S1 Derivation of the local concentration

To determine the potential speedup of localized hybridization circuits, we developed a biophysical model of tethered hybridization based on previous work [1], and calculated estimates for a local concentration $c$, based on parameters of the design of the tethers. We assume that the open hairpin DNA gate behaves analogously to the single-stranded DNA (ssDNA) spacer in [1], then use $L$ to denote its length and $s$ to denote persistence length. Also based on [1], we define $p(R)\,dR$ to be the probability that the open hairpin DNA gate is extended to a length $R$ within a range $dR$. Therefore, the probability $P$ for the exposed toehold to be in the reactive volume of the target complementary toehold is given by

$$P = p(R) a V_a$$

where $a$ is the width of the reactive volume, its volume is $V_a = a^3$, and the volume of a hemispherical shell of radius $R$ and thickness $a$ is given by $V_s = \frac{2}{3} \pi R^3 - \frac{2}{3} \pi (R - a)^3 \approx 2 \pi R^2 a$ (assuming $a \ll R$). Therefore,

$$P = p(R) \frac{a^3}{2 \pi R^2}$$

Again, following [1], the local concentration is given in particles /m$^3$ by dividing $P$ by the reactive volume $a^3$, which when converted to molar units is given by

$$c = \frac{p(R)}{2 \pi R^2} \cdot \frac{1}{1000 N_a}$$

where $N_a$ is Avogadro’s number.

S2 Probabilistic analysis using the chemical master equation

The chemical master equation (CME) describes the time-evolution of a stochastic system. If we describe the current state of a molecular system as a random variable $X \in \mathbb{R}^n$, the CME can be written as a linear first-order system of partial differential equations

$$\frac{dP(X = x, t)}{dt} = A.P(X = x, t) \quad (S1)$$

where $P(X = x, t)$ denotes the probability that the system at time $t$ takes on the vector-valued state $x$. Here, $A$ is the transition matrix, which describes the rate at which probability mass is transferred between states of the system. When performing stochastic simulations, we are essentially sampling the master equation for a stochastic trajectory. The master equation therefore describes all such trajectories. Unfortunately, combinatorial explosion of the number of states $(n)$ in the stochastic process often prevents direct use of the master equation, forcing us instead to use stochastic sampling (e.g. the method of [2]).

As an example of the CME, consider a binding reaction $R + L \overset{k}{\rightarrow} R-L$ in which we start with three molecules each of $R$ and $L$. The states of the system can be written down by sequentially applying the binding reaction to the initial configuration, which gives

$$\begin{pmatrix} R \\ L \\ R-L \end{pmatrix} = \left\{ \begin{pmatrix} 3 \\ 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix} \right\}$$
The state-space can be represented in a stoichiometry matrix $S$ given by

$$S = \begin{pmatrix} 3 & 3 & 0 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

where each row describes the number of copies of each species, which are arranged in columns. The associated entries of the transition matrix can be written down under the assumption of mass action as

$$A = \begin{pmatrix} -9k & 0 & 0 & 0 \\ 9k & -4k & 0 & 0 \\ 0 & 4k & -k & 0 \\ 0 & 0 & k & 0 \end{pmatrix}$$

where each row represents the gain of probability mass of each state from all other states in an infinitesimal time interval $\delta t$.

The CME can be integrated (numerically or otherwise) yielding a (deterministic) trajectory for the probabilities $P(X = x, t)$, which can be used to determine a variety of quantities that each evolve over time. Suppose $P_t$ is the matrix containing the trajectory, with each row relating the probability at a specific time, and each column representing a specific state. Then we define the following quantities

- the stochastic mean, $\langle x \rangle := \sum_j j P(X = j, t) = P_t S$
- the stochastic variance, $\sigma^2_x := \sum_j j^2 P(X = j, t) - \langle x \rangle^2 = P_t S^2 - P_t S$
- the marginal distribution of species $i$, $P(X_i = x_i, t)$

### S3 DSD Language and Implementation

#### S3.1 The DSD language

Models were constructed using Visual DSD [3], a programming language for the design and analysis of DNA strand displacement devices. Visual DSD is an implementation of the programming language and compiler described in [4], and features automatic compilation of programs to strand displacement reaction networks, together with stochastic and deterministic simulation methods. Programs are written in a textual syntax, described in [4], which supports modules and local parameters to allow for abstraction and code-reuse. A DSD program defines an initial collection of DNA species, which can be single or double-stranded, and the DSD compiler then computes the set of all strand displacement reactions that can be generated from these initial species. The generated reactions can then be simulated using stochastic or deterministic methods. The Visual DSD language is freely available from http://research.microsoft.com/dna.

