Strand displacement neural networks and RNA translators

Daniel Svane Nielsen

June 14, 2017

Abstract

The purpose of this project is to investigate a previously designed method of creating strand displacement neural networks, and combine them with RNA translation, for possible future use in general biosensors and molecular computers. The strand displacement neural network is created using a DNA motif called the seesaw gate. The seesaw gate requires some specific domains to function, so to use arbitrary sequences as input to the network, the sequences have to be translated. This report first goes through the theory of neural networks and strand displacement reactions, and later applies it for recreating the seesaw neural network. It is successfully shown that simple neural networks (perceptrons) can be trained to realize specific logic operations based on their truth tables, but with some limitations. The RNA translator was intended to be transcribed from a DNA template, and the displacement reactions tested using a fluorescent reporter. It was not possible to transcribe the RNA strands, so no succesful RNA translation is shown in this report.

Contents

1. Introduction

Logic circuits using DNA and RNA has many interesting applications in diagnostics and treatment. An example is cancer detection, where miRNA's can be used as biomarkers [1]. These biomarkers can be used as inputs for logic circuits, which can be designed to activate fluorescence signals [2] or enzymes [3] when combinations of biomarkers are present or absent. For example, a circuit could be designed to activate when 2 unique miRNA's are present at the same time.

A different approach to building logic circuits is using neural networks. One of the advantages of this approach, is that large-scale networks does not have to be designed by someone with knowledge of logic gates and circuitry. The inputs of the circuit simply have to be defined along with the required output, and the neural network can be trained to give the desired functionality.

The neural network approach to strand displacement circuits has already been developed by Qian et all in 2011 [4]. The design is based on the seesaw gate motif, which requires that the inputs to the network must have very specific sequences. To use custom input sequences, like the miRNAs from cancer cells, the sequences have to be translated. This has also already been done, using two linked strand displacement reactions [5].

This project is split into two parts.

The first part aims to lay out the groundwork for a software where the miRNAs one wishes to detect can be entered. The strands and concentrations required for a neural network that implements a desired truth table is then calculated, and could be readily ordered and used as a biosensor.

The second part aims to translate one RNA sequence into another RNA sequence, and measuring the outcome by a fluorescent reporter. This is almost identical to the experiment in Picuri et al 2009 [5], but using RNA instead of DNA in the translation reactions. This is mostly done to show that the translators could also use RNA, but also to get some experience in a laboratory.

2. Theory

2.1 Strand displacement

Strand displacement can be used to implement DNA devices which can be used in molecular computing [6]. It uses the predictability of Watson-Crick base pairing (A pairs with T, C pairs with G) of the nucleotides of DNA to design reactions. When the nucleotides of two strands of DNA base pair with each other, the complex will have a certain free energy. By introducing another strand which will pair stronger in the complex (give a lower free energy), the original strand can be displaced, as seen in Figure 2.1.

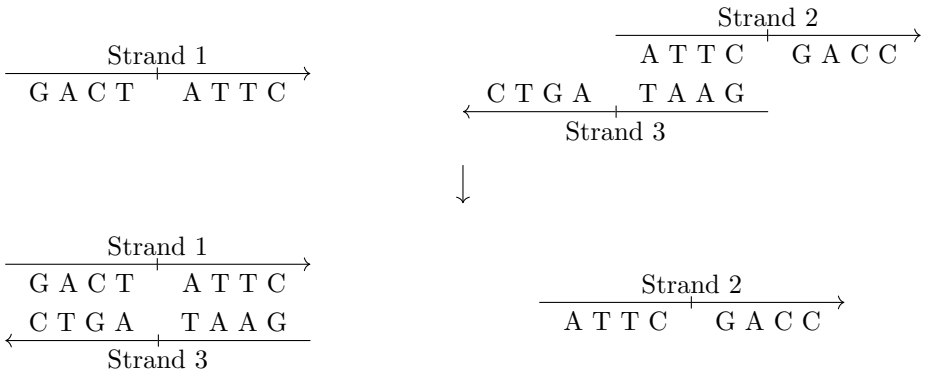


Figure 2.1: Strand displacement reaction where strand 1 base pairs at a lower free energy with strand 2, and displaces strand 3. The arrow denotes the strands 3' end.

The reaction in Figure 2.1 can be simplified by leaving out the specific bases, and instead naming each base pairing region. The same reaction as in Figure 2.1 can be seen in its simplified version in Figure 2.2 on the next page.

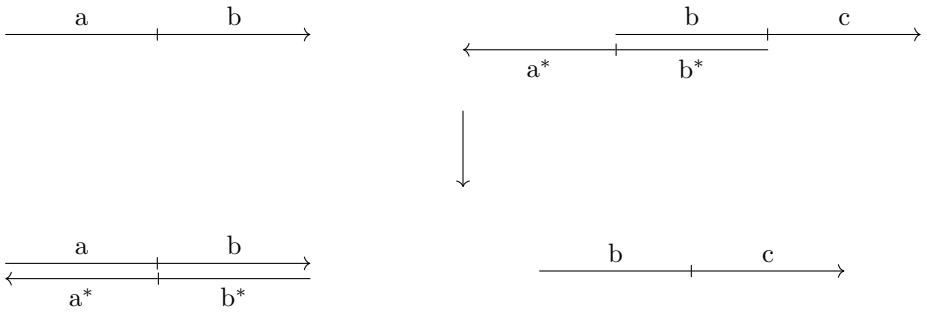


Figure 2.2: Simplified view of the reaction from Figure 2.1 on page 3. The sequence of nucleotides is replaced with a domain name. The asterix on the sequence name denotes that it is the reverse complement of the normal sequence (a^* is the reverse complement of a).

The free domain a^* in Figure 2.2 is known as a toehold [7]. The toehold serves the purpose of catalyzing the reaction by giving the strand ab a place to initialize displacement. The displacement reaction speed has been shown to be expontentially related to the toehold sequence length.

The result of the reaction in Figure 2.2, is that the strand bc is only single-stranded when displaced by the strand ab. The reaction can then be seen as a YES gate with input ab and outout bc. Why this is useful in molecular computing is not immediately apparent, but the strand displacement technique can be extended to larger reactions. A more interesting application is the AND gate seen in Figure 2.3 on the following page.

