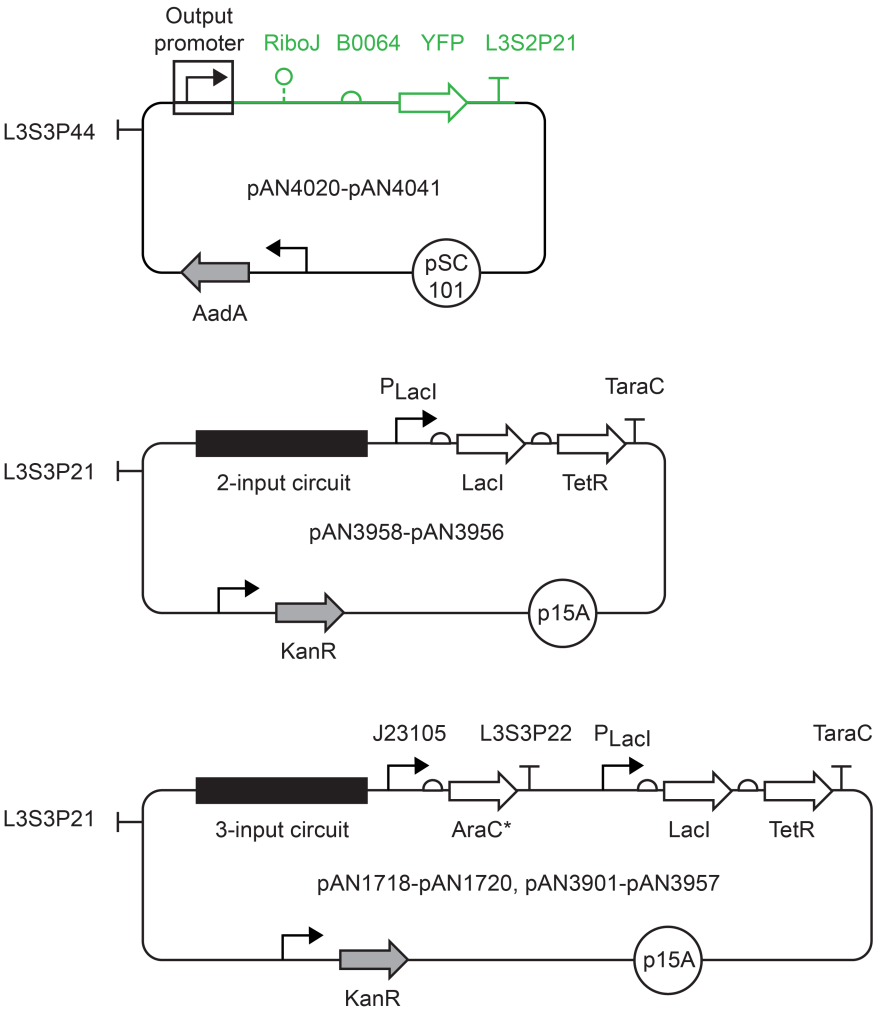

Figure S42: ***Plasmid backbones for measuring circuits.*** *Top: output plasmid. Middle: 2-input circuit backbone. Bottom 3-input circuit backbone.*

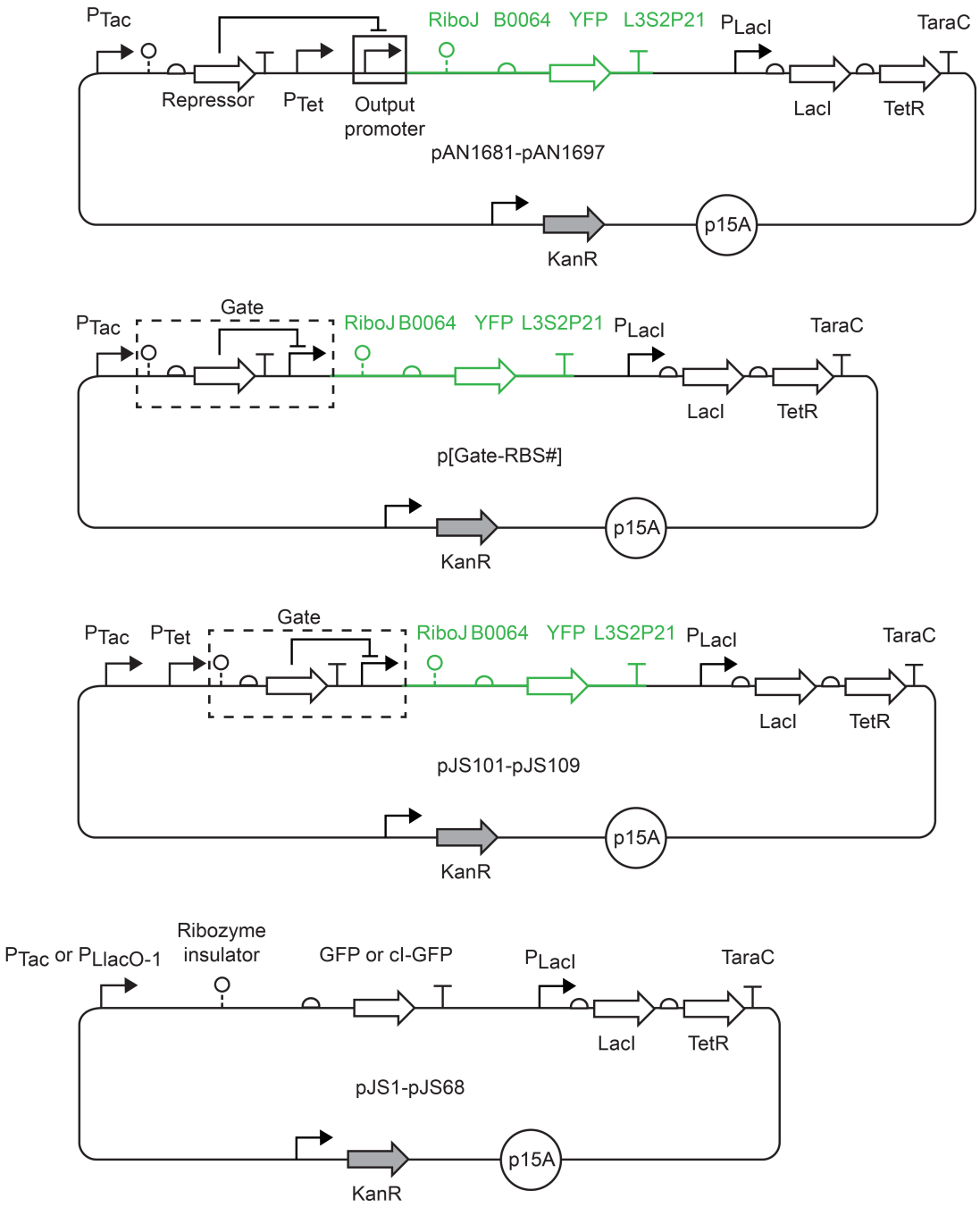

Figure S43: ***Gate characterization plasmids.*** *Top: roadblocking test plasmids. Second from top: Insulated gate measurement plasmids. Third from top: toxicity measurement plasmids. Bottom: Ribozyme test plasmids.*

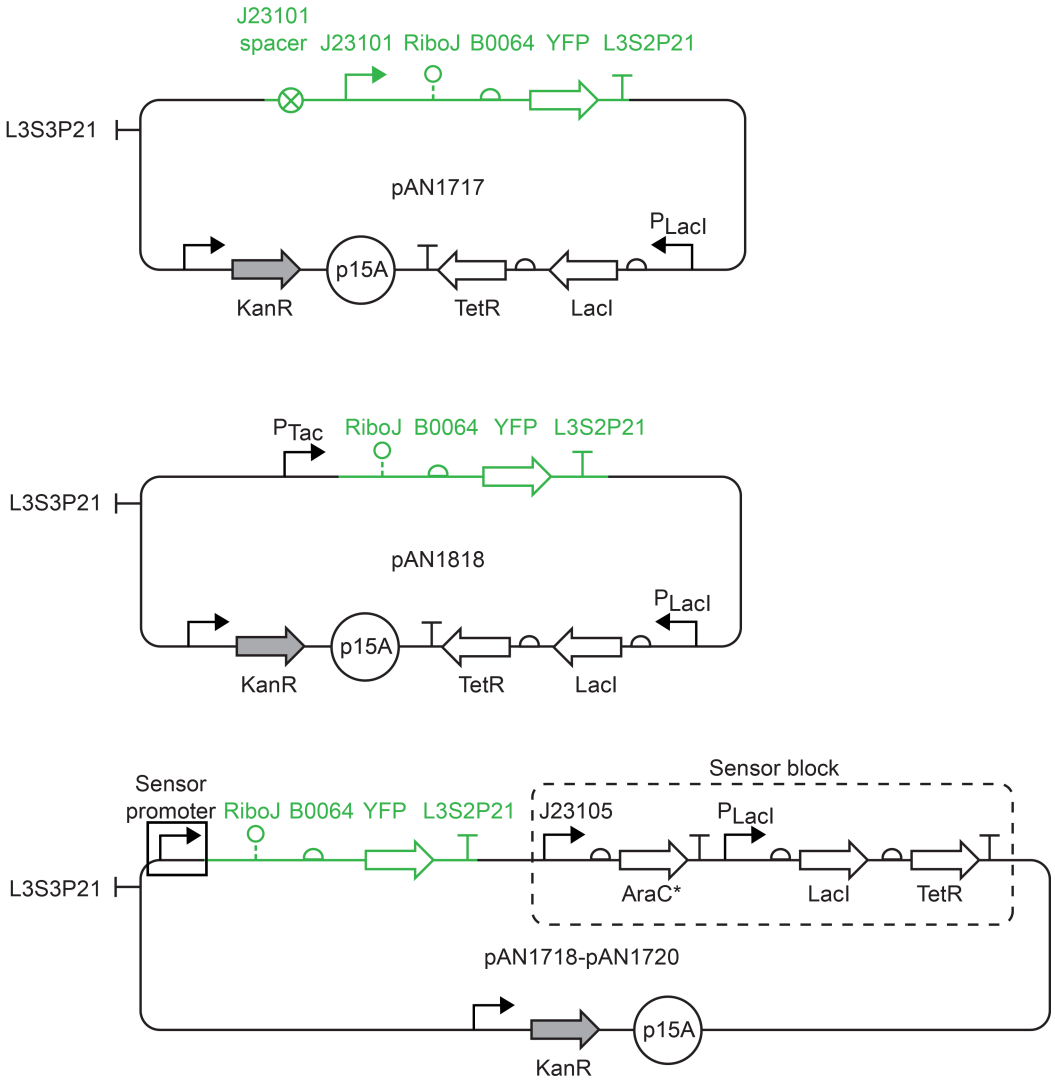

Figure S44: ***Sensor measurement and RPU plasmids.*** *Top: RPU standard plasmid. Middle: P_{Tac} activity plasmid. Bottom: Sensor measurement plasmids.*