Here we summarize the textual syntax of the DSD programming language used in this paper. The syntax is defined in terms of elementary sequences and species. A sequence $S$ comprises one or more domains, which can be long domains $x$ or short domains $tx^\wedge$. A species can be an upper strand $<S>$, a lower strand $[S]$ or a gate $G$. An upper strand $<S>$ denotes a sequence $S$ oriented from left to right, while a lower strand $[S]$ denotes a sequence $S$ oriented from right to left. A double strand $[S]$ denotes an upper strand $<S>$ bound to the complementary lower strand $[S^*]$. A gate $G$ is composed of double-stranded segments of the form $[L']<L>[S]<R>[R']$, which represents an upper strand $<L S R>$ bound to a lower strand $[L' S^* R']$ along the double-stranded region $[S]$. The overhanging sequences $L$, $L'$ and $R$, $R'$ can potentially be empty, in which case we simply omit them. Gates are built up by concatenating segments $G1$ and $G2$ along a common lower strand, written $G1:G2$. We let $D$ range over systems of species. Multiple systems $D1$, $D2$ can be present in parallel, written $D1|D2$. We also allow module definitions of the form $\text{def } X(n)=D$, where $n$ are the module parameters and $X(m)$ is an instance of the module $D$ with parameters $n$ replaced by $m$. We assume a fixed set of module definitions, which are declared at the start of the program.
S3.2 Implementation in Visual DSD

The latest version of Visual DSD incorporates a new library for numerically integrating ordinary differential equations, called Open Solving Library for ODEs (OSLO), which is available as a stand-alone package from http://research.microsoft.com/en-us/projects/oslo. For non-stiff problems and numerical integration of the CME, the RK54(7)M algorithm is used, and the solution post-processed to return the stochastic mean and variance, using the definitions above. Visual DSD shows the stochastic mean as a solid line, with a single standard deviation about the mean illustrated by a filled area plot.

Extensions to the DSD programming language enable probabilistic analyses to be run from a single button click. The new language elements are as follows:

- **directive simulation [simulation_mode]:** Setting the simulation mode in the code window removes the need for further interactions with the user interface, which in this case would amount to the selection of the appropriate simulation method from a drop-down menu. The simulation directive can be set to
  
  - **stochastic:** stochastic simulation using Gillespie’s direct method [2]
  - **deterministic:** integration of ODEs using the RK54(7)M algorithm
  - **deterministicstiff:** integration of ODEs using Gear’s BDF method
  - **cme:** integration of the chemical master equation (as described above)
  - **cmestiff:** integration of the chemical master equation using Gear’s BDF method

- **directive parameters [list]:** Free parameters can be defined, such that model simulations expect to be supplied with specific parameter values, and model exports appear with parameterized rate constants, initial conditions, etc.

- **directive sweep:** This directive enables multiple parameter values to be compared during simulation. The sweep parameter must also be defined in directive parameters. There are two ways of defining sweeps:
  
  - **Unnamed:** “directive sweep parameter_name = [1.0, 2.0, 3.0]” will sweep over the values 1.0, 2.0 and 3.0 for the parameter called parameter_name. Multiple usages of this directive produce independent sweep simulations, such that for each sweep directive, the value of other sweep parameters is set equal to its default value (as defined in directive parameters).
  - **Named:** “directive sweep mysweep = { parameter1 = [1.0, 2.0], parameter2 = [1.0, 2.0] }” will sweep over parameter1 and parameter2, creating the cross product of their values. Note that this contrasts with multiple **unnamed** sweeps. However, using multiple **named** sweeps produces independent sweep simulations, with unset parameters taking on their default values provided in directive parameters.
Figure S1: Models of diffusible input strands and transducer circuits. (a) All circuits with diffusible inputs. (b) Representative circuits with transduced inputs.
S4.1 Diffusible protector strands improve performance of localized circuits with diffusible inputs

To prevent localized protector strands from out competing the binding of diffusible input strands, the protector strands were changed to be non-localized. As such, for all circuits containing AND gates, the intermediate species of the form \( \langle XT \rangle \) were assumed to be diffusible. To facilitate probabilistic analysis, the protector strands were assumed to be at a constant concentration \( c_1 = 10 \text{ nM} \) throughout all calculations. Diffusible protectors led to a large improvement in the rate of progression in all circuits containing AND gates (Fig. S2).