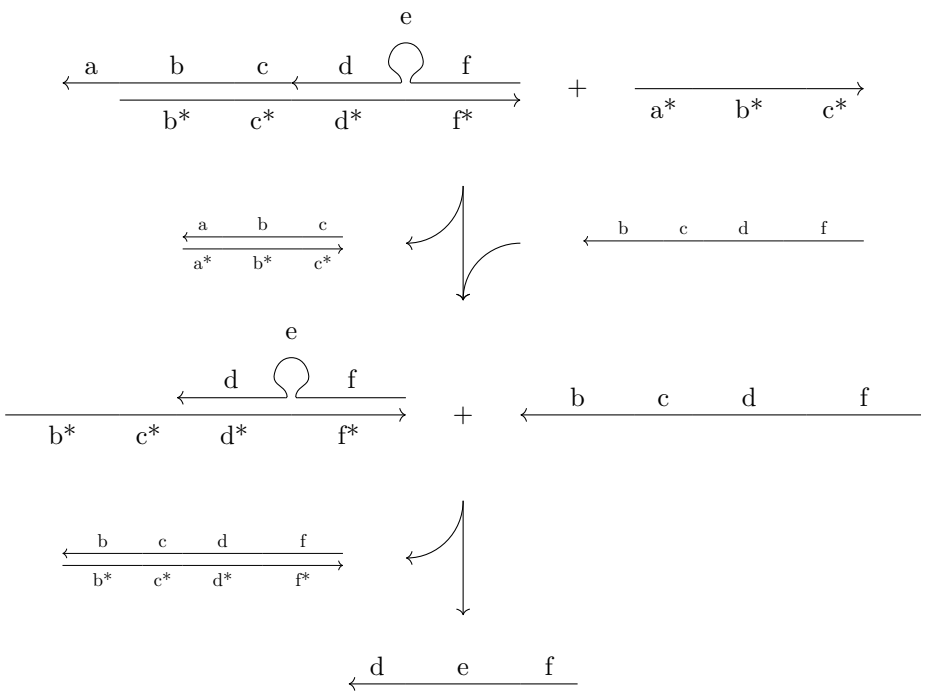


Figure 2.3: An AND gate made with strand displacement. After two strand displacements by the input strands $a^*b^*c^*$ and $bcdf$, the ouput strand def is released and can react in further strand displacement reactions with other logic gates, or activate enzymes and fluorescence signals. Without either of the input strands, the output will not be displaced. Adapted from Zhang et al 2011 [6].

The actual concentrations of each of the strand species when reacting, can be predicted by the free energy of each complex [7], but the time it takes for the reaction to take place is harder to predict. To do time analysis of larger strand displacement networks, software like Visual DSD [8] can be used.

2.2 Neural networks

2.2.1 In silico neural networks

Artificial neural networks (referred to just as neural networks) is a software implementation of the connection of neurons in the brain. In computer science they have been used to solve a wide variety of problems, like character and facial recognition. They present an advantage over conventional programmatic methods, as they don't need explicit coding for each new problem [9]. The simplest instance of neural networks is the single neuron, also known as the perceptron [10] (perceptron and neuron is used interchangeably in this report). The perceptron can be used for simple decision making (giving yes/no answers) from a set of inputs. The logical 2-input AND operator is an example of a problem that can be solved with the perceptron [11], where the perceptron can output 1 if both the inputs are 1, otherwise 0.

The perceptron works by taking all the inputs, and multiplying each input by its associated weight. If the sum of the multiplied inputs exceeds a certain threshold, the perceptron will output 1, otherwise 0. This is shown schematically in Figure 2.4.

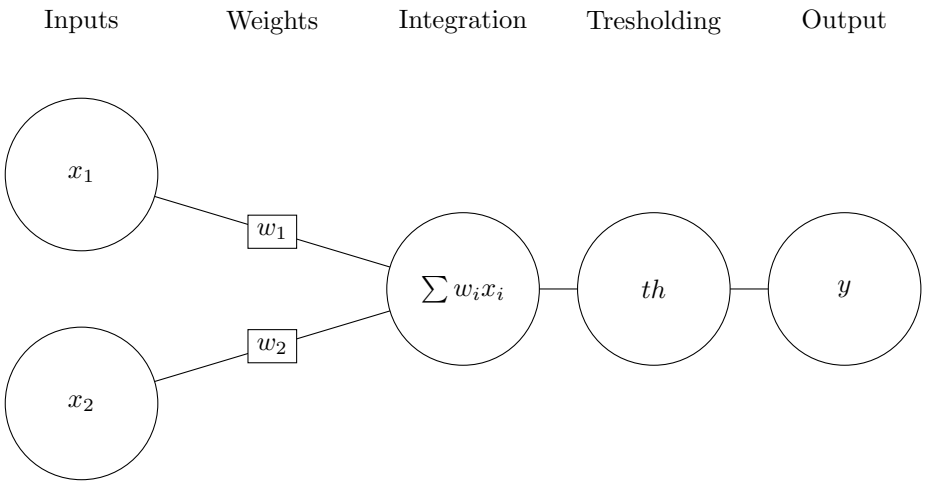


Figure 2.4: Diagram of the artificial perceptron.

The representation can be simplified to the type shown in Figure 2.5 on the following page. The weights and thresholds for the perceptron version of the 2-input AND gate can also be seen in Figure 2.5 on the next page.

The perceptron is limited to classifying inputs that are linearly separable [11], which rules out the XOR operation.

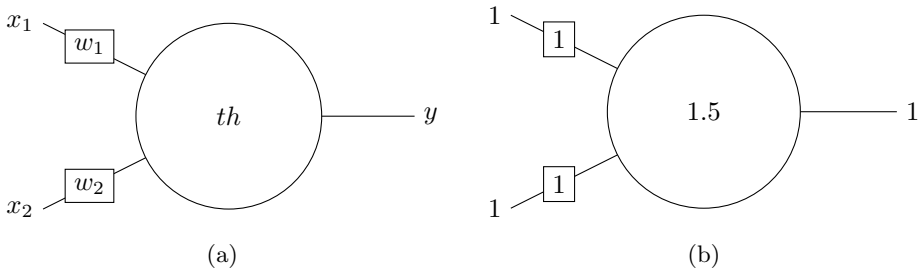


Figure 2.5: **(a)** Simple representation of a 2-input perceptron. **(b)** Weights and thresholds on a perceptron that implements the 2-input AND gate.