IX.B. DNA sequences

Table S7: List of plasmids used in this work

Plasmid name	Description
pAN215	Non-insulated LmrA NOT gate
pAN216	Non-insulated LmrA NOR gate
pAN412	Insulated LmrA NOT gate
pAN413	Insulated LmrA NOR gate
pAN901	Circuit plasmid (no insulating terminator): A IMPLY B v1
pAN902	Circuit plasmid (no insulating terminator): B IMPLY A v1
pAN903	Circuit plasmid (no insulating terminator): A NIMPLY B v1
pAN904	Circuit plasmid (no insulating terminator): B NIMPLY A v1
pAN905	Circuit plasmid (no insulating terminator): NAND v1
pAN906	Circuit plasmid (no insulating terminator): AND v1
pAN907	Circuit plasmid (no insulating terminator): XOR v1
pAN908	Circuit plasmid (no insulating terminator): XNOR v1
pAN1201	Circuit bakcbone plasmid (LacZa insert)
pAN1250	Circuit plasmid (no insulating terminator): PTac-YFP
pAmeR-F1	Insulated AmeR NOT gate, RBS F1
pAmtR-A1	Insulated AmtR NOT gate, RBS A1
pBetI-E1	Insulated BetI NOT gate, RBS E1
pBM3R1-B1	Insulated BM3R1 NOT gate, RBS B1
pBM3R1-B2	Insulated BM3R1 NOT gate, RBS B2
pBM3R1-B3	Insulated BM3R1 NOT gate, RBS B3
pHlyIIR-H1	Insulated HlyIIR NOT gate, RBS H1
pIcaRA-I1	Insulated IcaRA NOT gate, RBS I1
pLitR-L1	Insulated LitR NOT gate, RBS L1
pLmrA-N1	Insulated LmrA NOT gate, RBS N1
pPhlF-P1	Insulated PhlF NOT gate, RBS P1
pPhlF-P2	Insulated PhlF NOT gate, RBS P2
pPhlF-P3	Insulated PhlF NOT gate, RBS P3
pPsrA-R1	Insulated PsrA NOT gate, RBS R1
pQacR-Q1	Insulated QacR NOT gate, RBS Q1
pQacR-Q2	Insulated QacR NOT gate, RBS Q2
pSrpR-S1	Insulated SrpR NOT gate, RBS S1
pSrpR-S2	Insulated SrpR NOT gate, RBS S2
pSrpR-S3	Insulated SrpR NOT gate, RBS S3
pSrpR-S4	Insulated SrpR NOT gate, RBS S4
pAN1681	Roadblocking test: PTet-YFP
pAN1682	Roadblocking test: PTet-PTac-YFP
pAN1683	Roadblocking test: PTet-PLux*-YFP
pAN1684	Roadblocking test: PTet-PBAD-YFP
pAN1685	Roadblocking test: PTet-PPhlF-YFP
pAN1686	Roadblocking test: PTet-PSrpR-YFP
pAN1687	Roadblocking test: PTet-PBM3R1-YFP
pAN1688	Roadblocking test: PTet-PBetI-YFP
pAN1689	Roadblocking test: PTet-PQacR-YFP
pAN1690	Roadblocking test: PTet-PAmtR-YFP
pAN1691	Roadblocking test: PTet-PHlyIIR-YFP
pAN1692	Roadblocking test: PTet-PIcaRA-YFP
pAN1693	Roadblocking test: PTet-PAmeR-YFP
pAN1694	Roadblocking test: PTet-PLitR-YFP
pAN1695	Roadblocking test: PTet-PPsrA-YFP
pAN1696	Roadblocking test: PTet-PLmrA-YFP
pAN1697	Roadblocking test: PTac-PTet-YFP
pAN1717	RPU standard plasmid
pAN1718	Circuit plasmid: PTac-YFP (constitutive AraC*, LacI, TetR)
pAN1719	Circuit plasmid: PTet-YFP (constitutive AraC*, LacI, TetR)
pAN1720	Circuit plasmid: PBAD-YFP (constitutive AraC*, LacI, TetR)
pAN1818	Circuit plasmid: PTac-YFP (constitutive LacI)
pAN3901	Circuit plasmid: 0xEA
pAN3902	Circuit plasmid: 0x70
pAN3903	Circuit plasmid: 0x3B
pAN3904	Circuit plasmid: 0x7F
pAN3905	Circuit plasmid: 0xC8
pAN3906	Circuit plasmid: 0x07
pAN3907	Circuit plasmid: 0x37
pAN3908	Circuit plasmid: 0xF7
pAN3909	Circuit plasmid: 0xAE
pAN3910	Circuit plasmid: 0x80
pAN3911	Circuit plasmid: 0xC4
pAN3912	Circuit plasmid: 0x0E
pAN3913	Circuit plasmid: 0xCD
pAN3914	Circuit plasmid: 0x0B
pAN3915	Circuit plasmid: 0xFB
pAN3916	Circuit plasmid: 0x08
pAN3917	Circuit plasmid: Multiplexer alternative assignment
pAN3918	Circuit plasmid: Multiplexer
pAN3919	Circuit plasmid: 0xC7
pAN3920	Circuit plasmid: 0x6E
pAN3921	Circuit plasmid: 0x8E
pAN3922	Circuit plasmid: 0x9F
pAN3923	Circuit plasmid: 0x87
pAN3924	Circuit plasmid: 0x04
pAN3925	Circuit plasmid: 0x38
pAN3926	Circuit plasmid: 0x6F

pAN3927	Circuit plasmid: 0xE8
pAN3928	Circuit plasmid: 0x78
pAN3929	Circuit plasmid: 0x4D
pAN3930	Circuit plasmid: 0xBD
pAN3931	Circuit plasmid: 0xF9
pAN3932	Circuit plasmid: 0x60
pAN3933	Circuit plasmid: 0x3D
pAN3934	Circuit plasmid: 0xB9
pAN3935	Circuit plasmid: 0x19
pAN3936	Circuit plasmid: 0x01
pAN3937	Circuit plasmid: Majority alternative assignment
pAN3938	Circuit plasmid: 0xF6
pAN3939	Circuit plasmid: 0x98
pAN3940	Circuit plasmid: 0xC6
pAN3941	Circuit plasmid: 0x82
pAN3942	Circuit plasmid: 0x06
pAN3943	Circuit plasmid: 0x36
pAN3944	Circuit plasmid: 0x1C
pAN3945	Circuit plasmid: 0x41
pAN3946	Circuit plasmid: 0xC9
pAN3947	Circuit plasmid: 0xC1
pAN3948	Circuit plasmid: Consensus alternative assignment
pAN3949	Circuit plasmid: Consensus
pAN3950	Circuit plasmid: Priority detector
pAN3951	Circuit plasmid: Demultiplexer
pAN3952	Circuit plasmid: Majority #1 - Original
pAN3953	Circuit plasmid: Majority #2 - Reversed order
pAN3954	Circuit plasmid: Majority #3 - NOTs and NORs clustered
pAN3955	Circuit plasmid: Majority #4 - Subcircuits clustered
pAN3956	Circuit plasmid: Majority #5 - Scrambled order
pAN3957	Circuit plasmid: Majority #6 - Alternating orientation
pAN3958	Circuit plasmid: A IMPLY B v2
pAN3959	Circuit plasmid: B IMPLY A v2
pAN3960	Circuit plasmid: A NIMPLY B v2
pAN3961	Circuit plasmid: B NIMPLY A v2
pAN3962	Circuit plasmid: NAND v2
pAN3963	Circuit plasmid: AND v2
pAN3964	Circuit plasmid: XOR v2
pAN3965	Circuit plasmid: XNOR v2
pAN4020	Output bakcbone plasmid (LacZalpha insert)
pAN4021	Output plasmid: PTac-YFP
pAN4022	Output plasmid: PTet-YFP
pAN4023	Output plasmid: PBM3R1-YFP
pAN4024	Output plasmid: PBetI-YFP
pAN4025	Output plasmid: PAmeR-YFP
pAN4026	Output plasmid: PPhlF-YFP
pAN4027	Output plasmid: PSrpR-YFP
pAN4028	Output plasmid: PTac-PAmeR-YFP
pAN4029	Output plasmid: PTac-PAmtR-YFP
pAN4030	Output plasmid: PTet-PAmtR-YFP
pAN4031	Output plasmid: PBAD-PAmtR-YFP
pAN4032	Output plasmid: PBM3R1-PHlyIIR-YFP
pAN4033	Output plasmid: PBM3R1-PTet-YFP
pAN4034	Output plasmid: PBetI-PAmeR-YFP
pAN4035	Output plasmid: PPhlF-PTet-YFP
pAN4036	Output plasmid: PPhlF-PAmtR-YFP
pAN4037	Output plasmid: PPhlF-PBetI-YFP
pAN4038	Output plasmid: PPhlF-PHlyIIR-YFP
pAN4039	Output plasmid: PSrpR-PAmtR-YFP
pAN4040	Output plasmid: PSrpR-PBetI-YFP
pAN4041	Output plasmid: PSrpR-PHlyIIR-YFP
pJS1	Ribozyme test plasmid: pTac-GFP
pJS2	Ribozyme test plasmid: pTac-CI-GFP
pJS3	Ribozyme test plasmid: pLlacO-1-GFP
pJS4	Ribozyme test plasmid: pLlacO-1-CI-GFP
pJS5	Ribozyme test plasmid: pTac-RiboJ00-GFP
pJS6	Ribozyme test plasmid: pTac-RiboJ00-CI-GFP
pJS7	Ribozyme test plasmid: pLlacO-1-RiboJ00-GFP
pJS8	Ribozyme test plasmid: pLlacO-1-RiboJ00-CI-GFP
pJS9	Ribozyme test plasmid: pTac-RiboJ10-GFP
pJS10	Ribozyme test plasmid: pTac-RiboJ10-CI-GFP
pJS11	Ribozyme test plasmid: pLlacO-1-RiboJ10-GFP
pJS12	Ribozyme test plasmid: pLlacO-1-RiboJ10-CI-GFP
pJS13	Ribozyme test plasmid: pTac-RiboJ51-GFP
pJS14	Ribozyme test plasmid: pTac-RiboJ51-CI-GFP
pJS15	Ribozyme test plasmid: pLlacO-1-RiboJ51-GFP
pJS16	Ribozyme test plasmid: pLlacO-1-RiboJ51-CI-GFP
pJS17	Ribozyme test plasmid: pTac-RiboJ53-GFP
pJS18	Ribozyme test plasmid: pTac-RiboJ53-CI-GFP
pJS19	Ribozyme test plasmid: pLlacO-1-RiboJ53-GFP
pJS20	Ribozyme test plasmid: pLlacO-1-RiboJ53-CI-GFP
pJS21	Ribozyme test plasmid: pTac-RiboJ54-GFP
pJS22	Ribozyme test plasmid: pTac-RiboJ54-CI-GFP
pJS23	Ribozyme test plasmid: pLlacO-1-RiboJ54-GFP
pJS24	Ribozyme test plasmid: pLlacO-1-RiboJ54-CI-GFP
pJS25	Ribozyme test plasmid: pTac-RiboJ57-GFP
pJS26	Ribozyme test plasmid: pTac-RiboJ57-CI-GFP
pJS27	Ribozyme test plasmid: pLlacO-1-RiboJ57-GFP
pJS28	Ribozyme test plasmid: pLlacO-1-RiboJ57-CI-GFP
pJS29	Ribozyme test plasmid: pTac-RiboJ60-GFP

pJS30	Ribozyme test plasmid: pTac-RiboJ60-CI-GFP
pJS31	Ribozyme test plasmid: pLlacO-1-RiboJ60-GFP
pJS32	Ribozyme test plasmid: pLlacO-1-RiboJ60-CI-GFP
pJS33	Ribozyme test plasmid: pTac-RiboJ64-GFP
pJS34	Ribozyme test plasmid: pTac-RiboJ64-CI-GFP
pJS35	Ribozyme test plasmid: pLlacO-1-RiboJ64-GFP
pJS36	Ribozyme test plasmid: pLlacO-1-RiboJ64-CI-GFP
pJS37	Ribozyme test plasmid: pTac-AraJ-GFP
pJS38	Ribozyme test plasmid: pTac-AraJ-CI-GFP
pJS39	Ribozyme test plasmid: pLlacO-1-AraJ-GFP
pJS40	Ribozyme test plasmid: pLlacO-1-AraJ-CI-GFP
pJS41	Ribozyme test plasmid: pTac-BydvJ-GFP
pJS42	Ribozyme test plasmid: pTac-BydvJ-CI-GFP
pJS43	Ribozyme test plasmid: pLlacO-1-BydvJ-GFP
pJS44	Ribozyme test plasmid: pLlacO-1-BydvJ-CI-GFP
pJS45	Ribozyme test plasmid: pTac-CchJ-GFP
pJS46	Ribozyme test plasmid: pTac-CchJ-CI-GFP
pJS47	Ribozyme test plasmid: pLlacO-1-CchJ-GFP
pJS48	Ribozyme test plasmid: pLlacO-1-CchJ-CI-GFP
pJS49	Ribozyme test plasmid: pTac-ElvJ-GFP
pJS50	Ribozyme test plasmid: pTac-ElvJ-CI-GFP
pJS51	Ribozyme test plasmid: pLlacO-1-ElvJ-GFP
pJS52	Ribozyme test plasmid: pLlacO-1-ElvJ-CI-GFP
pJS53	Ribozyme test plasmid: pTac-LtsvJ-GFP
pJS54	Ribozyme test plasmid: pTac-LtsvJ-CI-GFP
pJS55	Ribozyme test plasmid: pLlacO-1-LtsvJ-GFP
pJS56	Ribozyme test plasmid: pLlacO-1-LtsvJ-CI-GFP
pJS57	Ribozyme test plasmid: pTac-PlmJ-GFP
pJS58	Ribozyme test plasmid: pTac-PlmJ-CI-GFP
pJS59	Ribozyme test plasmid: pLlacO-1-PlmJ-GFP
pJS60	Ribozyme test plasmid: pLlacO-1-PlmJ-CI-GFP
pJS61	Ribozyme test plasmid: pTac-SarJ-GFP
pJS62	Ribozyme test plasmid: pTac-SarJ-CI-GFP
pJS63	Ribozyme test plasmid: pLlacO-1-SarJ-GFP
pJS64	Ribozyme test plasmid: pLlacO-1-SarJ-CI-GFP
pJS65	Ribozyme test plasmid: pTac-ScmJ-GFP
pJS66	Ribozyme test plasmid: pTac-ScmJ-CI-GFP
pJS67	Ribozyme test plasmid: pLlacO-1-ScmJ-GFP
pJS68	Ribozyme test plasmid: pLlacO-1-ScmJ-CI-GFP
pJS101	Toxicity test plasmid: pTac-pTet-SrpR
pJS102	Toxicity test plasmid: pTac-pTet-PhlF
pJS103	Toxicity test plasmid: pTac-pTet-BM3R1
pJS104	Toxicity test plasmid: pTac-pTet-AmtR
pJS105	Toxicity test plasmid: pTac-pTet-HlyIIR
pJS106	Toxicity test plasmid: pTac-pTet-LitR
pJS107	Toxicity test plasmid: pTac-pTet-QacR
pJS108	Toxicity test plasmid: pTac-pTet-IcaRA
pJS109	Toxicity test plasmid: pTac-pTet-BetI