Figure S2: Analysis of localized circuits with diffusible inputs and diffusible protector strands. All circuits were analyzed for their probability of completion when both inputs and protector strands (of the form \( \langle XT \rangle \)) were assumed to be diffusible. Diffusible species were assumed to stay at constant concentration, to facilitate probabilistic analysis. Inputs were set at \( c_0 = 100 \text{ nM} \), protector strands were set at \( c_1 = 10 \text{ nM} \), rates were as specified in the main text and local concentration was \( c = 10^4 \text{ nM} \).
S4.2 Transducers improve performance of localized circuits with diffusible inputs

As described in the main text, circuits with transduced diffusible inputs were considered. Most circuits with transduced inputs were faster than circuits with diffusible inputs and diffusible protector strands (Fig. S3).

Figure S3: Analysis of circuits with transduced diffusible inputs. All circuits were analyzed for their probability of completion when inputs were assumed to be transduced. Diffusible species were assumed to stay at a constant concentration, to facilitate probabilistic analysis. Inputs were set at \( c_0 = 100 \text{ nM} \), rates were as specified in the main text and local concentration was \( c = 10^4 \text{ nM} \).
S4.3 Localized circuits with transduced diffusible inputs and diffusible protector strands are limited by input binding

To investigate the effect of diffusion on circuits with transduced inputs and diffusible protector strands, we analyzed localized circuits with a high operating concentration of $c_0 = 1000$ nM, compared with $c_0 = 100$ nM (Fig. S4). At this higher operating concentration the circuits proceeded an order of magnitude faster. This confirms that the transduction of inputs by diffusion is indeed the rate limiting step for these circuits.

Figure S4: Localized circuits with transduced diffusible inputs are limited by input diffusion. All circuits were analyzed for their mean copy number (probability of completion) when the operating concentration $c_0$ of transduced diffusible inputs was increased to 1000 nM, as compared with 100 nM used in the main text, rates were as specified in the main text and local concentration was $c = 10^4$ nM. At the higher operating concentration $c_0$ the circuits proceeded an order of magnitude faster.
S4.4 Inlining the FANOUT in transduced LSB circuits reduces leaks

To assess the impact of leaks on the performance of the LSB circuits, we compiled these circuits in Infinite mode. This was because leaks in Detailed mode increased the number of states in the CTMC dramatically, which rendered these calculations intractable. However, to account for the observed plateau in response time in Infinite mode as local concentration increased, we multiplied $k_{\text{eff}}$ by $10^3$ nM to ensure that Infinite mode analysis was consistent with Detailed mode in terms of signal response time (compare leftmost panels of Fig. S5 with calculations of transduced inputs in the main text). In contrast, we scaled the effective leak rate by $c = 10^4$, since the leak rate is much lower and therefore does not become limited by branch migration. The effective leak rate was therefore set to $10^4 \times l \times \frac{k_{\text{migrate}}}{k_{\text{unbind}} + k_{\text{migrate}}}$.

In Fig. S5, analysis results of two alternative versions of the LSB0 and LSB1 circuits are shown. The first version is equivalent to the design analyzed in the main text, which avoids the use of the leaky FANOUT circuit by propagating diffusible inputs to multiple localized inputs, by means of multiple transducer gates. Since the diffusible inputs are present in excess, they can be transduced into multiple localized inputs simultaneously, without the need for FANOUT. The second design uses a single transducer for the diffusible input $x_{40}$, and then a localized FANOUT to produce two localized inputs $x_{401}$ and $x_{402}$ for the LSB0 circuit. Similarly, it uses a single transducer for $x_{41}$ followed by a localized FANOUT to produce $x_{411}$ and $x_{412}$ for the LSB1 circuit. To illustrate how removing the FANOUT reduces leaks, we show calculations over 24 hours, which showed that the production of erroneous output signal is approximately three times faster in LSB circuits using the FANOUT module. In addition, since the FANOUT is effectively the only component that exhibits zero-toehold leaks in the absence of inputs, removing this component significantly reduces leaks over longer time-periods. Furthermore, since circuits without the FANOUT exhibit no leaks at all in absence of inputs (analysis not shown), this would in enable circuits to be assembled and stored for long periods without leakage.