2.2.2 In vitro perceptron

It has previously been shown that the function of the artificial neuron can also be implemented using strand displacement reactions. The system is based on the seesaw gate motif [12], and can fulfill most of the functionality of a real neuron [4]. The seesaw gate is a catalytic gate with a threshold, designed for use in scalable strand displacement circuits. The seesaw gate uses a toehold to accelerate reactions, and has a left and right recognition domain to connect to other seesaw gates. The seesaw is named after the back and forth reaction of the strand displacement reactions, seen in Figure 2.6.

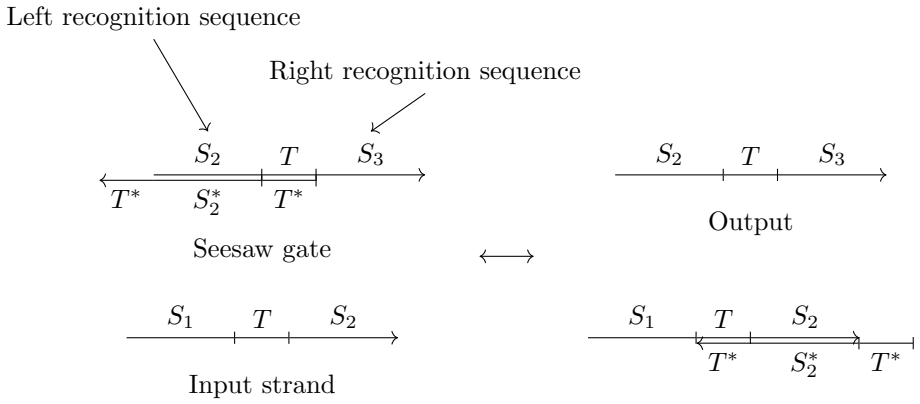
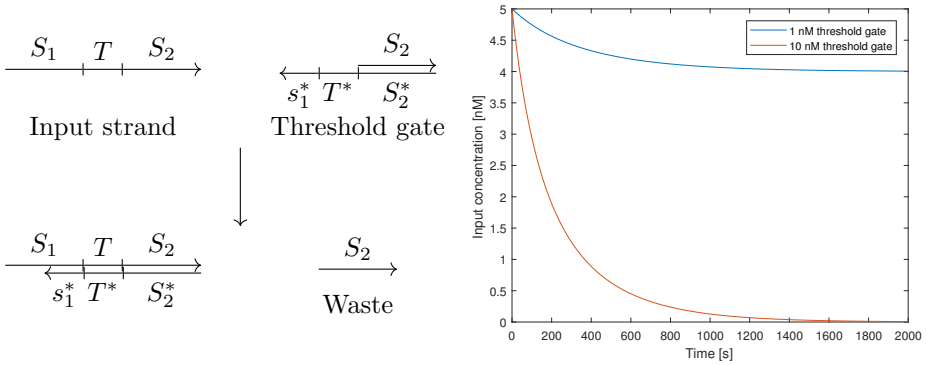


Figure 2.6: The back on forth reactions of the seesaw gate.

The reaction can be pushed to the right by using a fuel strand, or the input can be reduced by using a threshold gate, the details of which are discussed below.

Thresholding

In the artificial neuron, the neuron will activate when its inputs exceeds a threshold. This is implemented using a threshold gate which will bind the input and prevent it from reacting downstream in the network. If the threshold gate concentration is higher than the input concentration, the input will be suppressed by the threshold. If the threshold gate concentration is lower than the input concentration, not all of the input is suppressed, and will be able to react further downstream in the network. This reaction is shown in Figure 2.7a.



(a) (b)

Figure 2.7: **(a)** Reaction of an input strand with a threshold gate. The product has no free toehold domain, and can't undergo reverse reaction. The waste has no toehold, and can't parcitipate in further reactions. **(b)** Time analysis of the concentration of input strand (5 nM start concentration). A threshold concentration higher than the input (red) will bind all input strand. A lower concentration (blue), will allow the input strand to participate in further displacement reactions.

Integration

If the seesaw neuron have multiple inputs, they will have to be "collected" before thresholding, as they don't have the same left recognition sequence (see Figure 2.6 on page 7). This is done through an integrating gate, which will collect all inputs with the same right recognition sequence, and release a common signal which can be thresholded. The integration gate is seen in Figure 2.8a on the next page.

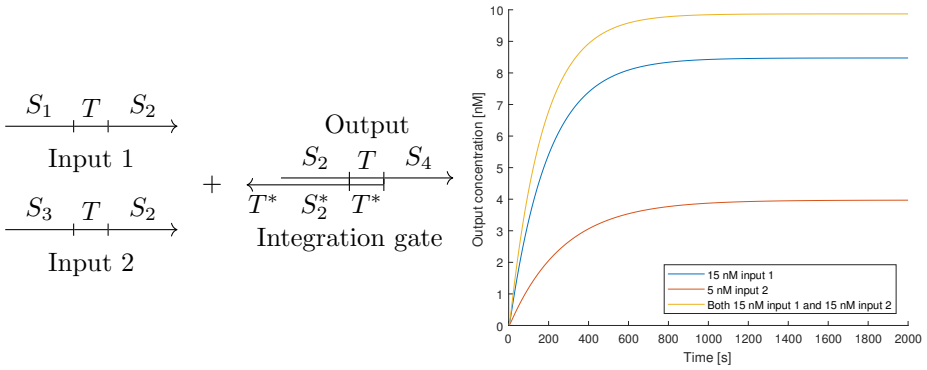


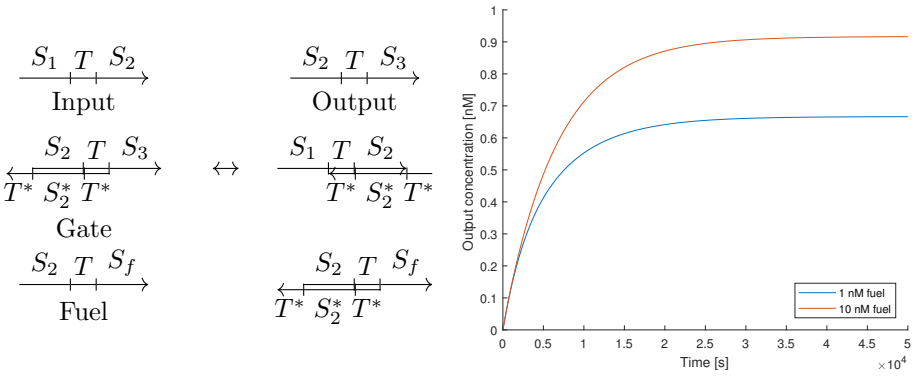
Figure 2.8: **(a)** Reaction of 2 input strands with an integration gate. The input strands have the same right recognition sequence S_2, and will both displace the top strand of the integration gate, releasing the output. **(b)** Time analysis of the concentration of the output strand. The concentration of the integration gate is 20 nM. The first input of 15 nM increases the output strand concentration, compared to the second input of 5 nM. When both strands are added the concentration of output increases further.