Table S8: Sequences of insulated gates and sensor modules used in this work

Part name	Type	DNA sequence[a]
AmeR (RBS-F1) gate	gate	CTGAAGGGGTCAGTTGATGTGCTTTCAACTCTGATGAGTCAGTGATGACGAAACCCCCTCTACAAATAATTTTGTTT AACTATGGACTATGTTTTCACATACGAGGGGGATTAGATGAACAAAACCATTGATCAGGTGCGTAAAGGTGATCGTA AAAGCGATCTGCCGGTTCGTCGTCGTCCGCGTCGTAGTGCCGAAGAAACCCGTCGTGATATTCTGGCAAAAGCCGAA GAACTGTTTCGTGAACGTGGTTTTAATGCAGTTGCCATTGCAGATATTGCAAGCGCACTGAATATGAGTCCGGCAAA TGTGTTTAAACATTTTAGCAGCAAAAACGCACTGGTTGATGCAATTGGTTTTGGTCAGATTGGTGTTTTTGAACGTC AGATTTGTCCGCTGGATAAAAGCCATGCACCGCTGGATCGTCTGCGTCATCTGGCACGTAATCTGATGGAACAGCAT CATCAGGATCATTTCAAACACATACGGGTTTTTATTCAGATCCTGATGACCGCCAAACAGGATATGAAATGTGGCGA TTATTACAAAAGCGTGATTGCAAAACTGCTGGCCGAAATTATTCGTGATGGTGTTGAAGCAGGTCTGTATATTGCAA CCGATATTCCGGTTCTGGCAGAAACCGTTCTGCATGCACTGACCAGCGTTATTCATCCGGTTCTGATTGCACAAGAA GATATTGGTAATCTGGCAACCCGTTGTGATCAGCTGGTTGATCTGATTGATGCAGGTCTGCGTAATCCGCTGGCAAA ATAACCAATTATTGAACACCCTAACGGGTGTTTTTTTTTTTTTGGTCTACCTCGTCACTAGAGGGCGATAGTGACAA ACTTGACAACTCATCACTTCCTAGGTATAATGCTAGC
AmtR (RBS-A1) gate	gate	CTGAAGGGTGTCTCAAGGTGCGTACCTTGACTGATGAGTCCGAAAGGACGAAACACCCCTCTACAAATAATTTTGTT TAAAATGTTCCCTAATAATCAGCAAAGAGGTTACTAGATGGCAGGCGCAGTTGGTCGTCCGCGTCGTAGTGCACCGC GTCGTGCAGGTAAAAATCCGCGTGAAGAAATTCTGGATGCAAGCGCAGAACTGTTTACCCGTCAGGGTTTTGCAACC ACCAGTACCCATCAGATTGCAGATGCAGTTGGTATTCGTCAGGCAAGCCTGTATTATCATTTTCCGAGCAAAACCGA AATCTTTCTGACCCTGCTGAAAAGCACCGTTGAACCGAGCACCGTTCTGGCAGAAGATCTGAGCACCCTGGATGCAG GTCCGGAAATGCGTCTGTGGGCAATTGTTGCAAGCGAAGTTCGTCTGCTGCTGAGCACCAAATGGAATGTTGGTCGT CTGTATCAGCTGCCGATTGTTGGTAGCGAAGAATTTGCAGAATATCATAGCCAGCGTGAAGCACTGACCAATGTTTT TCGTGATCTGGCAACCGAAATTGTTGGTGATGATCCGCGTGCAGAACTGCCGTTTCATATTACCATGAGCGTTATTG AAATGCGTCGCAATGATGGTAAAATTCCGAGTCCGCTGAGCGCAGATAGCCTGCCGGAAACCGCAATTATGCTGGCA GATGCAAGCCTGGCAGTTCTGGGTGCACCGCTGCCTGCAGATCGTGTTGAAAAAACCCTGGAACTGATTAAACAGGC AGATGCAAAATAACTCGGTACCAAAGACGAACAATAAGACGCTGAAAAGCGTCTTTTTTCGTTTTGGTCCCTTGTCC AACCAAATGATTCGTTACCAATTGACAGTTTCTATCGATCTATAGATAATGCTAGC
BM3R1 (RBS-B1) gate	gate	CTGAAGACTGTCGCCGGATGTGTATCCGACCTGACGATGGCCCAAAAGGGCCGAAACAGTCCTCTACAAATAATTTT GTTTAACTATGGACTATGTTTTAACTACTAGATGGAAAGCACCCCGACCAAACAGAAAGCAATTTTTAGCGCAAGCC TGCTGCTGTTTGCAGAACGTGGTTTTGATGCAACCACCATGCCGATGATTGCAGAAAATGCAAAAGTTGGTGCAGGC ACCATTTATCGCTATTTCAAAAACAAAGAAAGCCTGGTGAACGAACTGTTTCAGCAGCATGTTAATGAATTTCTGCA GTGTATTGAAAGCGGTCTGGCAAATGAACGTGATGGTTATCGTGATGGCTTTCATCACATTTTTGAAGGTATGGTGA CCTTTACCAAAAATCATCCGCGTGCACTGGGTTTTATCAAAACCCATAGCCAGGGCACCTTTCTGACCGAAGAAAGC CGTCTGGCATATCAGAAACTGGTTGAATTTGTGTGCACCTTTTTTCGTGAAGGTCAGAAACAGGGTGTGATTCGTAA TCTGCCGGAAAATGCACTGATTGCAATTCTGTTTGGCAGCTTTATGGAAGTGTATGAAATGATCGAGAACGATTATC TGAGCCTGACCGATGAACTGCTGACCGGTGTTGAAGAAAGCCTGTGGGCAGCACTGAGCCGTCAGAGCTAACTCGGT

BM3R1 (RBS-B2) gate	gate	ACCAAATTCCAGAAAAGAGACGCTTTCGAGCGTCTTTTTTCGTTTTGGTCCAATCCGCGTGATAGGTCTGATTCGTT ACCAATTGACGGAATGAACGTTCATTCCGATAATGCTAGC CTGAAGACTGTCGCCGGATGTGTATCCGACCTGACGATGGCCCAAAAGGGCCGAAACAGTCCTCTACAAATAATTTT GTTTAACTATGGACTATGTTTTTCAAAGACGAAAAACTACTAGATGGAAAGCACCCCGACCAAACAGAAAGCAATTT TTAGCGCAAGCCTGCTGCTGTTTGCAGAACGTGGTTTTGATGCAACCACCATGCCGATGATTGCAGAAAATGCAAAA GTTGGTGCAGGCACCATTTATCGCTATTTCAAAAACAAAGAAAGCCTGGTGAACGAACTGTTTCAGCAGCATGTTAA TGAATTTCTGCAGTGTATTGAAAGCGGTCTGGCAAATGAACGTGATGGTTATCGTGATGGCTTTCATCACATTTTTG AAGGTATGGTGACCTTTACCAAAAATCATCCGCGTGCACTGGGTTTTATCAAAACCCATAGCCAGGGCACCTTTCTG ACCGAAGAAAGCCGTCTGGCATATCAGAAACTGGTTGAATTTGTGTGCACCTTTTTTCGTGAAGGTCAGAAACAGGG TGTGATTCGTAATCTGCCGGAAAATGCACTGATTGCAATTCTGTTTGGCAGCTTTATGGAAGTGTATGAAATGATCG AGAACGATTATCTGAGCCTGACCGATGAACTGCTGACCGGTGTTGAAGAAAGCCTGTGGGCAGCACTGAGCCGTCAG AGCTAACTCGGTACCAAATTCCAGAAAAGAGACGCTTTCGAGCGTCTTTTTTCGTTTTGGTCCAATCCGCGTGATAG GTCTGATTCGTTACCAATTGACGGAATGAACGTTCATTCCGATAATGCTAGC
BM3R1 (RBS-B3) gate	gate	CTGAAGACTGTCGCCGGATGTGTATCCGACCTGACGATGGCCCAAAAGGGCCGAAACAGTCCTCTACAAATAATTTT GTTTAACCAAACGAGGCCGGGAGGATGGAAAGCACCCCGACCAAACAGAAAGCAATTTTTAGCGCAAGCCTGCTGCT GTTTGCAGAACGTGGTTTTGATGCAACCACCATGCCGATGATTGCAGAAAATGCAAAAGTTGGTGCAGGCACCATTT ATCGCTATTTCAAAAACAAAGAAAGCCTGGTGAACGAACTGTTTCAGCAGCATGTTAATGAATTTCTGCAGTGTATT GAAAGCGGTCTGGCAAATGAACGTGATGGTTATCGTGATGGCTTTCATCACATTTTTGAAGGTATGGTGACCTTTAC CAAAAATCATCCGCGTGCACTGGGTTTTATCAAAACCCATAGCCAGGGCACCTTTCTGACCGAAGAAAGCCGTCTGG CATATCAGAAACTGGTTGAATTTGTGTGCACCTTTTTTCGTGAAGGTCAGAAACAGGGTGTGATTCGTAATCTGCCG GAAAATGCACTGATTGCAATTCTGTTTGGCAGCTTTATGGAAGTGTATGAAATGATCGAGAACGATTATCTGAGCCT GACCGATGAACTGCTGACCGGTGTTGAAGAAAGCCTGTGGGCAGCACTGAGCCGTCAGAGCTAACTCGGTACCAAAT TCCAGAAAAGAGACGCTTTCGAGCGTCTTTTTTCGTTTTGGTCCAATCCGCGTGATAGGTCTGATTCGTTACCAATT GACGGAATGAACGTTCATTCCGATAATGCTAGC
BetI (RBS-E1) gate	gate	CTGAAGAAGTCAATTAATGTGCTTTTAATTCTGATGAGTCGGTGACGACGAAACTTCCTCTACAAATAATTTTGTTT AACCCCCCGAGGAGTAGCACATGCCGAAACTGGGTATGCAGAGCATTCGTCGTCGTCAGCTGATTGATGCAACCCTG GAAGCAATTAATGAAGTTGGTATGCATGATGCAACCATTGCACAGATTGCACGTCGTGCCGGTGTTAGCACCGGTAT TATTAGCCATTATTTCCGCGATAAAAACGGTCTGCTGGAAGCAACCATGCGTGATATTACCAGCCAGCTGCGTGATG CAGTTCTGAATCGTCTGCATGCACTGCCGCAGGGTAGCGCAGAACAGCGTCTGCAGGCAATTGTTGGTGGTAATTTT GATGAAACCCAGGTTAGCAGCGCAGCAATGAAAGCATGGCTGGCATTTTGGGCAAGCAGCATGCATCAGCCGATGCT GTATCGTCTGCAGCAGGTTAGCAGTCGTCGTCTGCTGAGCAATCTGGTTAGCGAATTTCGTCGTGAACTGCCTCGTG AACAGGCACAAGAGGCAGGTTATGGTCTGGCAGCACTGATTGATGGTCTGTGGCTGCGTGCAGCACTGAGCGGTAAA CCGCTGGATAAAACCCGTGCAAATAGCCTGACCCGTCATTTTATCACCCAGCATCTGCCGACCGATTAACCAATTAT TGAACACCCTTCGGGGTGTTTTTTTGTTTCTGGTCTACCAGCGCGGGTGAGAGGGATTCGTTACCAATTGACAATTG ATTGGACGTTCAATATAATGCTAGC
HlyIIR (RBS-H1) gate	gate	CTGAAGTAGTCACCGGCTGTGCTTGCCGGTCTGATGAGCCTGTGAAGGCGAAACTACCTCTACAAATAATTTTGTTT AAACCCCCGAGATGAAATACATCCTGTTTGAGGTGTGCGAAATGGGTAAAAGCCGTGAACAGACCATGGAAAATATT CTGAAAGCAGCCAAAAAGAAATTCGGCGAACGTGGTTATGAAGGCACCAGCATTCAAGAAATTACCAAAGAAGCCAA AGTTAACGTTGCAATGGCCAGCTATTACTTTAATGGCAAAGAGAACCTGTACTACGAGGTGTTCAAAAAATACGGTC TGGCAAATGAACTGCCGAACTTTCTGGAAAAAAACCAGTTTAATCCGATTAATGCCCTGCGTGAATATCTGACCGTT TTTACCACCCACATTAAAGAAAATCCGGAAATTGGCACCCTGGCCTATGAAGAAATTATCAAAGAAAGCGCACGCCT GGAAAAAATCAAACCGTATTTTATCGGCAGCTTCGAACAGCTGAAAGAAATTCTGCAAGAGGGTGAAAAACAGGGTG TGTTTCACTTTTTTAGCATCAACCATACCATCCATTGGATTACCAGCATTGTTCTGTTTCCGAAATTCAAAAAATTC ATCGATAGCCTGGGTCCGAATGAAACCAATGATACCAATCATGAATGGATGCCGGAAGATCTGGTTAGCCGTATTAT TAGCGCACTGACCGATAAACCGAACATTTAAAACGCATGAGAAAGCCCCCGGAAGATCACCTTCCGGGGGCTTTTTT ATTGCGCACCAGGAATCTGAACGATTCGTTACCAATTGACATATTTAAAATTCTTGTTTAAAATGCTAGC
IcaRA (RBS-I1) gate	gate	CTGAAGCCCCATAGGGTGGTGTGTACCACCCCTGATGAGTCCAAAAGGACGAAATGGGGCCTCTACAAATAATTTTG TTTAAATTGCTATGGACTATGTTTCAAAGTGAGAATACTAGGTGAAAGACAAAATTATCGATAACGCCATCACCCTG TTTAGCGAAAAAGGTTATGACGGCACCACCCTGGATGATATTGCAAAAAGCGTGAACATCAAAAAAGCCAGCCTGTA TTATCACTTTGATAGCAAAAAAAGCATCTACGAGCAGAGCGTTAAATGCTGTTTCGATTATCTGAACAACATCATCA TGATGAACCAGAACAAAAGCAACTATAGCATCGATGCCCTGTATCAGTTTCTGTTTGAGTTCATCTTCGATATCGAG GAACGCTATATTCGTATGTATGTTCAGCTGAGCAACACACCGGAAGAATTTTCAGGTAACATTTATGGCCAGATCCA GGATCTGAATCAGAGCCTGAGCAAAGAAATCGCCAAATTCTATGACGAAAGCAAAATCAAAATGACCAAAGAGGACT TCCAGAATCTGATTCTGCTGTTTCTGGAAAGCTGGTATCTGAAAGCCAGCTTTAGCCAGAAATTTGGTGCAGTTGAA GAAAGCAAAAGCCAGTTTAAAGATGAGGTTTATAGCCTGCTGAACATCTTTCTGAAGAAATAAACAATTTTCGAAAA AACCCGCTTCGGCGGGTTTTTTTATAGCTAAAAGTCAACTCATAAGATTCTGATTCGTTACCAATTGACAATTCACC TACCTTTCGTTAGGTTAGGTTGT
LitR (RBS-L1) gate	gate	CTGAAGTCATAAGTCTGGGCTAAGCCCACTGATGAGTCGCTGAAATGCGACGAAACTTATGACCTCTACAAATAATT TTGTTTAAGTCCTATGGACTTTTTCATACAGGAGAACCCTCGATGGATACCATTCAGAAACGTCCGCGTACCCGTCT GAGTCCGGAAAAACGTAAAGAACAGCTGCTGGATATTGCCATTGAAGTTTTTAGCCAGCGTGGTATTGGTCGTGGTG GTCATGCAGATATTGCAGAAATTGCACAGGTTAGCGTTGCAACCGTGTTTAACTATTTTCCGACCCGTGAAGATCTG GTTGATGATGTTCTGAACAAAGTGGAAAACGAGTTTCACCAGTTCATCAATAACAGCATTAGCCTGGATCTGGATGT TCGTAGCAATCTGAATACCCTGCTGCTGAACATTATTGATAGCGTTCAGACCGGCAACAAATGGATTAAAGTTTGGT TTGAATGGTCAACCAGCACCCGTGATGAAGTTTGGCCTCTGTTTCTGAGCACCCATAGCAATACCAATCAGGTGATC AAAACCATGTTTGAAGAGGGTATTGAACGCAATGAAGTGTGCAATGATCATACACCGGAAAATCTGACCAAAATGCT GCATGGTATTTGCTATAGCGTGTTTATTCAGGCCAATCGTAATAGCAGCAGCGAAGAAATGGAAGAAACCGCAAATT GCTTTCTGAATATGCTGTGCATCTACAAATAACTCGGTACCAAATTCCAGAAAAGACACCCGAAAGGGTGTTTTTTC GTTTTGGTCCCGAGCGTAGAGCTTAGATTCGTTACCAATTGACAAATTTATAAATTGTCAGTATAATGCTAGC
LmrA (RBS-N1) gate	gate	CTGAAGGAGTCAATTAATGTGCTTTTAATTCTGATGAGACGGTGACGTCGAAACTCCCTCTACAAATAATTTTGTTT AATACGCTATGGACTATGTTTTCTGCTATGGACTATGTTTTCACACACGAGATGCCTCGATGAGCTATGGTGATAGC CGTGAAAAAATTCTGAGCGCAGCAACCCGTCTGTTCAGCTGCAGGGTTATTATGGCACCGGTCTGAATCAGATTAT CAAAGAAAGCGGTGCACCGAAAGGTAGCCTGTATTATCATTTTCCGGGTGGTAAAGAACAGCTGGCAATTGAAGCAG TGAACGAAATGAAAGAATATATCCGCCAGAAAATCGCCGATTGTATGGAAGCATGTACCGATCCGGCAGAAGGTATT CAGGCATTTCTGAAAGAACTGAGCTGTCAGTTTAGCTGTACCGAAGATATTGAAGGTCTGCCGGTTGGTCTGCTGGC AGCAGAAACCAGCCTGAAAAGCGAACCGCTGCGTGAAGCATGTCATGAAGCATATAAAGAATGGGCCAGCGTGTATG AAGAAAAACTGCGTCAGACCGGTTGTAGCGAAAGCCGTGCAAAAGAAGCAAGCACCGTTGTTAATGCAATGATTGAA GGTGGTATTCTGCTGAGCCTGACCGCAAAAAATAGCACACCGCTGCTGCATATTAGCAGCTGTATTCCGGATCTGCT GAAACGTTAATAAGGTTGAAAAATAAAAACGGCGCTAAAAAGCGCCGTTTTTTTTGACGGTGGTACGCTCATTCACT AGGTCTGATTCGTTACCAATTGACAACTGGTGGTCGAATCAAGATAATAGACCAGTCACTATATTT
PhlF (RBS-P1)	gate	CTGAAGCGGTCAACGCATGTGCTTTGCGTTCTGATGAGACAGTGATGTCGAAACCGCCTCTACAAATAATTTTGTTT AACTATGGACTATGTTTGAAAGGGAGAAATACTAGATGGCACGTACCCCGAGCCGTAGCAGCATTGGTAGCCTGCGT