In the case of fully localized circuits that require the FANOUT, to reduce leaks we can reduce the local concentration of the protector strand in the FANOUT circuit, or alternatively render these strands diffusible and added as fuel. Using diffusible protector strands would have the disadvantage of slowing down the FANOUT circuit, but would in principle eliminate leaks in absence of input and fuel.
Figure S5: Leak from the FANOUT circuit can be replaced with multiple input channels. Probabilistic analyses are shown for LSB0 and LSB1 circuits with zero toehold leaks enabled. The effective leak rate of \( l \times \frac{k_{migrate}}{k_{bind} + k_{migrate}} \) (where \( l = 10^{-9} \text{ nM}^{-1}\text{s}^{-1} \)) was pre-multiplied by \( c = 10^4 \text{ nM} \). For normal toehold-mediated strand displacement, \( k_{\text{eff}} \) was instead multiplied by \( 10^3 \text{ nM} \) to account for the slowdown at high local concentrations in detailed mode. (a,c) Analysis of LSB0 and LSB1 circuits with transduced inputs and diffusible protector strands. (b,d) Analysis of LSB0 and LSB1 circuits where a FANOUT circuit is used to copy the \( x_{41} \) and \( x_{40} \) inputs into \( x_{411} / x_{412} \) and \( x_{401} / x_{402} \) respectively. In each figure part, two panels are shown due to the different timescales observable in the calculation output.

S4.5 State-space sizes for localized circuits with transduced inputs and diffusible protector strands

Table S1: State-space sizes for probabilistic model-checking. Recorded are the number of states in a CTMC for each circuit and each combination of inputs, corresponding to the circuits with transduced inputs and diffusible protector strands (main text).

<table>
<thead>
<tr>
<th>(NX,NY)</th>
<th>OR</th>
<th>AND</th>
<th>(NW,NX,NY,NZ)</th>
<th>ANDOR</th>
<th>ORAND</th>
<th>LSB0</th>
<th>LSB1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1)</td>
<td>10</td>
<td>5</td>
<td>(0,0,0,0)</td>
<td>1</td>
<td>1</td>
<td>26160</td>
<td>9200</td>
</tr>
<tr>
<td>(1,1)</td>
<td>96</td>
<td>45</td>
<td>(0,0,0,1)</td>
<td>5</td>
<td>8</td>
<td>35600</td>
<td>12000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0,0,1,0)</td>
<td>8</td>
<td>8</td>
<td>56960</td>
<td>19200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0,0,1,1)</td>
<td>48</td>
<td>64</td>
<td>78080</td>
<td>25600</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0,1,0,0)</td>
<td>5</td>
<td>11</td>
<td>23736</td>
<td>5750</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0,1,0,1)</td>
<td>25</td>
<td>93</td>
<td>33825</td>
<td>6875</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0,1,1,0)</td>
<td>40</td>
<td>93</td>
<td>54120</td>
<td>11000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0,1,1,1)</td>
<td>240</td>
<td>784</td>
<td>75536</td>
<td>14000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,0,0,0)</td>
<td>8</td>
<td>11</td>
<td>12775</td>
<td>14912</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,0,0,1)</td>
<td>40</td>
<td>93</td>
<td>18125</td>
<td>20800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,0,1,0)</td>
<td>64</td>
<td>93</td>
<td>29000</td>
<td>33280</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,0,1,1)</td>
<td>384</td>
<td>784</td>
<td>40400</td>
<td>46080</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,1,0,0)</td>
<td>48</td>
<td>112</td>
<td>9200</td>
<td>9384</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,1,0,1)</td>
<td>240</td>
<td>976</td>
<td>14000</td>
<td>12000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,1,1,0)</td>
<td>284</td>
<td>976</td>
<td>22400</td>
<td>19200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1,1,1,1)</td>
<td>2300</td>
<td>8448</td>
<td>32000</td>
<td>15600</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(NX,FANOUT,Length, WIRE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
S5  Generation of CTMCs from Visual DSD

We used Visual DSD to generate continuous time Markov chains for all of the circuits analyzed in this paper. Since all circuits were assumed to be tethered, we omitted explicit modeling of tethers. For diffusible input and protector strands, we assumed that these strands were at constant concentration. For a diffusible strand at constant concentration $c_0$, each bimolecular reaction with a given rate $k$ between a diffusible strand and a tethered strand was modeled as a unimolecular reaction with rate $c_0 \cdot k$.