Weighting

The inputs to the gate are weighted using their concentration. By making the integration gates concentration the sum of all the input concentrations, a high concentration of one input strand will contribute more to the activation than that of a low concentration, as shown in Figure 2.8b.

The weight is decided by the concentration of output of an input gate and its fuel strand (see Figure 2.10 on page 11). The fuel serves the purpose of pushing the output of one gate to its target concentration, shown in Figure 2.9 on the following page.

A problem with this approach to weighting, is that the inputs can only contribute positively to the sum. In silico neural networks can use negative weights to simulate inhibitory synaptic connections, and is needed to implement many kind of boolean functions [11]. The in vitro network can't have negative concentrations of input sequences, so other approaches have to be considered. The problem can be solved using dual-rail logic [4], where each input is replaced by two inputs. The details of the dual-rail logic circuits is not a part of this project, and thus the networks will be limited to simple AND and OR circuits, since the NOT operation requires negative weights, and the XOR operation is not linearly separable [11].



(a) (b)

Figure 2.9: **(a)** Reaction of an input strand with a gate and fuel. The input displaces the output strand from the gate, and the fuel blocks the reverse reaction, pushing the equilibrium towards the free output. **(b)** Time analysis of the concentration of the output strand. Input and gate concentration is initially 1 nM. When the fuel concentration is low (blue), the output concentration doesn't reach the initial gate concentration. When the fuel concentration is high (red), the output concentration is pushed closer towards its maximum concentration.

Neuron

By combining the discussed functional elements of the seesaw gate, a "neuron" can be created using strand displacement reactions, as depicted in Figure 2.10 on the next page.

Instead of displaying all strands of the seesaw circuit, each gate can be represented by a simplified schematic. The first input gate from Figure 2.10 on the following page is shown in its simple version in Figure 2.11 on page 12. Using the simple representation, the seesaw neuron from Figure 2.10 on the following page can be simplified to the representation in Figure 2.12 on page 13.

The recommendations from the original paper [4] is that each fuel strands concentrations is twice that of the gate concentration. The threshold concentration on the input gates (as shown in Figure 2.12 on page 13) allows for concentrations of input strand less than 0.2 (relative concentration) to still be detected as 0.

The representation from Figure 2.12 on page 13 can further be simplified to the original perceptron shown in Figure 2.4 on page 6. The seesaw neuron thus successfully implements all the required functionality of the artificial neuron, except for negative weights.

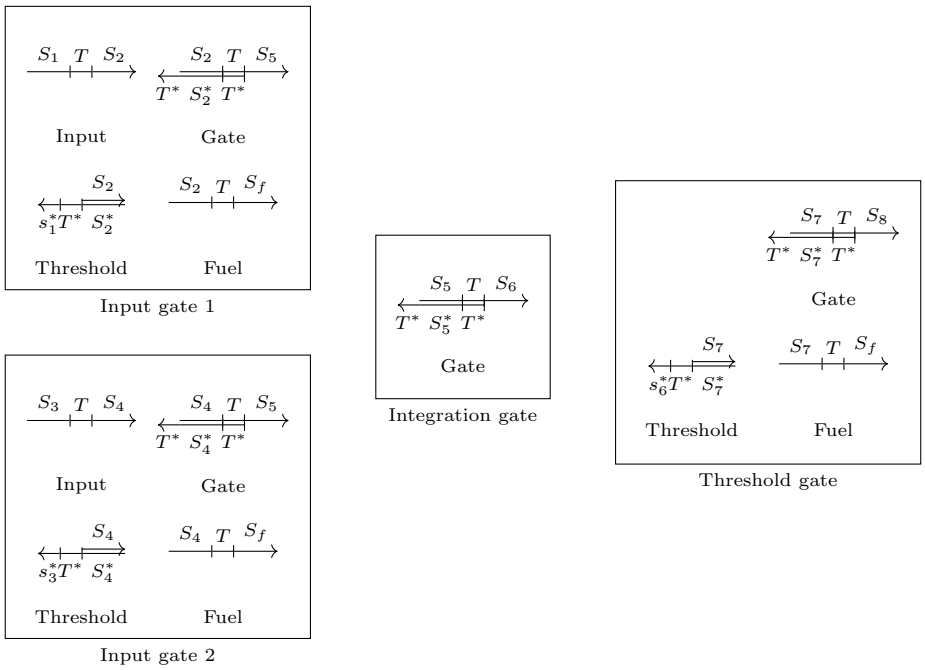
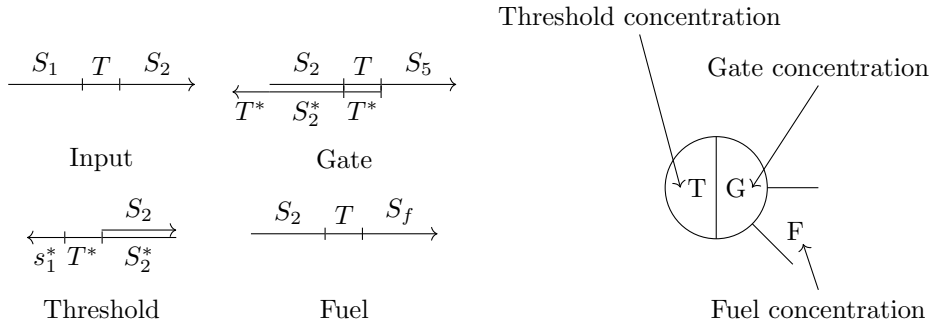


Figure 2.10: Schematic of a 2-input neuron implemented with seesaw gates. The neuron has an input gate for each input, which makes sure that the input concentration is higher than a given threshold before the input is registered. The gate and fuel concentrations in each input gate affects the weight of the input before it is sent to the integration gate. The integration gate collects the right recognition sequence from the input gates for the threshold gate. The threshold gate activates if the sum of the weighted inputs is larger than the threshold concentration of the treshold gate.