gate		AGTCCGCATACCCATAAAGCAATTCTGACCAGCACCATTGAAATCCTGAAAGAATGTGGTTATAGCGGTCTGAGCAT TGAAAGCGTTGCACGTCGTGCCGGTGCAAGCAAACCGACCATTTATCGTTGGTGGACCAATAAAGCAGCACTGATTG CCGAAGTGTATGAAAATGAAAGCGAACAGGTGCGTAAATTTCCGGATCTGGGTAGCTTTAAAGCCGATCTGGATTTT CTGCTGCGTAATCTGTGGAAAGTTTGGCGTGAAACCATTTGTGGTGAAGCATTTCGTTGTGTTATTGCAGAAGCACA GCTGGACCCTGCAACCCTGACCCAGCTGAAAGATCAGTTTATGGAACGTCGTCGTGAGATGCCGAAAAAACTGGTTG AAAATGCCATTAGCAATGGTGAACTGCCGAAAGATACCAATCGTGAACTGCTGCTGGATATGATTTTTGGTTTTTGT TGGTATCGCCTGCTGACCGAACAGCTGACCGTTGAACAGGATATTGAAGAATTTACCTTCCTGCTGATTAATGGTGT TTGTCCGGGTACACAGCGTTAAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAA GGCGACGTACGGTGGAATCTGATTCGTTACCAATTGACATGATACGAAACGTACCGTATCGTTAAGGT
PhlF (RBS-P2) gate	gate	CTGAAGCGGTCAACGCATGTGCTTTGCGTTCTGATGAGACAGTGATGTCGAAACCGCCTCTACAAATAATTTTGTTT AAGGAGCTATGGACTATGTTTGAAAGGCTGAAATACTAGATGGCACGTACCCCGAGCCGTAGCAGCATTGGTAGCCT GCGTAGTCCGCATACCCATAAAGCAATTCTGACCAGCACCATTGAAATCCTGAAAGAATGTGGTTATAGCGGTCTGA GCATTGAAAGCGTTGCACGTCGTGCCGGTGCAAGCAAACCGACCATTTATCGTTGGTGGACCAATAAAGCAGCACTG ATTGCCGAAGTGTATGAAAATGAAAGCGAACAGGTGCGTAAATTTCCGGATCTGGGTAGCTTTAAAGCCGATCTGGA TTTTCTGCTGCGTAATCTGTGGAAAGTTTGGCGTGAAACCATTTGTGGTGAAGCATTTCGTTGTGTTATTGCAGAAG CACAGCTGGACCCTGCAACCCTGACCCAGCTGAAAGATCAGTTTATGGAACGTCGTCGTGAGATGCCGAAAAAACTG GTTGAAAATGCCATTAGCAATGGTGAACTGCCGAAAGATACCAATCGTGAACTGCTGCTGGATATGATTTTTGGTTT TTGTTGGTATCGCCTGCTGACCGAACAGCTGACCGTTGAACAGGATATTGAAGAATTTACCTTCCTGCTGATTAATG GTGTTTGTCCGGGTACACAGCGTTAAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGGCTTTTTTTTTCGAC CAAAGGCGACGTACGGTGGAATCTGATTCGTTACCAATTGACATGATACGAAACGTACCGTATCGTTAAGGT
PhlF (RBS-P3) gate	gate	CTGAAGCGGTCAACGCATGTGCTTTGCGTTCTGATGAGACAGTGATGTCGAAACCGCCTCTACAAATAATTTTGTTT AACTTTACGAGGGCGATCCTATGGCACGTACCCCGAGCCGTAGCAGCATTGGTAGCCTGCGTAGTCCGCATACCCAT AAAGCAATTCTGACCAGCACCATTGAAATCCTGAAAGAATGTGGTTATAGCGGTCTGAGCATTGAAAGCGTTGCACG TCGTGCCGGTGCAAGCAAACCGACCATTTATCGTTGGTGGACCAATAAAGCAGCACTGATTGCCGAAGTGTATGAAA ATGAAAGCGAACAGGTGCGTAAATTTCCGGATCTGGGTAGCTTTAAAGCCGATCTGGATTTTCTGCTGCGTAATCTG TGGAAAGTTTGGCGTGAAACCATTTGTGGTGAAGCATTTCGTTGTGTTATTGCAGAAGCACAGCTGGACCCTGCAAC CCTGACCCAGCTGAAAGATCAGTTTATGGAACGTCGTCGTGAGATGCCGAAAAAACTGGTTGAAAATGCCATTAGCA ATGGTGAACTGCCGAAAGATACCAATCGTGAACTGCTGCTGGATATGATTTTTGGTTTTTGTTGGTATCGCCTGCTG ACCGAACAGCTGACCGTTGAACAGGATATTGAAGAATTTACCTTCCTGCTGATTAATGGTGTTTGTCCGGGTACACA GCGTTAAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAAGGCGACGTACGGTGG AATCTGATTCGTTACCAATTGACATGATACGAAACGTACCGTATCGTTAAGGT
PsrA (RBS-R1) gate	gate	CTGAAGCGCTGTCTGTACTTGTATCAGTACACTGACGAGTCCCTAAAGGACGAAACACCGCCTCTACAAATAATTTT GTTTAATTTAATTCGCGGAAGCGCAGAGATAAGGGGTATCATGGCACAGAGCGAAACCGTTGAACGTATTCTGGATG CAGCAGAACAGCTGTTTGCAGAACGTGGTTTTGCAGAAACCAGCCTGCGTCTGATTACCAGCAAAGCCGGTGTTAAT CTGGCAGCAGTGAATTATCATTTTGGCAGCAAAAAAGCACTGATTCAGGCAGTTTTTAGCCGTTTTCTGGGTCCGTT TTGTGCAAGCCTGGAACGTGAACTGGAACGTCGTCAGGCACGTCCGGAACAGAAACCGAGCCTGGAAGAACTGCTGG AAATGCTGGTTGAACAGGCACTGGCAGTTCAGCCTCGTAGCAATAATGATCTGAGCATTTTTATGTGTCTGCTGGGT CTGGCATTTAGCCAGAGCCAGGGTCATCTGCGTCGTTATCTGGAAGATATGTATGGTAAAGTGTTCCGTCGTTATAT GCTGCTGGTTAATGAAGCAGCACCGCGTGTTCCGCCTCTGGAACTGTTTTGGCGTGTTCATTTTATGCTGGGTGCAG CAGCATTTAGCATGAGCGGTATTAAAGCACTGCGTGCAATTGCAGAAACCGATTTTGGTATTAACACCAGCATTGAA CAGGTTATGCGTCTGATGGTTCCGTTTCTGGCAGCAGGTATGCGTGCAGATAGCGGTGTTACCGATGAAGCAATGGC AGCAGCACAGCTGCGTCCGCGTAGCAAAACCAGCACCAGCGCAACCACCGCAAAAGCATAATCCGGCAATTAAAAAA GCGGCTAACCACGCCGCTTTTTTTACGTCTGCATGATCGAACGCTTCAAGGAACAAACGTTTGATTGACAGCTAGCT CAGTCCTAGGTAGAGTGCTAGC
QacR (RBS-Q1) gate	gate	CTGAAGTCGTCAAGTGCTGTGCTTGCACTTCTGATGAGGCAGTGATGCCGAAACGACCTCTACAAATAATTTTGTTT AAGTAAGCCATGCCATTGGCTTTTGATAGAGGATAACTACTAGATGAACCTGAAAGATAAAATTCTGGGCGTTGCCA AAGAACTGTTTATCAAAAATGGCTATAACGCAACCACCACCGGTGAAATTGTTAAACTGAGCGAAAGCAGCAAAGGC AATCTGTATTATCACTTTAAAACCAAAGAGAACCTGTTTCTGGAAATCCTGAACATCGAAGAAAGCAAATGGCAAGA GCAGTGGAAAAAAGAACAAATCAAATGCAAAACCAACCGCGAGAAATTCTATCTGTATAATGAACTGAGCCTGACCA CCGAATATTACTATCCGCTGCAGAATGCCATCATCGAGTTTTATACCGAGTACTATAAAACCAACAGCATCAACGAG AAAATGAACAAACTGGAAAACAAATACATCGATGCCTACCACGTGATCTTTAAAGAAGGTAATCTGAACGGCGAATG GTGCATTAATGATGTTAATGCCGTGAGCAAAATTGCAGCAAATGCCGTTAATGGCATTGTTACCTTTACCCATGAGC AGAATATCAACGAACGCATTAAACTGATGAACAAATTCAGCCAGATCTTTCTGAATGGCCTGAGCAAATAAGTCAGT TTCACCTGTTTTACGTAAAAACCCGCTTCGGCGGGTTTTTACTTTTGGGGTATGGAAGCTATACGTTACCAATTGAC AGCTAGCTCAGTCCTACTTTAGTATATAGACCGTGCGATCGGTCTATA
QacR (RBS-Q2) gate	gate	CTGAAGTCGTCAAGTGCTGTGCTTGCACTTCTGATGAGGCAGTGATGCCGAAACGACCTCTACAAATAATTTTGTTT AAGCCATGCCATTGGCTTTTGATAGAGGACAACTACTAGATGAACCTGAAAGATAAAATTCTGGGCGTTGCCAAAGA ACTGTTTATCAAAAATGGCTATAACGCAACCACCACCGGTGAAATTGTTAAACTGAGCGAAAGCAGCAAAGGCAATC TGTATTATCACTTTAAAACCAAAGAGAACCTGTTTCTGGAAATCCTGAACATCGAAGAAAGCAAATGGCAAGAGCAG TGGAAAAAAGAACAAATCAAATGCAAAACCAACCGCGAGAAATTCTATCTGTATAATGAACTGAGCCTGACCACCGA ATATTACTATCCGCTGCAGAATGCCATCATCGAGTTTTATACCGAGTACTATAAAACCAACAGCATCAACGAGAAAA TGAACAAACTGGAAAACAAATACATCGATGCCTACCACGTGATCTTTAAAGAAGGTAATCTGAACGGCGAATGGTGC ATTAATGATGTTAATGCCGTGAGCAAAATTGCAGCAAATGCCGTTAATGGCATTGTTACCTTTACCCATGAGCAGAA TATCAACGAACGCATTAAACTGATGAACAAATTCAGCCAGATCTTTCTGAATGGCCTGAGCAAATAAGTCAGTTTCA CCTGTTTTACGTAAAAACCCGCTTCGGCGGGTTTTTACTTTTGGGGTATGGAAGCTATACGTTACCAATTGACAGCT AGCTCAGTCCTACTTTAGTATATAGACCGTGCGATCGGTCTATA
SrpR (RBS-S1) gate	gate	CTGAAGCGCTCAACGGGTGTGCTTCCCGTTCTGATGAGTCCGTGAGGACGAAAGCGCCTCTACAAATAATTTTGTTT AAGAGTCTATGGACTATGTTTTCACAGAGGAGGTACCAGGATGGCACGTAAAACCGCAGCAGAAGCAGAAGAAACCC GTCAGCGTATTATTGATGCAGCACTGGAAGTTTTTGTTGCACAGGGTGTTAGTGATGCAACCCTGGATCAGATTGCA CGTAAAGCCGGTGTTACCCGTGGTGCAGTTTATTGGCATTTTAATGGTAAACTGGAAGTTCTGCAGGCAGTTCTGGC AAGCCGTCAGCATCCGCTGGAACTGGATTTTACACCGGATCTGGGTATTGAACGTAGCTGGGAAGCAGTTGTTGTTG CAATGCTGGATGCAGTTCATAGTCCGCAGAGCAAACAGTTTAGCGAAATTCTGATTTATCAGGGTCTGGATGAAAGC GGTCTGATTCATAATCGTATGGTTCAGGCAAGCGATCGTTTTCTGCAGTATATTCATCAGGTTCTGCGTCATGCAGT TACCCAGGGTGAACTGCCGATTAATCTGGATCTGCAGACCAGCATTGGTGTTTTTAAAGGTCTGATTACCGGTCTGC TGTATGAAGGTCTGCGTAGCAAAGATCAGCAGGCACAGATTATCAAAGTTGCACTGGGTAGCTTTTGGGCACTGCTG CGTGAACCGCCTCGTTTTCTGCTGTGTGAAGAAGCACAGATTAAACAGGTGAAATCCTTCGAATAATTCAGCCAAAA AACTTAAGACCGCCGGTCTTGTCCACTACCTTGCAGTAATGCGGTGGACAGGATCGGCGGTTTTCTTTTCTCTTCTC AATCTATGATTGGTCCAGATTCGTTACCAATTGACAGCTAGCTCAGTCCTAGGTATATACATACATGCTTGTTTGTT TGTAAAC
SrpR (RBS-S2)	gate	CTGAAGCGCTCAACGGGTGTGCTTCCCGTTCTGATGAGTCCGTGAGGACGAAAGCGCCTCTACAAATAATTTTGTTT AAGAGTCTATGGACTATGTTTTCACATATGAGATACCAGGATGGCACGTAAAACCGCAGCAGAAGCAGAAGAAACCC