![Diagram of reaction networks](image)

Figure S6: Reaction network for simplified models of localized OR, AND and FANOUT gates. We make the simplifying assumption that the output signal $O$ is tethered to the substrate, rather being part of a hairpin loop. We assume that all reactions proceed at roughly the same rate $r$, which is a parameter of the model. Note that the CTMCs in this figure are identical to the corresponding CTMCs in the main text.

Due to an existing limitation in the ability of Visual DSD to model branched structures, we used an encoding in which the output signal $O$ is tethered to the substrate, instead of being part of a hairpin loop. The CTMCs for elementary OR, AND and FANOUT gates generated using this encoding are shown in (Fig. S6). We note that the CTMCs are identical to the corresponding CTMCs generated with the hairpin encoding in the main text, up to renaming of states. In both cases, all transition rates are given by $c \cdot k_{eff}$. Although this simplification was not required to model elementary localized circuits, it was needed when the outputs of two circuits were used as inputs to an AND circuit, since this would result in the formation of a branched structure, which is currently not supported by Visual DSD. By explicitly separating the output from the circuit in the Visual DSD model, we avoided the need to represent branched structures, while still enabling the correct CTMC to be generated. We emphasize that this encoding is used purely as a means of generating the CTMC, and does not correspond to the biophysical implementation of the circuit. Since the CTMC is parameterized by local concentrations of interacting...
strands, a given CTMC can be used to explore a broad range of biophysical implementations, by varying the local concentrations accordingly.

**Figure S7**: The localized OR hairpin circuit (a) generates the identical CTMC as the localized non-hairpin encoding of the circuit (b). CTMCs were generated in Detailed mode, where $k_m$ represents the branch migration rate, $k_u$ the toehold unbinding rate and $ck_b$ the toehold binding rate scaled by the local concentration $c$ of interacting strands, which is a parameter of the model. To simplify the analysis and in the absence of further experimental data, all local concentrations were assumed to be approximately the same.
Figure S8: The localized AND hairpin circuit (a) generates the identical CTMC as the localized non-hairpin encoding of the circuit (b). CTMCs were generated as in Fig. S7.

We note that here we have assumed three distinct rates for binding, unbinding and branch migration, and that these rates are assumed to be largely unchanged across different circuit locations. However, in the general case each localized reaction could potentially take place at a different rate. For instance, different binding reactions could be scaled by different local concentrations due to variations in inter-tether distance, free energy, and tension arising from differences in molecular configurations at different locations. In the absence of detailed experimental measurements or more fine-grained biophysical models to constrain these parameters, we made the simplifying assumption that branch migration and unbinding rates are approximately constant throughout the circuit, and that binding rates are all scaled by the same local concentration. For the specific case of hairpin binding we note that, when \( c = 10^4 \) nM, hairpin binding is estimated by our model to be \( 10 \text{ s}^{-1} \), which coincides with experimental measurements [5, 6]. However, further work is needed to include additional parameters that more accurately characterize the variation in hairpin binding rates as a function of signal length.
S6  DSD code

directive duration Tf points 1000
directive compilation detailed
directive simulation cme
directive declare
directive migrate bm (* Per-nucleotide rate *)
directive concentration nM
directive time s
directive plot Output(O); Output(lsb0); Output(lsb1); Output(msb0); Output(msb1); Output(O1)
directive parameters [ rateScale = 1.0E+5; leakScale = 1.0E+5; Tf=300.0; c0 = 100.0; k = 1e-3; unbind = 10.0; c1 = 10.0; l = 1.0E-9; bm = 400.0; NW=0.0; NX=0.0; NY=0.0; NZ=0.0; N1=0.0; N2=0.0; N3=0.0; N4=0.0; N5=0.0; N6=0.0; N7=0.0; N8=0.0; N9=0.0; N10=0.0 ]
new T8 k*rateScale, unbind
def NS = 100.0
def CIN = c0 / rateScale
def COUT = c1 / rateScale