Training

The training algorithm typically used for digital perceptrons is based on the ability of the neurons to have negative weights and thresholds [13]. A modified algorithm is presented in the Qian et al 2011 [4], which works for dual rail circuits. For the circuits in this report without dual rail logic, the algorithm can be seen in Code 2.1 on the next page.

The algorithm in Code 2.1 on the following page will keep increasing the weights until the output of the network matches the desired output. The input sets are taken from the truth table that the circuit should realize. If the circuit for example should realize the simple AND gate (see Table 2.1 on page 13), the network is activated with the input rows from the truth table, and tested whether the output concentration equals (or is very close) to the output in the truth table.



(a) Seesaw gate strands

(b) Seesaw gate schematic

Figure 2.11: The strands and concentrations of the units of a seesaw gate can be represented to simplify larger diagrams.

```
initialize all weights to 0 and threshold to 10
while training is incomplete
  for all the input sets
    find the output of the network from the input set
    if output is not correct
      mark training as incomplete
  if training is marked as incomplete
    increase the weights
```

Code 2.1: Pseudocode for the seesaw perceptron training algorithm

2.2.3 Input translation

As per design of the seesaw gate, the input sequences for the strand displacement neural network needs very specific left-recognition, right-recognition and toehold sequences. To detect sequences that does not have these elements, the sequences have to be translated. It has previously been demonstrated that miRNAs can be translated to other sequences [5], by using two half-translators which are basic strand displacement reactions. An example of input translation can be seen in Figure 2.13 on the following page.

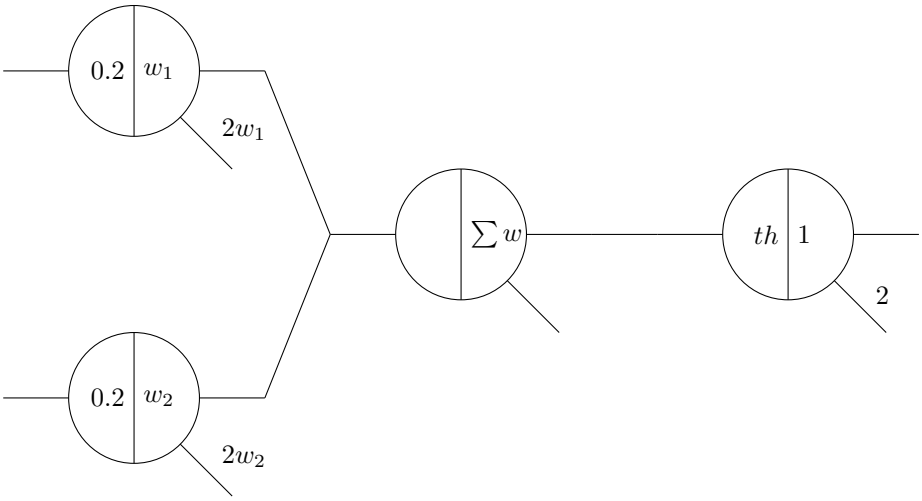


Figure 2.12: The seesaw neuron from Figure 2.10 on page 11 shown as its simple representation.

Input 1	**Input 2**	**Output**
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1: Truth table for an AND gate.

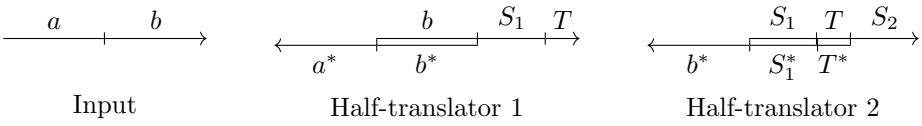


Figure 2.13: Translation of arbitrary input sequences into the syntax required for the seesaw neural network. The input strand ab displaces bS_1T from the first half-translator. bS_1T can then displace S_1TS_2 from the second half translator. This process successfully translates the input ab into S_1TS_2, which can then be used for input in a neural network (see Figure 2.10 on page 11).

3. Method

3.1 Perceptron simulation

To simulate a perceptron with strand displacement reactions, the Visual DSD software was used to implement each of the subunits of a seesaw neuron discussed in the theory section. The syntax of the four subunits required to create a perceptron can be seen in Code 3.1 (refer to the Visual DSD manual for the syntax [14]). Each of the subunits can be called with the desired left and right recognition sequences, when combining them for larger units. Note that each of the recognition sequences (S1 for example) is split into three (S1L, S1, S1R). This is needed to implement the threshold gate, which has a short part of the left recognition sequence (Figure 2.7a on page 8). The actual perceptron can be pieced together by combining the subunits from Code 3.1. The 2-input AND using seesaw gates seen in Figure 2.10 on page 11, can be written in Visual DSD as seen in Code 3.2 on the next page.

To abstract from the Visual DSD syntax, a transpiler was created in Javascript which can generate the needed Visual DSD code for perceptrons of variable input size. The transpiler can take a truth table defined as an array, and generates the needed Visual DSD code to create a perceptron of required input size. For further details, the source code is available on Github for running locally on a Windows machine [15].

```
def signal(N, S1L, S1, S1R, S2L, S2, S2R) = N * <S1L^ S1
    S1R^ T^ S2L^ S2 S2R^>
def threshold(N, S1R, S2L, S2, S2R) = N * {S1R^* T^*}[S2L^
     S2 S2R^]
def gate(N, S2L, S2, S2R, S3L, S3, S3R) = N * {T^*}[S2L^
    S2 S2R^ T^]<S3L^ S3 S3R^>
def fuel(N, S1L, S1, S1R) = N * <S1L^ S1 S1R^ T^ Sf>
```

Code 3.1: Visual DSD definition of the seesaw gate subunits.

```
(
(* First input gate *)
signal(0.9, S1L, S1, S1R, S2L, S2, S2R) |
threshold(0.2, S1R, S2L, S2, S2R) |
gate(1.0, S2L, S2, S2R, S3L, S3, S3R) |
fuel(2, S2L, S2, S2R) |
(* Second input gate*)
signal(0.9, S4L, S4, S4R, S5L, S5, S5R) |
threshold(0.2, S4R, S5L, S5, S5R) |
gate(1.0, S5L, S5, S5R, S3L, S3, S3R) |
fuel(2, S5L, S5, S5R) |
(* Summation gate *)
gate(2, S3L, S3, S3R, S6L, S6, S6R) |
(* Thresholding gate *)
threshold(1.5, S3R, S6L, S6, S6R) |
gate(1.0, S6L, S6, S6R, S7L, S7, S7R) |
fuel(2, S6L, S6, S6R)
)
```

Code 3.2: Code for a seesaw neuron in Visual DSD

The training algorithm generates the Visual DSD code for the seesaw neuron, runs it through the command-line version of the Visual DSD software, and simulates the output concentration of the perceptron. This output can then be compared to the expected output from the truth table, and the weights adjusted using the algorithm discussed in the theory section.