gate		GTCAGCGTATTATTGATGCAGCACTGGAAGTTTTTGTTGCACAGGGTGTTAGTGATGCAACCCTGGATCAGATTGCA CGTAAAGCCGGTGTTACCCGTGGTGCAGTTTATTGGCATTTTAATGGTAAACTGGAAGTTCTGCAGGCAGTTCTGGC AAGCCGTCAGCATCCGCTGGAACTGGATTTTACACCGGATCTGGGTATTGAACGTAGCTGGGAAGCAGTTGTTGTTG CAATGCTGGATGCAGTTCATAGTCCGCAGAGCAAACAGTTTAGCGAAATTCTGATTTATCAGGGTCTGGATGAAAGC GGTCTGATTCATAATCGTATGGTTCAGGCAAGCGATCGTTTTCTGCAGTATATTCATCAGGTTCTGCGTCATGCAGT TACCCAGGGTGAACTGCCGATTAATCTGGATCTGCAGACCAGCATTGGTGTTTTTAAAGGTCTGATTACCGGTCTGC TGTATGAAGGTCTGCGTAGCAAAGATCAGCAGGCACAGATTATCAAAGTTGCACTGGGTAGCTTTTGGGCACTGCTG CGTGAACCGCCTCGTTTTCTGCTGTGTGAAGAAGCACAGATTAAACAGGTGAAATCCTTCGAATAATTCAGCCAAAA AACTTAAGACCGCCGGTCTTGTCCACTACCTTGCAGTAATGCGGTGGACAGGATCGGCGGTTTTCTTTTCTCTTCTC AATCTATGATTGGTCCAGATTCGTTACCAATTGACAGCTAGCTCAGTCCTAGGTATATACATACATGCTTGTTTGTT TGTAAAC
SrpR (RBS-S3) gate	gate	CTGAAGCGCTCAACGGGTGTGCTTCCCGTTCTGATGAGTCCGTGAGGACGAAAGCGCCTCTACAAATAATTTTGTTT AAGAGTCTATGGACTATGTTTTCACAAAGGAAGTACCAGGATGGCACGTAAAACCGCAGCAGAAGCAGAAGAAACCC GTCAGCGTATTATTGATGCAGCACTGGAAGTTTTTGTTGCACAGGGTGTTAGTGATGCAACCCTGGATCAGATTGCA CGTAAAGCCGGTGTTACCCGTGGTGCAGTTTATTGGCATTTTAATGGTAAACTGGAAGTTCTGCAGGCAGTTCTGGC AAGCCGTCAGCATCCGCTGGAACTGGATTTTACACCGGATCTGGGTATTGAACGTAGCTGGGAAGCAGTTGTTGTTG CAATGCTGGATGCAGTTCATAGTCCGCAGAGCAAACAGTTTAGCGAAATTCTGATTTATCAGGGTCTGGATGAAAGC GGTCTGATTCATAATCGTATGGTTCAGGCAAGCGATCGTTTTCTGCAGTATATTCATCAGGTTCTGCGTCATGCAGT TACCCAGGGTGAACTGCCGATTAATCTGGATCTGCAGACCAGCATTGGTGTTTTTAAAGGTCTGATTACCGGTCTGC TGTATGAAGGTCTGCGTAGCAAAGATCAGCAGGCACAGATTATCAAAGTTGCACTGGGTAGCTTTTGGGCACTGCTG CGTGAACCGCCTCGTTTTCTGCTGTGTGAAGAAGCACAGATTAAACAGGTGAAATCCTTCGAATAATTCAGCCAAAA AACTTAAGACCGCCGGTCTTGTCCACTACCTTGCAGTAATGCGGTGGACAGGATCGGCGGTTTTCTTTTCTCTTCTC AATCTATGATTGGTCCAGATTCGTTACCAATTGACAGCTAGCTCAGTCCTAGGTATATACATACATGCTTGTTTGTT TGTAAAC
SrpR (RBS-S4) gate	gate	CTGAAGCGCTCAACGGGTGTGCTTCCCGTTCTGATGAGTCCGTGAGGACGAAAGCGCCTCTACAAATAATTTTGTTT AACTATGGACTATGTTTTCACACAGGAAATACCAGGATGGCACGTAAAACCGCAGCAGAAGCAGAAGAAACCCGTCA GCGTATTATTGATGCAGCACTGGAAGTTTTTGTTGCACAGGGTGTTAGTGATGCAACCCTGGATCAGATTGCACGTA AAGCCGGTGTTACCCGTGGTGCAGTTTATTGGCATTTTAATGGTAAACTGGAAGTTCTGCAGGCAGTTCTGGCAAGC CGTCAGCATCCGCTGGAACTGGATTTTACACCGGATCTGGGTATTGAACGTAGCTGGGAAGCAGTTGTTGTTGCAAT GCTGGATGCAGTTCATAGTCCGCAGAGCAAACAGTTTAGCGAAATTCTGATTTATCAGGGTCTGGATGAAAGCGGTC TGATTCATAATCGTATGGTTCAGGCAAGCGATCGTTTTCTGCAGTATATTCATCAGGTTCTGCGTCATGCAGTTACC CAGGGTGAACTGCCGATTAATCTGGATCTGCAGACCAGCATTGGTGTTTTTAAAGGTCTGATTACCGGTCTGCTGTA TGAAGGTCTGCGTAGCAAAGATCAGCAGGCACAGATTATCAAAGTTGCACTGGGTAGCTTTTGGGCACTGCTGCGTG AACCGCCTCGTTTTCTGCTGTGTGAAGAAGCACAGATTAAACAGGTGAAATCCTTCGAATAATTCAGCCAAAAAACT TAAGACCGCCGGTCTTGTCCACTACCTTGCAGTAATGCGGTGGACAGGATCGGCGGTTTTCTTTTCTCTTCTCAATC TATGATTGGTCCAGATTCGTTACCAATTGACAGCTAGCTCAGTCCTAGGTATATACATACATGCTTGTTTGTTTGTA AAC
AraC* transcription unit	sensor module	TCACCATATATCAAGTTTACGGCTAGCTCAGTCCTAGGTACTATGCTAGCTACTAGAGAAAGAGGAGAAATACTAGA TGGCTGAAGCGCAAAATGATCCCCTGCTGCCGGGATACTCGTTTAATGCCCATCTGGTGGCGGGTTTAACGCCGATT GAGGCCAACGGTTATCTCGATTTTTTTATCGACCGACCGCTGGGAATGAAAGGTTATATTCTCAATCTCACCATTCG CGGTCAGGGGGTGGTGAAAAATCAGGGACGAGAATTTGTTTGCCGACCGGGTGATATTTTGCTGTTCCCGCCAGGAG AGATTCATCACTACGGTCGTCATCCGGAGGCTCGCGAATGGTATCACCAGTGGGTTTACTTTCGTCCGCGCGCCTAC TGGCATGAATGGCTTAACTGGCCGTCAATATTTGCCAATACGGGGTTCTTTCGCCCGGATGAAGCGCACCAGCCGCA TTTCAGCGACCTGTTTGGGCAAATCATTAACGCCGGGCAAGGGGAAGGGCGCTATTCGGAGCTGCTGGCGATAAATC TGCTTGAGCAATTGTTACTGCGGCGCATGGAAGCGATTAACGAGTCGCTCCATCCACCGATGGATAATCGGGTACGC GAGGCTTGTCAGTACATCAGCGATCACCTGGCAGACAGCAATTTTGATATCGCCAGCGTCGCACAGCATGTTTGCTT GTCGCCGTCGCGTCTGTCACATCTTTTCCGCCAGCAGTTAGGGATTAGCGTCTTAAGCTGGCGCGAGGACCAACGTA TCAGCCAGGCGAAGCTGCTTTTGAGCACCACCCGGATGCCTATCGCCACCGTCGGTCGCAATGTTGGTTTTGACGAT CAACTCTATTTCTCGCGGGTATTTAAAAAATGCACCGGGGCCAGCCCGAGCGAGTTCCGTGCCGGTTAATAACCAAT TATTGAAGGCCGCTAACGCGGCCTTTTTTTGTTTCTGGTCTCCC
LacI and TetR transcription unit	sensor module	GCGGCGCGCCATCGAATGGCGCAAAACCTTTCGCGGTATGGCATGATAGCGCCCGGAAGAGAGTCAATTCAGGGTGG TGAATATGAAACCAGTAACGTTATACGATGTCGCAGAGTATGCCGGTGTCTCTTATCAGACCGTTTCCCGCGTGGTG AACCAGGCCAGCCACGTTTCTGCGAAAACGCGGGAAAAAGTGGAAGCGGCGATGGCGGAGCTGAATTACATTCCCAA CCGCGTGGCACAACAACTGGCGGGCAAACAGTCGTTGCTGATTGGCGTTGCCACCTCCAGTCTGCCCCTGCACGCGC CGTCGCAAATTGTCGCGGCGATTAAATCTCGCGCCGATCAACTGGGTGCCAGCGTGGTGGTGTCGATGGTAGAACGA AGCGGCGTCGAAGCCTGTAAAGCGGCGGTGCACAATCTTCTCGCGCAACGCGTCAGTGGGCTGATCATTAACTATCC GCTGGATGACCAGGATGCCATTGCTGTGGAAGCTGCCTGCACTAATGTTCCGGCGTTATTTCTTGATGTCTCTGACC AGACACCCATCAACAGTATTATTTTCTCCCATGAGGACGGTACGCGACTGGGCGTGGAGCATCTGGTCGCATTGGGT CACCAGCAAATCGCGCTGTTAGCGGGCCCATTAAGTTCTGTCTCGGCGCGTCTGCGTCTGGCTGGCTGGCATAAATA TCTCACTCGCAATCAAATTCAGCCGATAGCGGAACGGGAAGGCGACTGGAGTGCCATGTCCGGTTTTCAACAAACCA TGCAAATGCTGAATGAGGGCATCGTTCCCACTGCGATGCTGGTTGCCAACGATCAGATGGCGCTGGGCGCAATGCGC GCCATTACCGAGTCCGGGCTGCGCGTTGGTGCGGATATCTCGGTAGTGGGATACGACGATACCGAAGATAGCTCATG TTATATCCCGCCGTTAACCACCATCAAACAGGATTTTCGCCTGCTGGGGCAAACCAGCGTGGACCGCTTGCTGCAAC TCTCTCAGGGCCAGGCGGTGAAGGGCAATCAGCTGTTGCCAGTCTCACTGGTGAAAAGAAAAACCACCCTGGCGCCC AATACGCAAACCGCCTCTCCCCGCGCGTTGGCCGATTCATTAATGCAGCTGGCACGACAGGTTTCCCGACTGGAAAG CGGGCAGTGATAATCCAGGAGGAAAAAAATGTCCAGATTAGATAAAAGTAAAGTGATTAACAGCGCATTAGAGCTGC TTAATGAGGTCGGAATCGAAGGTTTAACAACCCGTAAACTCGCCCAGAAGCTAGGTGTAGAGCAGCCTACATTGTAT TGGCATGTAAAAAATAAGCGGGCTTTGCTCGACGCCTTAGCCATTGAGATGTTAGATAGGCACCATACTCACTTTTG CCCTTTAGAAGGGGAAAGCTGGCAAGATTTTTTACGTAATAACGCTAAAAGTTTTAGATGTGCTTTACTAAGTCATC GCGATGGAGCAAAAGTACATTTAGGTACACGGCCTACAGAAAAACAGTATGAAACTCTCGAAAATCAATTAGCCTTT TTATGCCAACAAGGTTTTTCACTAGAGAATGCATTATATGCACTCAGCGCTGTGGGGCATTTTACTTTAGGTTGCGT ATTGGAAGATCAAGAGCATCAAGTCGCTAAAGAAGAAAGGGAAACACCTACTACTGATAGTATGCCGCCATTATTAC GACAAGCTATCGAATTATTTGATCACCAAGGTGCAGAGCCAGCCTTCTTATTCGGCCTTGAATTGATCATATGCGGA TTAGAAAAACAACTTAAATGTGAAAGTGGGTCCTAATAA