def Output(O) = <O>
def STRAND(X) = <T^ X>
def REVdiff(X) = constant COUT * <X T^>
def OR(X,Y,Z,O) = ( {T^*}[X T^]<O> | {T^*}[Y T^]<O> )
def OR(X,Y,Z,O) = ( {T^*}[X T^]<O> | {T^*}[Y T^]<O> | {T^*}[Z T^]<O> )
def AND(X,Y,Z,O) = ( {T^*}[X T^];{T^*}[Y T^]<O> )
def AND3(X,Y,Z,O) = ( {T^*}[X T^];{T^*}[Z T^]<O> )
def Output(O) = <O>
def STRAND(X) = <T^ X>
def REVdiff(X) = constant COUT * <X T^>
def OR(X,Y,Z,O) = ( {T^*}[X T^]<O> | {T^*}[Y T^]<O> )
def OR(X,Y,Z,O) = ( {T^*}[X T^]<O> | {T^*}[Y T^]<O> | {T^*}[Z T^]<O> )
def AND(X,Y,Z,O) = ( {T^*}[X T^];{T^*}[Y T^]<O> )
def AND3(X,Y,Z,O) = ( {T^*}[X T^];{T^*}[Z T^]<O> )
def STRANDdiff(X) = constant COUT * STRAND(X)
def STRANDdiff(N,X,Y,Z) = constant COUT * STRAND2(N,X,Y)
def REPORTER(X) = ( REPORTER(X) | constant COUT * Output(X) )
def REPORTERdiff(X) = ( REPORTER(X) | constant COUT * Output(X) )
new x10 new x11 new x20 new x21
new x30 new x31 new x40 new x41
new x40o new x41o
new a1 new a2 new a3 new a4
new a5 new a6 new a7 new a8
new a9 new a10
new msb0 new msb1 new lsb0 new lsb1
new W new N new X new Y new Z new A new B new O
new X1 new X2 new X3
new X1 new X2 new X3
new Y1 new Y2 new Y3
new O1 new O2 new O3
new Wa new Wb new Xa new Xb new Ya new Yb new Za new Zb
new Aa new Ab new Ba new Bb new Ba new Bb new Oa new Ob
new x10a new x10b new x11a new x11b
new x20a new x20b new x21a new x21b
new x30a new x30b new x31a new x31b
new x40a new x40b new x41a new x41b
new x401 new x402 new x411 new x412
new x40s = (x40a, x40b) def x41s = (x41a, x41b)
def x30s = (x30a, x30b) def x31s = (x31a, x31b)
def x20s = (x20a, x20b) def x21s = (x21a, x21b)
def x10s = (x10a, x10b) def x11s = (x11a, x11b)
def Ws = (Wa,Wb) def Xs = (Xa,Xb) def Ys = (Ya,Yb) def Zs = (Za,Zb)
def MSB() =
  ( AND(x30, x40, msb0) | REPORTER(msb0) )
def Strand4(Nx4, Nx3, Nx2, Nx1) =
  ( STRAND2(Nx4, x40, x41)
  | STRAND2(Nx3, x30, x31)
  | STRAND2(Nx2, x20, x21)
  | STRAND2(Nx1, x10, x11)
)
def Strand4diff(Nx4, Nx3, Nx2, Nx1) =
  ( STRAND2diff(Nx4, x40, x41)
  | STRAND2diff(Nx3, x30, x31)
  | STRAND2diff(Nx2, x20, x21)
  | STRAND2diff(Nx1, x10, x11)
)
def Transducer4(Nx4, Nx3, Nx2, Nx1) =
  ( Transducer2(Nx4, x40s, x401, x41s, x411)
  | Transducer2(Nx4, x40s, x402, x41s, x412)
  | Transducer2(Nx3, x30s, x30, x31s, x31)
  | Transducer2(Nx2, x20s, x20, x21s, x21)
  | Transducer2(Nx1, x10s, x10, x11s, x11)
)
def LSB0(x401, x402) =
  ( OR(x11, x21, a6)
  | AND(x10, x20, a7)
  | AND(x31, x401, a8)
  | OR3(a6, x30, x402, a9)
  | OR(a7, a8, a10)
  | AND(a9, a10, lsb0)
  | REPORTER(lsb0)
)
def LSB0local(N4, N3, N2, N1) = ( Strand4(N4, N3, N2, N1) | FANOUT(x40, x401, x402) | LSB0(x401, x402) )
def LSB0mix2(N4, N3, N2, N1) = ( Strand4diff(N4, N3, N2, N1) | LSB0(x40, x40) ) (* Add diffusible inputs *)
def LSB0trans(N4, N3, N2, N1) = ( Transducer4(N4, N3, N2, N1) | LSB0(x401, x402) ) (* Transduced inputs *)
def LSB0diff(x401, x402) =
  ( OR(x10, x20, a7)
  | OR(x30, x402, a9)
  | OR(a7, a8, a10)
  | ANDdiff(a9, a10, lsb0)
  | REPORTER(lsb0)
)
def LSB0mix(N4, N3, N2, N1) = ( Strand4diff(N4, N3, N2, N1) | LSB0diff(x40, x40) )
def LSB0transdiff(N4, N3, N2, N1) = ( Transducer4(N4, N3, N2, N1) | LSB0diff(x401, x402) )
def LSB1(x411, x412) =
  ( OR(x411, x412, a1)
  | OR(x411, x30, a3)
  | ANDdiff(x20, x12, a4)
  | AND(a1, a3, a5)
  | REPORTER(lsb1)
)
def LSB1local(N4, N3, N2, N1) = { Strand4(N4, N3, N2, N1) | FANOUT(x41, x411, x412) | LSB1(x411, x412) }
def LSB1mix2(N4, N3, N2, N1) = ( Strand4diff(N4, N3, N2, N1) | LSB1(x41, x41) )
def LSB1trans(N4, N3, N2, N1) = ( Transducer4(N4, N3, N2, N1) | LSB1(x411, x412) )
def LSB1diff(x411, x412) =
  ( OR(x11, x21, a1)
  | ANDdiff(x10, x20, a2)
  | OR(x411, x30, a3)
  | AND3diff(a2, x412, x31, a4)
  | ANDdiff(a1, a3, a5)
  | OR(a4, a5, lsb1)
  | REPORTER(lsb1)
)
def LSB1mix(N4, N3, N2, N1) = ( Strand4diff(N4, N3, N2, N1) | LSB1diff(x41, x41) )
def LSB1transdiff(N4,N3,N2,N1) = ( Transducer4(N4,N3,N2,N1) | LSB1diff(x411,x412) )