3.2 Translation

3.2.1 Design

The translator was designed to output the sequence CUAGACUGAAGCUCCUUGAGG (a miRNA previously used in the group). The input sequence was not specifically chosen, but generated by Nupack [16] to fulfill the design parameters. Two versions of the translator were designed, one where the half-translators annealed with 10 bp (Code A.1 on page 38) referred to as "short translator", and one where they annealed with 20 bp (Code A.2 on page 38) referred to as "long translator". The long translator was closer to the setup of the original DNA translator [5], which had annealing lengths of 23-25 bp.

When transcribing the RNA using the T7 RNA polymerase, the sequences are going to start with GG [17], so these were included in the design. Repeats of identical nucleotides were also prevented.

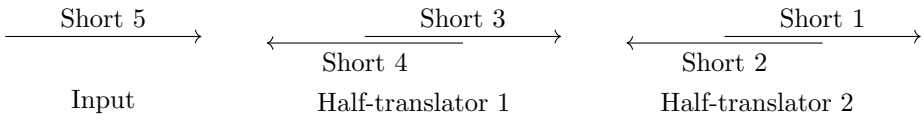


Figure 3.1: Sequence naming of the short translator sequences (see Table 4.2 on page 27).

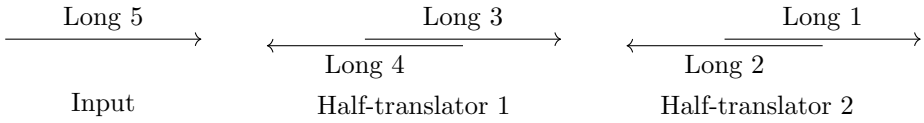


Figure 3.2: Sequence naming of the long translator sequences (see Table 4.2 on page 27).

To transcribe the RNA, the RNA sequences were converted to DNA, and the T7 promoter was added to the sequence. The reverse complement of the seqences was found, to serve as the template strand. The T7 RNA polymerase does not need the full template to be double-stranded, only the promoter sequence [17], so the resulting template will be the DNA version of the reverse complement of the target RNA (plus the reverse complement of the promoter), annealed with the promoter sequence. This is depicted in Figure 3.3. A program was written to convert the target RNA sequences to their DNA template directly on the output from Nupack, and format it for batch IDT ordering [18].

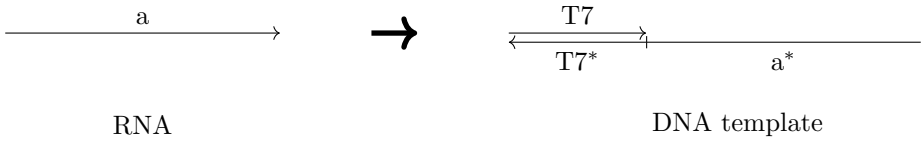


Figure 3.3: DNA template from RNA sequence for the T7 RNA polymerase. The reverse complement is taken of the RNA sequence, and the T7 promoter is added. When the T7 polymerase transcribes the template, the strand a is produced.

3.2.2 Transcription

Typhoon settings

The settings of the Typhoon scanner for all the SYBR Gold stained gels, was with the 488 nm laser and 555bp20 filter.

Dissolving oligos

The oligos from IDT were dissolved in TE buffer (Table A.1 on page 37, A refers to Appendix) to an approximate concentration of 120 µM, based on the quantity of substance written on the tubes. Their absorbance at 260 nm was then measured on the Nanodrop, and their concentration calculated using their extinction coeffecient. A program was written which can take the .csv output of the Nanodrop, and calculate the concentration based on each strands extinction coefficient [19]. The measured concentrations (Figure A.2 on page 39) was used to dilute the samples further, to a concentration of 100 µM.

Annealing template and promoter

To anneal the templates to the promoter, each of the template strands were mixed with equal amounts of promoter strand in annealing buffer (Table A.2 on page 37), to a final concentration of 1 µM. The mixed samples were then heated to 90° C for 5 minutes, and left to cool down to room temperature.

To check if samples annealed properly, they were run on a 20% native PAGE gel for 3 hours. Each lane was loaded with 50 µl sample, and 10 µl native loading buffer (Table A.3 on page 39). Afterwards the gel was stained in SYBR Gold, and visualised on the Typhoon scanner.

The second annealing gel was run in the same way as the first, except that only the short sequences were used, and the single strands were used as controls, along with a 10 bp ladder. 10 µl of templates, single strands and ladder was run with 1 µl of native loading buffer.

Transcription (first protocol)

The annealed DNA templates were mixed with the transcription reagents according to Table 3.1 on the next page. The samples were left overnight at 37°C. The day after, 1 µL of RNase-free DNase was added to the samples, and heated for 37°C for an hour. Afterwards, 100 µL of denaturing loading buffer was added to each sample, and 10 µL of the DNA 0 and DNA 5 strands (as controls) were mixed with 10 µL of denaturing loading buffer, and heated for 5 minutes at 90°C.

The transcribed sequences and controls were run on a 20% denaturing PAGE gel, for 4 hours at 20 W. The gel was then stained with SYBR Gold and scanned on the Typhoon.

	Initial conc.	Final conc.	Volume
Transcription buffer	10X	1X	10 µL
DTT	100 mM	10 mM	10 µL
NTP mix	25 mM	2.5 mM	10 µL
Template	500 nM	50 nM	10 µL
T7 RNA polymerase			1 µL
Nuclease-free water			59 µL
Total			100 µL

Table 3.1: Mixing of compounds for the first transcription done on the templates for both the short and long translater.