a. DNA sequence colors correspond to ribozyme insulators (blue), RBSs (green), protein coding sequences (red), terminators (black), output promoters (orange), and sensor transcription units (purple).

Table S9: Genetic part sequences

Part name	Type	DNA sequence[a]
BBa_J23101	promoter	tttacagctagctcagtcctaggtattatgctagc
BBa_J23105	promoter	tttacggctagctcagtcctaggtactatgctagc
P_{LacI}	promoter	gcggcgcgccatcgaatggcgcaaaacctttcgcggtatggcatgatagcgcccggaagagagtcaattcag ggtggtgaat
P_{Tac}	promoter(*44*)	aacgatcgttggctgtgttgacaattaatcatcggctcgtataatgtgtggaattgtgagcgctcacaatt
P_{Tet}	promoter(*13*)	tactccaccgttggctttttccctatcagtgatagagattgacatccctatcagtgatagagataatgagc ac
P_{BAD}	promoter(*45*)	actttcatactcccgccattcagagaagaaaccaattgtccatattgcatcagacattgccgtcactgcgt ctttactggctcttctcgctaaccaaaccggtaaccccgcttattaaaagcattctgtaacaaagcgggac caaagccatgacaaaaacgcgtaacaaaagtgtctataatcacggcagaaaagtccacattgattatttgca cggcgtcacactttgctatgccatagcatttttatccataagattagcggatcctacctgacgctttttatc gcaactctctactgtttctccatacccgtttttttgggctagc
P_{AmeR}	promoter(*13*)	tcgtcactagagggcgatagtgacaaacttgacaactcatcacttcctaggtataatgctagc
P_{AmtR}	promoter(*13*)	cttgtccaaccaaatgattcgttaccaattgacagtttctatcgatctatagataatgctagc
P_{BetI}	promoter(*13*)	agcgcgggtgagagggattcgttaccaattgacaattgattggacgttcaatataatgctagc
P_{BM3R1}	promoter(*13*)	aatccgcgtgtaggtctgattcgttaccaattgacggaatgaacgttcattccgataatgctagc
P_{HlyIIR}	promoter(*13*)	accaggaatctgaacgattcgttaccaattgacatatttaaaattcttgtttaaaatgctagc
P_{IcaRA}	promoter(*13*)	gtcaactcataagattctgattcgttaccaattgacaattcacctacctttcgttaggttaggttgt
P_{LitR}	promoter(*13*)	cgagcgtagagcttagattcgttaccaattgacaaatttataaattgtcagtataatgctagc
P_{LmrA}	promoter(*13*)	cgctcattcactaggtctgattcgttaccaattgacaactggtggtcgaatcaagataatagaccagtcact atattt
P_{PhlF}	promoter(*13*)	cgacgtacggtggaatctgattcgttaccaattgacatgatacgaaacgtaccgtatcgttaaggt
P_{PsrA}	promoter(*13*)	tgatcgaacgcttcaaggaacaaacgtttgattgacagctagctcagtcctaggtataatgctagc
P_{QacR}	promoter(*13*)	ggtatggaagctatacgttaccaattgacagctagctcagtcctactttagtatatagaccgtgcgatcggt ctata
P_{SrpR}	promoter(*13*)	tctatgattggtccagattcgttaccaattgacagctagctcagtcctaggtatatacatacatgcttgttt gtttgtaaac
RiboJ	insulator(*2*)	agctgtcaccggatgtgctttccggtctgatgagtccgtgaggacgaaacagcctctacaaataatttttgtt taa
RiboJ54	insulator	aggggtcagttgatgtgctttcaactctgatgagtcagtgatgacgaaacccctctacaaataatttttgtt taa
BydvJ	insulator	agggtgtctcaaggtgcgtaccttgactgatgagtccgaaaggacgaaacacccctctacaaataatttttgt ttaa
RiboJ57	insulator	agaagtcaattaatgtgcttttaattctgatgagtcggtgacgacgaaacttcctctacaaataatttttgtt taa
SarJ	insulator(*2*)	gactgtcgccggatgtgtatccgacctgacgatggcccaaaagggccgaaacagtcctctacaaataatttt gtttaa
RiboJ51	insulator	agtagtcaccggctgtgcttgccggtctgatgagcctgtgaaggcgaaactacctctacaaataatttttgtt taa
ElvJ	insulator	agccccatagggtggtgtgtaccacccctgatgagtccaaaaggacgaaatggggcctctacaaataatttt gtttaa
PlmJ	insulator(*2*)	agtcataagtctgggctaagcccactgatgagtcgctgaaatgcgacgaaacttatgacctctacaaataat tttgtttaa
RiboJ64	insulator	aggagtcaattaatgtgcttttaattctgatgagacggtgacgtcgaaactccctctacaaataatttttgtt taa
RiboJ53	insulator	agcggtcaacgcatgtgctttgcgttctgatgagacagtgatgtcgaaaccgcctctacaaataatttttgtt taa
ScmJ	insulator	agcgctgtctgtacttgtatcagtacactgacgagtccctaaaggacgaaacaccgcctctacaaataatttt tgtttaa
RiboJ60	insulator	ctgaagtcgtcaagtgctgtgcttgcacttctgatgaggcagtgatgccgaaacgacctctacaaataatttt tgtttaa
RiboJ10	insulator	agcgctcaacgggtgtgcttcccgttctgatgagtccgtgaggacgaaagcgcctctacaaataatttttgtt taa
yfp	gene(*30*)	atggtgagcaagggcgaggagctgttcaccggggtggtgcccatcctggtcgagctggacggcgacgtaaac ggccacaagttcagcgtgtccggcgagggcgagggcgatgccacctacggcaagctgaccctgaagttcatc tgcaccacaggcaagctgcccgtgccctggcccaccctcgtgaccaccttcggctacggcctgcaatgcttc gcccgctaccccgaccacatgaagctgcacgacttcttcaagtccgccatgcccgaaggctacgtccaggag cgcaccatcttcttcaaggacgacggcaactacaagacccgcgccgaggtgaagttcgagggcgacaccctg gtgaaccgcatcgagctgaagggcatcgacttcaaggaggacggcaacatcctggggcacaagctggagtac aactacaacagccacaacgtctatatcatggccgacaagcagaagaacggcatcaaggtgaacttcaagatc cgccacaacatcgaggacggcagcgtgcagctcgccgaccactaccagcagaacaccccaatcggcgacggc cccgtgctgctgcccgacaaccactaccttagctaccagtccgccctgagcaaagaccccaacgagaagcgc gatcacatggtcctgctggagttcgtgaccgccgccgggatcactctcggcatggacgagctgtacaagtaa
lacI	gene(*13*)	atgaaaccagtaacgttatacgatgtcgcagagtatgccggtgtctcttatcagaccgtttcccgcgtggtg aaccaggccagccacgtttctgcgaaaacgcgggaaaaagtggaagcggcgatggcggagctgaattacatt cccaaccgcgtggcacaacaactggcgggcaaacagtcgttgctgattggcgttgccacctccagtctggcc ctgcacgcgccgtcgcaaattgtcgcggcgattaaatctcgcgccgatcaactgggtgccagcgtggtggtg tcgatggtagaacgaagcggcgtcgaagcctgtaaagcggcggtgcacaatcttctcgcgcaacgcgtcagt gggctgatcattaactatccgctggatgaccaggatgccattgctgtggaagctgcctgcactaatgttccg gcgttatttcttgatgtctctgaccagacacccatcaacagtattattttctcccatgaggacggtacgcga ctgggcgtggagcatctggtcgcattgggtcaccagcaaatcgcgctgttagcgggcccattaagttctgtc tcggcgcgtctgcgtctggctggctggcataaatatctcactcgcaatcaaattcagccgatagcggaacgg gaaggcgactggagtgccatgtccggttttcaacaaaccatgcaaatgctgaatgagggcatcgttcccact gcgatgctggttgccaacgatcagatggcgctgggcgcaatgcgcgccattaccgagtccgggctgcgcgtt ggtgcggatatctcggtagtgggatacgacgataccgaagatagctcatgttatatcccgccgttaaccacc

