1 Algorithm for finding the weights

In order to find the weights used in the main manuscript, we used a mix of perceptron algorithm and random search to find the weights $W_{ij}$. We decomposed our search in two steps. In a first step, we searched and optimized the weights assuming that inputs are in large excess over the weights. In a second step, we checked if those optimized weights were actually appropriate for intermediate ranges of input concentrations with numerical simulations.

In the first step, we started from random weights, chose randomly one pattern $X_1 \ldots X_4$ and computed the weighted sums $\sum_i W_{ij} X_i$ (which approximates the concentration of template $T_j$ produced in the limit of a large excess of inputs). If the template $T_j$ that is associated with this pattern did not maximize the weighted sums $\sum_i W_{ij} X_i$, the pattern was added to the weight $W_{ij}$. The process was iterated until all templates maximized the weighted sum associated with their patterns, or else it was restarted if the iteration was unsuccessful for too long. We then rounded the weights to conserve only two significant digits (which is justified by an experimental accuracy on the strand concentrations of a few percent).

In a second step, we scaled the weights and $c_0$ to obtain a robust response to leaks. Indeed the previous step does not ensure that the circuit will correctly associate templates and patterns when inputs are not in large excess, which is the case when realistic leaks are considered. If $c_0$ is chosen too large, a leak of $0.5c_0$ for a given bit will still be in large excess over the weights and give the same pattern as a bit set to $c_0$, which is clearly wrong if the actual bit was 0. We manually scaled and tested the weights and inputs concentration $c_0$ by simulating the kinetic response of the circuit to the patterns of Figure 4 until we found a robust response for all presented patterns (we used the parameters indicated in the manuscript).