def ANDlocal(NX,NY) = ( NX*STRAND(X) | NY*STRAND(Y) | AND(X,Y,O) | REPORTER(O) )

def ANDmix2(NX,NY) = ( NX*STRANDdiff(X) | NY*STRANDdiff(Y) | ANDdiff(X,Y,O) | REPORTER(O) )

def ANDtrans(NX,NY) = ( Transducer(NX,(Xa,Xb),X) | Transducer(NY,(Ya,Yb),Y) | ANDdiff(X,Y,O) | REPORTER(O) )

def ANDtransdiff(NX,NY) = ( Transducer(NX,(Xa,Xb),X) | Transducer(NY,(Ya,Yb),Y) | ANDdiff(X,Y,O) | REPORTER(O) )

def ORlocal(NX,NY) = ( NX*STRAND(X) | NY*STRAND(Y) | OR(X,Y,O) | REPORTER(O) )

def ORmix(NX,NY) = ORmix2(NX,NY)

def ORtrans(NX,NY) = ( Transducer(NX,(Xa,Xb),X) | Transducer(NY,(Ya,Yb),Y) | ORdiff(X,Y,O) | REPORTER(O) )

def ORtransdiff(NX,NY) = ORtrans(NX,NY)

def FANOUTlocal(NX) = ( NX*STRAND(X) | FANOUT(X,O1,O2) | REPORTER(O1) | REPORTER(O2) )

def FANOUTmix2(NX) = ( NX*STRANDdiff(X) | FANOUT(X,O1,O2) | REPORTER(O1) | REPORTER(O2) )

def FANOUTtrans(NX) = ( Transducer(NX,(Xa,Xb),X) | FANOUT(X,O1,O2) | REPORTER(O1) | REPORTER(O2) )

def FANOUTmix(NX) = FANOUTmix2(NX)

def FANOUTtransdiff(NX) = FANOUTtrans(NX)

def WIRElocal(N,X,Y) = (N * STRAND(X) | WIRE(X,Y) )

def WIREmix2(N,X,Y) = (N * STRANDdiff(X) | WIRE(X,Y) )

def WIREtrans(N,X,Y) =
  ( constant N * CIN * <T^ Xa Xb>
  | N * [T^*][Xa][Xb T^]<X>
  | WIRE(X,Y) )

def WIRElocal(N,X,Y) = WIREmix2(N,X,Y)

def WIRElocal(N,X,Y) = WIREtrans(N,X,Y)