tetR	gene(*13*)	atcaaacaggattttcgcctgctggggcaaaccagcgtggaccgcttgctgcaactctctcagggccaggcg gtgaagggcaatcagctgttgccagtctcactggtgaaaagaaaaaccaccctggcgcccaatacgcaaacc gcctctccccgcgcgttggccgattcattaatgcagctggcacgacaggtttcccgactggaaagcgggcag tgataa
*araC**	gene(*40*)	atgtccagattagataaaagtaaagtgattaacagcgcattagagctgcttaatgaggtcggaatcgaaggt ttaacaacccgtaaactcgcccagaagctaggtgtagagcagcctacattgtattggcatgtaaaaaataag cgggctttgctcgacgccttagccattgagatgttagataggcaccatactcacttttgccctttagaaggg gaaagctggcaagatttttttacgtaataacgctaaaagttttagatgtgctttactaagtcatcgcgatgga gcaaaagtacatttaggtacacggcctacagaaaaacagtatgaaactctcgaaaatcaattagccttttta tgccaacaaggtttttcactagagaatgcattatatgcactcagcgctgtggggcattttactttaggttgc gtattggaagatcaagagcatcaagtcgctaaagaagaaagggaaacacctactactgatagtatgccgcca ttattacgacaagctatcgaattatttgatcaccaaggtgcagagccagccttcttattcggccttgaattg atcatatgcggattagaaaaacaacttaaatgtgaaagtgggtcctaa
ameR	gene(*13*)	atggctgaagcgcaaaatgatcccctgctgccgggatactcgtttaatgcccatctggtggcgggtttaacg ccgattgaggccaacggttatctcgatttttttatcgaccgaccgctgggaatgaaaggttatattctcaat ctcaccattcgcggtcaggggtggtgaaaaatcagggacgagaatttgtttgccgaccgggtgatattttg ctgttcccgccaggagagattcatcactacggtcgtcatccggaggctcgcgaatggtatcaccagtgggtt tactttcgtccgcgcgcctactggcatgaatggcttaactggccgtcaatatttgccaatacggggttcttt cgcccggatgaagcgcaccagccgcatttcagcgacctgtttgggcaaatcattaacgccgggcaaggggaa gggcgctattcggagctgctggcgataaatctgcttgagcaattgttactgcggcgcatggaagcgattaac gagtcgctccatccaccgatggataatcgggtacgcgaggcttgtcagtacatcagcgatcacctggcagac agcaattttgatatcgccagcgtcgcacagcatgtttgcttgtcgccgtcgcgtctgtcacatcttttccgc cagcagttagggattagcgtcttaagctggcgcgaggaccaacgtatcagccaggcgaagctgcttttgagc accacccggatgcctatcgccaccgtcggtcgcaatgttggttttgacgatcaactctatttctcgcgggta tttaaaaaatgcaccggggccagcccgagcgagttccgtgccggttgtgaagaaaaagtgaatgatgtagcc gtcaagttgtcataa
amtR	gene(*13*)	atgaacaaaaccattgatcaggtgcgtaaaggtgatcgtaaaagcgatctgccggttcgtcgtcgtccgcgt cgtagtgccgaagaaacccgtcgtgatattctggcaaaagccgaagaactgtttcgtgaacgtggttttaat gcagttgccattgcagatattgcaagcgcactgaatatgagtccggcaaatgtgtttaaacattttagcagc aaaaacgcactggttgatgcaattggttttggtcagattggtgtttttgaacgtcagatttgtccgctggat aaaagccatgcaccgctggatcgtctgcgtcatctggcacgtaatctgatggaacagcatcatcaggatcat ttcaaacacatacgggtttttattcagatcctgatgaccgccaaacaggatatgaaatgtggcgattattac aaaagcgtgattgcaaaactgctggccgaaattattcgtgatggtgttgaagcaggtctgtatattgcaacc gatattccggttctggcagaaaccgttctgcatgcactgaccagcgttattcatccggttctgattgcacaa gaagatattggtaatctggcaacccgttgtgatcagctggttgatctgattgatgcaggtctgcgtaatccg ctggcaaaataa
betI	gene(*13*)	atggcaggcgcagttggtcgtccgcgtcgtagtgcaccgcgtcgtgcaggtaaaaatccgcgtgaagaaatt ctggatgcaagcgcagaactgtttacccgtcagggttttgcaaccaccagtacccatcagattgcagatgca gttggtattcgtcaggcaagcctgtattatcattttccgagcaaaaccgaaatctttctgaccctgctgaaa agcaccgttgaaccgagcaccgttctggcagaagatctgagcaccctggatgcaggtccggaaatgcgtctg tgggcaattgttgcaagcgaagttcgtctgctgctgagcaccaaatggaatgttggtcgtctgtatcagctg ccgattgttggtagcgaagaatttgcagaatatcatagccagcgtgaagcactgaccaatgtttttcgtgat ctggcaaccgaaattgttggtgatgatccgcgtgcagaactgccgtttcatattaccatgagcgttattgaa atgcgtcgcaatgatggtaaaattccgagtccgctgagcgcagatagcctgccggaaaccgcaattatgctg gcagatgcaagcctggcagttctgggtgcaccgctgcctgcagatcgtgttgaaaaaaccctggaactgatt aaacaggcagatgcaaaataa
bm3R1	gene(*13*)	atgccgaaactgggtatgcagagcattcgtcgtcgtcagctgattgatgcaaccctggaagcaattaatgaa gttggtatgcatgatgcaaccattgcacagattgcacgtcgtgccggtgttagcaccggtattattagccat tatttccgcgataaaaacggtctgctggaagcaaccatgcgtgatattaccagccagctgcgtgatgcagtt ctgaatcgtctgcatgcactgccgcagggtagcgcagaacagcgtctgcaggcaattgttggtggtaatttt gatgaaacccaggttagcagcgcagcaatgaaagcatggctggcattttgggcaagcagcatgcatcagccg atgctgtatcgtctgcagcaggttagcagtcgtcgtctgctgagcaatctggttagcgaatttcgtcgtgaa ctgcctcgtgaacaggcacaagaggcaggttatggtctggcagcactgattgatggtctgtggctgcgtgca gcactgagcggtaaaccgctggataaaacccgtgcaaatagcctgacccgtcattttatcacccagcatctg ccgaccgattaa
hlyIIR	gene(*13*)	atggaaagcaccccgaccaaacagaaagcaatttttagcgcaagcctgctgctgtttgcagaacgtggtttt gatgcaaccaccatgccgatgattgcagaaaatgcaaaagttggtgcaggcaccatttatcgctatttcaaa aacaaagaaagcctggtgaacgaactgtttcagcagcatgttaatgaatttctgcagtgtattgaaagcggt ctggcaaatgaacgtgatggttatcgtgatggctttcatcacatttttgaaggtatggtgacctttaccaaa aatcatccgcgtgcactgggttttatcaaaacccatagccagggcacctttctgaccgaagaaagccgtctg gcatatcagaaactggttgaatttgtgtgcaccttttttcgtgaaggtcagaaacagggtgtgattcgtaat ctgccggaaaatgcactgattgcaattctgtttggcagctttatggaagtgtatgaaatgatcgagaacgat tatctgagcctgaccgatgaactgctgaccggtgttgaagaaagcctgtgggcagcactgagccgtcagagc taa
icaRA	gene(*13*)	atgaaatacatcctgtttgaggtgtcgaaatgggtaaaagccgtgaacagaccatggaaaatattctgaaa gcagccaaaaagaaattcggcgaacgtggttatgaaggcaccagcattcaagaaattaccaaagaagccaaa gttaacgttgcaatggccagctattactttaatggcaaagagaacctgtactacgaggtgttcaaaaaatac ggtctggcaaatgaactgccgaactttctggaaaaaaaccagtttaatccgattaatgccctgcgtgaatat ctgaccgtttttaccacccacattaaagaaaatccggaaattggcaccctggcctatgaagaaattatcaaa gaaagcgcacgcctggaaaaaatcaaaccgtattttatcggcagcttcgaacagctgaaagaaattctgcaa gagggtgaaaaacagggtgtgtttcacttttttagcatcaaccataccatccattggattaccagcattgtt ctgtttccgaaattcaaaaaattcatcgatagcctgggtccgaatgaaaccaatgataccaatcatgaatgg atgccggaagatctggttagccgtattattagcgcactgaccgataaaccgaacatttaa
		gtgaaagacaaaattatcgataacgccatcaccctgtttagcgaaaaaggttatgacggcaccaccctggat gatattgcaaaaagcgtgaacatcaaaaaagccagcctgtattatcactttgatagcaaaaaaagcatctac gagcagagcgttaaatgctgtttcgattatctgaacaacatcatcatgatgaaccagaacaaaagcaactat agcatcgatgccctgtatcagtttctgtttgagttcatcttcgatatcgaggaacgctatattcgtatgtat gttcagctgagcaacacaccggaagaattttcaggtaacatttatggccagatccaggatctgaatcagagc ctgagcaaagaaatcgccaaattctatgacgaaagcaaaatcaaaatgaccaaagaggacttccagaatctg attctgctgtttctggaaagctggtatctgaaagccagctttagccagaaatttggtgcagttgaagaaagc aaaagccagtttaaagatgaggtttatagcctgctgaacatctttctgaagaaataa

litR	gene(*13*)	atggataccattcagaaacgtccgcgtacccgtctgagtccggaaaaacgtaaagaacagctgctggatatt gccattgaagtttttagccagcgtggtattggtcgtggtggtcatgcagatattgcagaaattgcacaggtt agcgttgcaaccgtgtttaactattttccgacccgtgaagatctggttgatgatgttctgaacaaagtggaa aacgagtttcaccagttcatcaataacagcattagcctggatctggatgttcgtagcaatctgaataccctg ctgctgaacattattgatagcgttcagaccggcaacaaatggattaaagtttggtttgaatggtcaaccagc acccgtgatgaagtttggcctctgtttctgagcacccatagcaataccaatcaggtgatcaaaaccatgttt gaagagggtattgaacgcaatgaagtgtgcaatgatcatacaccggaaaatctgaccaaaatgctgcatggt atttgctatagcgtgtttattcaggccaatcgtaatagcagcagcgaagaaatggaagaaaccgcaaattgc tttctgaatatgctgtgcatctacaaataa
lmrA	gene(*13*)	atgagctatggtgatagccgtgaaaaaattctgagcgcagcaacccgtctgtttcagctgcagggttattat ggcaccggtctgaatcagattatcaaagaaagcggtgcaccgaaaggtagcctgtattatcattttccgggt ggtaaagaacagctggcaattgaagcagtgaacgaaatgaaagaatatatccgccagaaaatcgccgattgt atggaagcatgtaccgatccggcagaaggtattcaggcatttctgaaagaactgagctgtcagtttagctgt accgaagatattgaaggtctgccggttggtctgctggcagcagaaaccagcctgaaaagcgaaccgctgcgt gaagcatgtcatgaagcatataaagaatgggccagcgtgtatgaagaaaaactgcgtcagaccggttgtagc gaaagccgtgcaaaagaagcaagcaccgttgttaatgcaatgattgaaggtggtattctgctgagcctgacc gcaaaaaatagcacaccgctgctgcatattagcagctgtattccggatctgctgaaacgttaa
phlF	gene(*13*)	atggcacgtaccccgagccgtagcagcattggtagcctgcgtagtccgcatacccataaagcaattctgacc agcaccattgaaatcctgaaagaatgtggttatagcggtctgagcattgaaagcgttgcacgtcgtgccggt gcaagcaaaccgaccatttatcgttggtggaccaataaagcagcactgattgccgaagtgtatgaaaatgaa agcgaacaggtgcgtaaatttccggatctgggtagctttaaagccgatctggattttctgctgcgtaatctg tggaaagtttggcgtgaaaccatttgtggtgaagcatttcgttgtgttattgcagaagcacagctggaccct gcaaccctgacccagctgaaagatcagtttatggaacgtcgtcgtgagatgccgaaaaaactggttgaaaat gccattagcaatggtgaactgccgaaagataccaatcgtgaactgctgctggatatgatttttggttttgt tggtatcgcctgctgaccgaacagctgaccgttgaacaggatattgaagaatttaccttcctgctgattaat ggtgtttgtccgggtacacagcgttaa
psrA	gene(*13*)	atggcacagagcgaaaccgttgaacgtattctggatgcagcagaacagctgtttgcagaacgtggttttgca gaaaccagcctgcgtctgattaccagcaaagccggtgttaatctggcagcagtgaattatcattttggcagc aaaaaagcactgattcaggcagtttttagccgttttctgggtccgttttgtgcaagcctggaacgtgaactg gaacgtcgtcaggcacgtccggaacagaaaccgagcctggaagaactgctggaaatgctggttgaacaggca ctggcagttcagcctcgtagcaataatgatctgagcatttttatgcgtctgctgggtctggcatttagccag agccagggtcatctgcgtcgttatctggaagatatgtatggtaaagtgttccgtcgttatatgctgctggtt aatgaagcagcaccgcgtgttccgcctctggaactgttttggcgtgttcattttatgctgggtgcagcagca tttagcatgagcggtattaaagcactgcgtgcaattgcagaaaccgattttggtattaacaccagcattgaa caggttatgcgtctgatggttccgtttctggcagcaggtatgcgtgcagatagcggtgttaccgatgaagca atggcagcagcacagctgcgtccgcgtagcaaaaccagcaccagcgcaaccaccgcaaaagcataa
qacR	gene(*13*)	atgaacctgaaagataaaattctgggcgttgccaaagaactgtttatcaaaaatggctataacgcaaccacc accggtgaaattgttaaactgagcgaaagcagcaaaggcaatctgtattatcactttaaaaccaaagagaac ctgtttctggaaatcctgaacatcgaagaaagcaaatggcaagagcagtggaaaaaagaacaaatcaaatgc aaaaccaaccgcgagaaattctatctgtataatgaactgagcctgaccaccgaatattactatccgctgcag aatgccatcatcgagttttataccgagtactataaaaccaacagcatcaacgagaaaatgaacaaactggaa aacaaatacatcgatgcctaccacgtgatctttaaagaaggtaatctgaacggcgaatggtgcattaatgat gttaatgccgtgagcaaaattgcagcaaatgccgttaatggcattgttacctttacccatgagcagaatatc aacgaacgcattaaactgatgaacaaattcagccagatctttctgaatggcctgagcaaataa
srpR	gene(*13*)	atggcacgtaaaaccgcagcagaagcagaagaaacccgtcagcgtattattgatgcagcactggaagttttt gttgcacagggtgttagtgatgcaaccctggatcagattgcacgtaaagccggtgttacccgtggtgcagtt tattggcattttaatggtaaactggaagttctgcaggcagttctggcaagccgtcagcatccgctggaactg gattttacaccggatctgggtattgaacgtagctgggaagcagttgttgttgcaatgctggatgcagttcat agtccgcagagcaaacagtttagcgaaatttctgatttatcagggtctggatgaaagcggtctgattcataat cgtatggttcaggcaagcgatcgttttctgcagtatattcatcaggttctgcgtcatgcagttacccagggt gaactgccgattaatctggatctgcagaccagcattggtgtttttaaaggtctgattaccggtctgctgtat gaaggtctgcgtagcaaagatcagcaggcacagattatcaaagttgcactgggtagcttttgggcactgctg cgtgaaccgcctcgttttctgctgtgtgaagaagcacagattaaacaggtgaaatccttcgaataa
L3S2P21	terminator(*6*)	ctcggtaccaaattccagaaaagaggcctcccgaaaggggggcctttttttcgttttggtcc
L3S3P31[b]	terminator(*6*)	ccaattattgaacaccctaacgggtgtttttttttttttggtctacc
L3S2P55	terminator(*6*)	ctcggtaccaaagacgaacaataagacgctgaaaagcgtctttttttcgttttggtcc
L3S3P11[b]	terminator(*6*)	ccaattattgaacacccttcggggtgtttttttgtttctggtctacc
L3S2P11	terminator(*6*)	ctcggtaccaaattccagaaaagagacgctttcgagcgtctttttttcgttttggtcc
ECK120033736	terminator(*6*)	aacgcatgagaaagccccccggaagatcaccttccgggggctttttttattgcgc
ECK120015170	terminator(*6*)	acaattttcgaaaaaacccgcttcggcgggttttttttatagctaaaa
L3S2P24	terminator(*6*)	ctcggtaccaaattccagaaaagacacccgaaagggtgtttttttcgttttggtcc
ECK120010876	terminator(*6*)	taaggttgaaaaataaaaacggcgctaaaaagcgccgtttttttttgacggtggta
ECK120033737	terminator(*6*)	ggaaacacagaaaaaagcccgcacctgacagtgcgggctttttttttcgaccaaagg
ECK120015440	terminator(*6*)	tccggcaatttaaaaaagcggctaaccacgccgctttttttacgtctgca
ECK120010818	terminator(*6*)	gtcagtttcacctgttttacgtaaaaacccgcttcggcgggttttttactttttgg
ECK120029600	terminator(*6*)	ttgagaagagaaaagaaaaccgccgatcctgtccaccgcattactgcaaggtagtggacaagaccggcggtc ttaagttttttggctgaa