RNA purification

The purification of the RNA was done using ethanol precipitation. The bands were cut out from the gel, placed in eppendorf tubes, and dissolved in elution buffer (0.3 M NaAc, 0.25 mM EDTA) and left on a shaker in the cold room overnight. The tubes were then centrifuged for 10 minutes at room temperature. The supernatant was transferred to 10 mL spin tubes and 100 µL 3M NaOAc was added, and the samples were vortexed and spun down. Then 2.5 mL ethanol 96% was added and vortexed for 30 seconds. The samples were incubated on dry ice for 15 minutes. The samples were centrifuged at 14000 rpm at 4°C for 15 minutes, and the supernatant was discorded. Then 150 µL of ethanol 70% was added to the sample, and spun again at 14000 rpm for 5 minutes at 4°C. The supernatant was discarded, and the sample dried in fume-hood.

After purification the pellets were dissolved in 100 µL TE buffer, their absorbance measured on the Nanodrop, and their concentration calculated using their extinction coefficients. To check that the purification worked, 2 µL of each sample with 1 µL of denaturing loading buffer was run on a 20% denaturing gel, stained with SYBR Gold, and visualised on the Typhoon.

Translator annealing

Based on the concentrations in Figure A.3 on page 40, the samples were further diluted in RNase-free water to 2 µM for the translator strands (1-4) and 1 µM for the input strand (5). Then 25 µL of each of the translator strands were mixed with their respective counterpart (1+2 and 3+4) to a final concentration of 1 µM in 50 µL. The reporter samples were also mixed and diluted in RNase-free water to the same concentration as the translator subunits. To each of the subunits and the reporter, 5 µL of 10X annealing buffer was added. The samples were heated at 90°C for 5 minutes, and cooled off at room temperature.

To check if the subunits and reporter annealed properly, 2 µL of each of the annealed samples and their single strands as controls were mixed with 1 µL of native loading buffer, and run on a 20% native PAGE gel.

	Initial conc.	Final conc.	Volume
Transcription buffer	10X	1X	4 µL
DTT	100 mM	5 mM	2 µL
NTP mix	25 mM	2.5 mM	4 µL
Template	500 nM	250 nM	20 µL
T7 RNA polymerase			4 µL
Nuclease-free water			6 µL
Total			40 µL

Table 3.2: Mixing of compounds based on the transcription protocol from NEB, for the second transcription of the long translator.

	Initial conc.	Final conc.	Volume
HF	5X	1X	10 µL
Forward primer (T7 promoter)	100 µM	2 µM	1 µL
Reverse primer	100 µM	2 µM	1 µL
dNTP mix	25 mM	250 nM	0.5 µL
Template	1 µM	10 nM	0.5 µL
Phusion polymerase			0.5 µL
Nuclease-free water			36 µL
Total			50 µL

Table 3.3: Mixing of compounds for the PCR reaction on the long translator sequences.

Transcription (second protocol)

The long translator sequences were prepared as in the first protocol. The transcription was run for 3 hours at 37°C and then run on a 20% denaturing PAGE gel at 25 W for 2 hours, stained with SYBR Gold and scanned on the Typhoon.

Transcription (third protocol)

The transcription protocol was changed to one recommended from the supplier of the T7 polymerase [20]. The templates were mixed with the reagents according to Table 3.2, and the rest was carried out as in the second protocol.

PCR

The template strands were mixed with the reagents for a PCR reaction (Table 3.3), and run on the thermal cycler using annealing temperature of 61°C. The result was visualised by running the PCR product on a 2% agarose SB at 300 V for 15 minutes, stained in SYBR Gold and scanned.

Transcription (fourth protocol)

Another transcription reaction was carried out on the double-stranded long translator sequences using the first protocol. The samples were run on a 10% denaturing PAGE gel for 2 hours at 25 W, and the gel was stained with SYBR Gold and scanned.

4. Results and discussion

4.1 Simulation of seesaw perceptron

The seesaw perceptron with variable input size was created using the Visual DSD syntax through an abstraction layer written in Javascript. The software can take the truth table that should be fulfilled as an array. A seesaw perceptron with the correct number of input gates and the correct left and right recognition sequences is then automatically generated. For example, using the software with the 2-input AND gate (Table 2.1 on page 13) would generate the sequences seen in Figure 2.10 on page 11. The source code is available on Github [15].

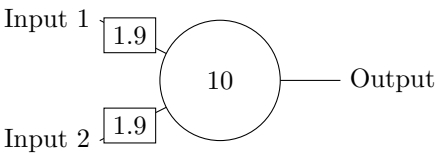
To train the perceptron, a row from the truth table is run through the Visual DSD simulation. Looking at the first row from Table 2.1 on page 13, the Visual DSD code for a 2-input perceptron is generated, with input 1 and input 2 concentrations of 0 nM, a weight of 0 and threshold of 10 (see Figure 2.12 on page 13). A time analysis is then done in Visual DSD of the output strand concentration. If the output concentration is equal to the output from the first row of the truth table (within a margin of error), the result is accepted, and the simulation moves on to the next row of the truth table. For the next row, new Visual DSD code is generated with input 1 concentration of 0 nM, and input 2 concentration of 1 nM. The output concentration is analyzed again. If the output is not equal to the one defined in the truth table, the weights of the perceptron is increased by 0.1, and the simulation is run for the next row of the truth table. This is repeated until a weight is found, where all rows from the truth table produces the correct output when run through the seesaw perceptron. This is summarized in Code 2.1 on page 12.

To test the perceptron compiling and training, 5 different truth tables were used, ranging from 2 to 3 inputs (Figures 4.1 to 4.5). The correct output was reached after 16-23 iterations of the learning algorithm. Input sizes greater than 3 were not tested, as the training time increases exponentially in time with the input size. Only truth tables that didn't involve NOT and XOR logic were used, as no solutions exist to these problems with the current algorithm (no linear separability and negative weights).