(*def Wires10local() =
  ( WIRElocal(N10,a10,a9) | WIRElocal(N9,a9,a8) | WIRElocal(N8,a8,a7) | WIRElocal(N7,a7,a6) | WIRElocal(N6,a6,a5)
   | WIRElocal(N5,a5,a4) | WIRElocal(N4,a4,a3) | WIRElocal(N3,a3,a2) | WIRElocal(N2,a2,a1) | WIRElocal(N1,a1,O)
   | REPORTER(O) )

def Wires10mix() =
  ( WIREmix(N10,a10,a9) | WIREmix(N9,a9,a8) | WIREmix(N8,a8,a7) | WIREmix(N7,a7,a6) | WIREmix(N6,a6,a5)
   | WIREmix(N5,a5,a4) | WIREmix(N4,a4,a3) | WIREmix(N3,a3,a2) | WIREmix(N2,a2,a1) | WIREmix(N1,a1,O)
   | REPORTER(O) )

def Wires10trans() =
  ( WIREtrans(N10,a10,a9) | WIREtrans(N9,a9,a8) | WIREtrans(N8,a8,a7) | WIREtrans(N7,a7,a6) | WIREtrans(N6,a6,a5)
   | WIREtrans(N5,a5,a4) | WIREtrans(N4,a4,a3) | WIREtrans(N3,a3,a2) | WIREtrans(N2,a2,a1) | WIREtrans(N1,a1,O)
   | REPORTER(O) )

def Wires10transdiff() = Wires10trans()*)

def AND3local(NX,NY,NZ) = ( NX*STRAND(X) | NY*STRAND(Y) | NZ*STRAND(Z) | AND3(X,Y,Z,O) | REPORTER(O) )

def ORANDrep() = ( OR(W,X,A) | OR(Y,Z,B) | AND(A,B,O) | REPORTER(O) )

def ORANDdiff() = ( OR(W,X,A) | OR(Y,Z,B) | ANDdiff(A,B,O) | REPORTER(O) )

def ORANDlocal(NW,NX,NY,NZ) = ( NW*STRAND(W) | NX*STRAND(X) | NY*STRAND(Y) | NZ*STRAND(Z) | ORANDrep() )

def ORANDmix2(NW,NX,NY,NZ) = ( NW*STRAND(W) | NX*STRANDdiff(X) | NY*STRANDdiff(Y) | NZ*STRANDdiff(Z) | ORANDrep() )

def ORANDtrans(NW,NX,NY,NZ) = ( Transducer(NW,WS,W) | Transducer(NX,XS,X) | Transducer(NY,YS,Y) | Transducer(NZ,ZS,Z) | ORANDrep() )

def ORANDtransdiff(NW,NX,NY,NZ) = ( Transducer(NW,WS,W) | Transducer(NX,XS,X) | Transducer(NY,YS,Y) | Transducer(NZ,ZS,Z) | ORANDdiff() )

def ANDORrep() = ( AND(W,X,A) | AND(Y,Z,B) | OR(A,B,O) | REPORTER(O) )

def ANDORdiff() = ( ANDdiff(W,X,A) | ANDdiff(Y,Z,B) | OR(A,B,O) | REPORTER(O) )

def ANDORlocal(NW,NX,NY,NZ) = ( NW*STRAND(W) | NX*STRAND(X) | NY*STRAND(Y) | NZ*STRAND(Z) | ANDORrep() )

def ANDORmix2(NW,NX,NY,NZ) = ( NW*STRAND(W) | NX*STRANDdiff(X) | NY*STRANDdiff(Y) | NZ*STRANDdiff(Z) | ANDORrep() )

def ANDORtrans(NW,NX,NY,NZ) = ( Transducer(NW,WS,W) | Transducer(NX,XS,X) | Transducer(NY,YS,Y) | Transducer(NZ,ZS,Z) | ANDORrep() )

def ANDORtransdiff(NW,NX,NY,NZ) = ( Transducer(NW,WS,W) | Transducer(NX,XS,X) | Transducer(NY,YS,Y) | Transducer(NZ,ZS,Z) | ANDORdiff() )

ANDORlocal(NW,NX,NY,NZ)
References