a. Underline indicates the upstream promoter spacer.

b. The "C" at nucleotide 45 from was mutated to "A" to eliminate a BsaI recognition site.

Table S10: FISH probe sequences

Probe name	Size (nt)	%GC	DNA sequence
eYFP 1	20	60	TCCTCGCCCTTGCTCACCAT
eYFP 2	20	50	GCTGAACTTGTGGCCGTTTA
eYFP 3	20	65	CAGGGTCAGCTTGCCGTAGG
eYFP 4	20	55	TGCCTGTGGTGCAGATGAAC
eYFP 5	20	60	GTAGCCGAAGGTGGTCACGA
eYFP 6	20	60	TAGCGGGCGAAGCATTGCAG
eYFP 7	20	60	GTGCAGCTTCATGTGGTCGG
eYFP 8	20	55	GCATGGCGGACTTGAAGAAG
eYFP 9	20	65	CGCTCCTGGACGTAGCCTTC
eYFP 10	20	50	GTCGTCCTTGAAGAAGATGG
eYFP 11	20	65	CGGCGCGGGTCTTGTAGTTG
eYFP 12	20	60	GTGTCGCCCTCGAACTTCAC
eYFP 13	20	55	TTCAGCTCGATGCGGTTCAC
eYFP 14	20	55	CGTCCTCCTTGAAGTCGATG
eYFP 15	20	55	AGCTTGTGCCCCAGGATGTT
eYFP 16	20	45	GTGGCTGTTGTAGTTGTACT
eYFP 17	20	50	TGTCGGCCATGATATAGACG
eYFP 18	20	50	ACCTTGATGCCGTTCTTCTG
eYFP 19	20	50	TGTTGTGGCGGATCTTGAAG
eYFP 20	20	55	TGTTCTGCTGGTAGTGGTCG
eYFP 21	20	55	CTAAGGTAGTGGTTGTCGGG
eYFP 22	20	65	CTTTGCTCAGGGCGGACTGG
eYFP 23	20	60	TGATCGCGCTTCTCGTTGGG
eYFP 24	20	60	CACGAACTCCAGCAGGACCA
eYFP 25	20	45	TACTTGTACAGCTCGTCCAT

X. Supplementary References

1. G. Bennett, What is a part? *Draft BIOFAB Hum. Pract. Rep. 10* (2010).

2. C. Lou, B. Stanton, Y.-J. Chen, B. Munsky, C. A. Voigt, Ribozyme-based insulator parts buffer synthetic circuits from genetic context. *Nat. Biotechnol.* **30**, 1137–1142 (2012).

3. S. T. Lovett, R. L. Hurley, V. A. Sutera, R. H. Aubuchon, M. A. Lebedeva, Crossing Over Between Regions of Limited Homology in Escherichia coli: RecA-Dependent and RecA-Independent Pathways. *Genetics*. **160**, 851–859 (2002).

4. S. C. Sleight, B. A. Bartley, J. A. Lieviant, H. M. Sauro, Designing and engineering evolutionary robust genetic circuits. *J. Biol. Eng.* **4**, 1–20 (2010).

5. S. C. Sleight, H. M. Sauro, Visualization of Evolutionary Stability Dynamics and Competitive Fitness of Escherichia coli Engineered with Randomized Multigene Circuits. *ACS Synth. Biol.* (2013), doi:10.1021/sb400055h.

6. Y.-J. Chen *et al.*, Characterization of 582 natural and synthetic terminators and quantification of their design constraints. *Nat. Methods*. **10**, 659–664 (2013).

7. A. Khvorova, A. Lescoute, E. Westhof, S. D. Jayasena, Sequence elements outside the hammerhead ribozyme catalytic core enable intracellular activity. *Nat. Struct. Mol. Biol.* **10**, 708–712 (2003).

8. D. Dufour, M. de la Peña, S. Gago, R. Flores, J. Gallego, Structure-function analysis of the ribozymes of chrysanthemum chlorotic mottle viroid: a loop-loop interaction motif conserved in most natural hammerheads. *Nucleic Acids Res.* **37**, 368–381 (2009).

9. D. E. Ruffner, G. D. Stormo, O. C. Uhlenbeck, Sequence requirements of the hammerhead RNA self-cleavage reaction. *Biochemistry (Mosc.)*. **29**, 10695–10702 (1990).

10. J. Haseloff, W. L. Gerlach, Simple RNA enzymes with new and highly specific endoribonuclease activities. *Nature*. **334**, 585–591 (1988).

11. H. W. Pley, K. M. Flaherty, D. B. McKay, Three-dimensional structure of a hammerhead ribozyme. *Nature*. **372**, 68–74 (1994).

12. M. Zuker, Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.* **31**, 3406–3415 (2003).

13. B. C. Stanton *et al.*, Genomic mining of prokaryotic repressors for orthogonal logic gates. *Nat. Chem. Biol.* **10**, 99–105 (2014).

14. D. B. Goodman, G. M. Church, S. Kosuri, Causes and Effects of N-Terminal Codon Bias in Bacterial Genes. *Science*, 1241934 (2013).

15. S. Kosuri *et al.*, Composability of regulatory sequences controlling transcription and translation in Escherichia coli. *Proc. Natl. Acad. Sci.* **110**, 14024–14029 (2013).

16. V. K. Mutalik *et al.*, Precise and reliable gene expression via standard transcription and translation initiation elements. *Nat. Methods*. **10**, 354–360 (2013).

17. H. M. Salis, E. A. Mirsky, C. A. Voigt, Automated design of synthetic ribosome binding sites to control protein expression. *Nat. Biotechnol.* **27**, 946–950 (2009).

18. P. Shen, H. V. Huang, Homologous Recombination in Escherichia Coli: Dependence on Substrate Length and Homology. *Genetics*. **112**, 441–457 (1986).

19. R. Brayton, A. Mishchenko, in *Computer Aided Verification*, T. Touili, B. Cook, P. Jackson, Eds. (Springer Berlin Heidelberg, 2010; http://link.springer.com/chapter/10.1007/978-3-642-14295-6_5), *Lecture Notes in Computer Science*, pp. 24–40.

20. R. Brayton, G. Hachtel, C. McMullen, A. Sangiovanni, in *Proc. 1984 Custom Integrated Circuits Conference* (1984).

21. E. Aarts, J. Korst, W. Michiels, in *Search Methodologies*, E. K. Burke, G. Kendall, Eds. (Springer US, 2005; http://link.springer.com/chapter/10.1007/0-387-28356-0_7), pp. 187–210.

22. N. Metropolis, S. Ulam, The Monte Carlo Method. *J. Am. Stat. Assoc.* **44**, 335–341 (1949).

23. M. J. Smanski *et al.*, Functional optimization of gene clusters by combinatorial design and assembly. *Nat. Biotechnol.* **32**, 1241–1249 (2014).

24. S. Bhatia, M. Smanski, C. Voigt, D. Densmore, A combinatorial constraint specification framework for genetic design.

25. E. Oberortner, S. Bhatia, E. Lindgren, D. Densmore, A Rule-Based Design Specification Language for Synthetic Biology. *J Emerg Technol Comput Syst*. **11**, 25:1–25:19 (2014).

26. J. R. Hauser, Noise margin criteria for digital logic circuits. *IEEE Trans. Educ.* **36**, 363–368 (1993).

27. J. R. Kelly *et al.*, Measuring the activity of BioBrick promoters using an in vivo reference standard. *J. Biol. Eng.* **3**, 4 (2009).

28. Part:BBa J23101 - parts.igem.org, (available at http://parts.igem.org/Part:BBa_J23101).

29. Part:BBa B0064 - parts.igem.org, (available at http://parts.igem.org/Part:BBa_B0064).

30. B. P. Cormack, R. H. Valdivia, S. Falkow, FACS-optimized mutants of the green fluorescent protein (GFP). *Gene*. **173**, 33–38 (1996).

31. A. Raj, P. van den Bogaard, S. A. Rifkin, A. van Oudenaarden, S. Tyagi, Imaging individual mRNA molecules using multiple singly labeled probes. *Nat. Methods*. **5**, 877–879 (2008).

32. S. O. Skinner, L. A. Sepúlveda, H. Xu, I. Golding, Measuring mRNA copy number in individual Escherichia coli cells using single-molecule fluorescent in situ hybridization. *Nat. Protoc.* **8**, 1100–1113 (2013).

33. C. D. Smolke, T. A. Carrier, J. D. Keasling, Coordinated, Differential Expression of Two Genes through Directed mRNA Cleavage and Stabilization by Secondary Structures. *Appl. Environ. Microbiol.* **66**, 5399–5405 (2000).

34. E. Hiszczyńska-Sawicka, J. Kur, Effect ofEscherichia coliIHF Mutations on Plasmid p15A Copy Number. *Plasmid*. **38**, 174–179 (1997).

35. T. Durfee *et al.*, The Complete Genome Sequence of Escherichia coli DH10B: Insights into the Biology of a Laboratory Workhorse. *J. Bacteriol.* **190**, 2597–2606 (2008).

36. M. Galdzicki *et al.*, The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotechnol.* **32**, 545–550 (2014).

37. N. Roehner *et al.*, Proposed Data Model for the Next Version of the Synthetic Biology Open Language. *ACS Synth. Biol.* **4**, 57–71 (2015).

38. T. S. Moon, C. Lou, A. Tamsir, B. C. Stanton, C. A. Voigt, Genetic programs constructed from layered logic gates in single cells. *Nature*. **491**, 249–253 (2012).

39. 5´ RACE System for Rapid Amplification of cDNA Ends, (available at https://www.lifetechnologies.com/us/en/home/references/protocols/nucleic-acid-amplification-and-expression-profiling/cdna-protocol/5-race-system-for-rapid-amplification-of-cdna-ends.html).

40. S. K. Lee *et al.*, Directed Evolution of AraC for Improved Compatibility of Arabinose- and Lactose-Inducible Promoters. *Appl. Environ. Microbiol.* **73**, 5711–5715 (2007).

41. D. M. Dykxhoorn, R. St. Pierre, T. Linn, A set of compatible tac promoter expression vectors. *Gene*. **177**, 133–136 (1996).

42. R. Lutz, H. Bujard, Independent and Tight Regulation of Transcriptional Units in Escherichia Coli Via the LacR/O, the TetR/O and AraC/I1-I2 Regulatory Elements. *Nucleic Acids Res.* **25**, 1203–1210 (1997).

43. J. W. Young *et al.*, Measuring single-cell gene expression dynamics in bacteria using fluorescence time-lapse microscopy. *Nat. Protoc.* **7**, 80–88 (2012).

44. H. A. de Boer, L. J. Comstock, M. Vasser, The tac promoter: a functional hybrid derived from the trp and lac promoters. *Proc. Natl. Acad. Sci.* **80**, 21–25 (1983).

45. T. S. Moon *et al.*, Construction of a Genetic Multiplexer to Toggle between Chemosensory Pathways in Escherichia coli. *J. Mol. Biol.* **406**, 215–227 (2011).