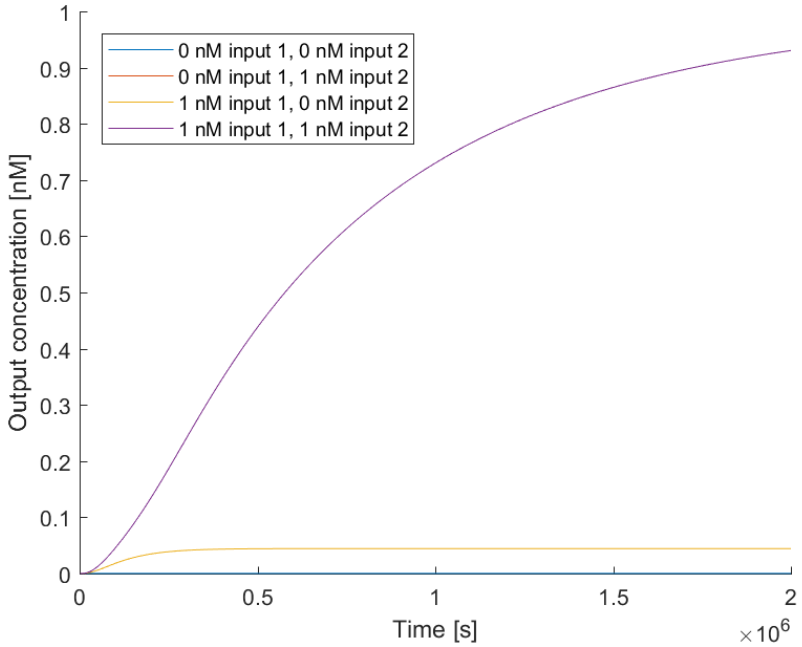
4.1.1 2-input AND

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

(a) Truth table for the 2-input AND gate.



(b) Diagram of the 2-input AND with correct weights and threshold.



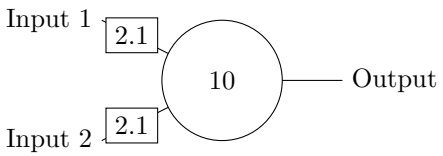
(c) Time analysis of the 2-input AND.

Figure 4.1: Simulation results of the trained 2-input AND gate. The network is trained to activate when both of the inputs are active. The correct output was obtained after 21 iterations of the training algorithm, with a weight of 1.9 for all inputs, and a threshold of 10.

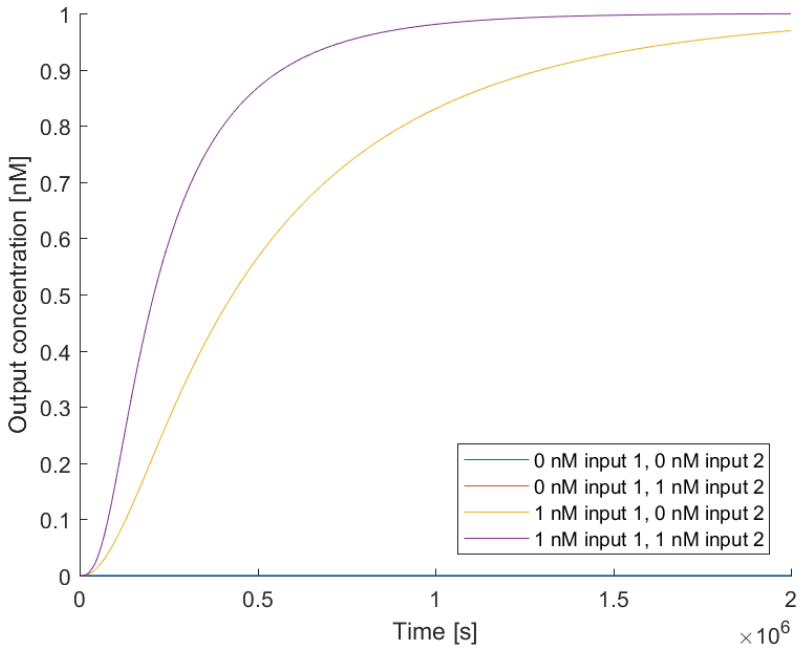
4.1.2 2-input OR

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

(a) Truth table for the 2-input OR gate.



(b) Diagram of the 2-input OR with correct weights and threshold.



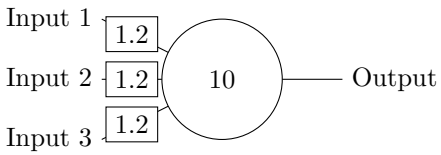
(c) Time analysis of the 2-input OR.

Figure 4.2: Simulation results of the trained 2-input OR gate. The network is trained to activate when one of the inputs is active. The correct output was obtained after 22 iterations of the training algorithm, with a weight of 2.1 for all inputs, and a threshold of 10.

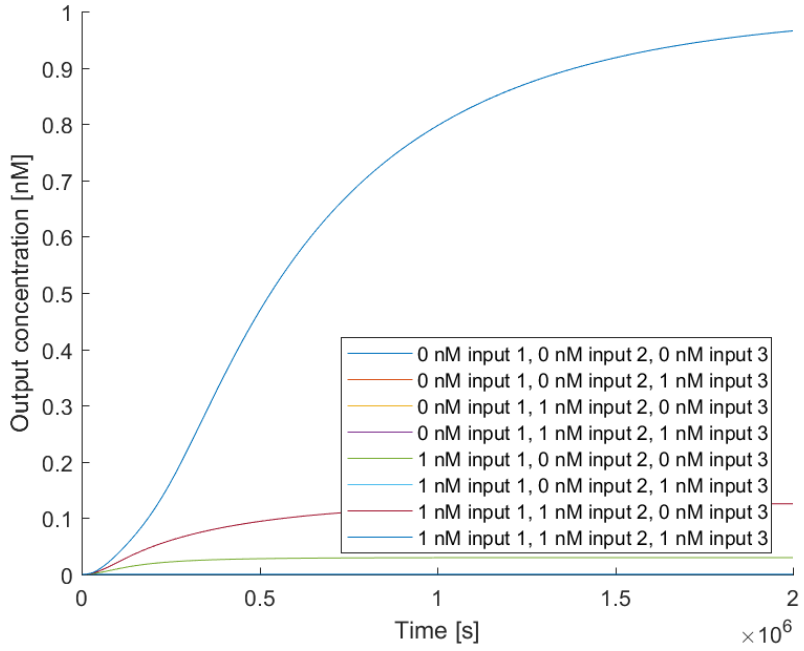
4.1.3 3-input AND

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(a) Truth table for the 3-input AND gate.



(b) Diagram of the 3-input AND with correct weights and threshold.



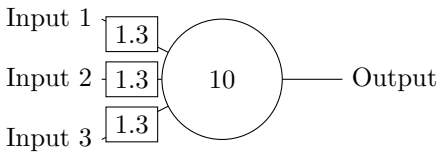
(c) Time analysis of the 3-input AND.

Figure 4.3: Simulation results of the trained 3-input AND gate. The network is trained to activate when all of the inputs are active. The correct output was obtained after 14 iterations of the training algorithm, with a weight of 1.2 for all inputs, and a threshold of 10.

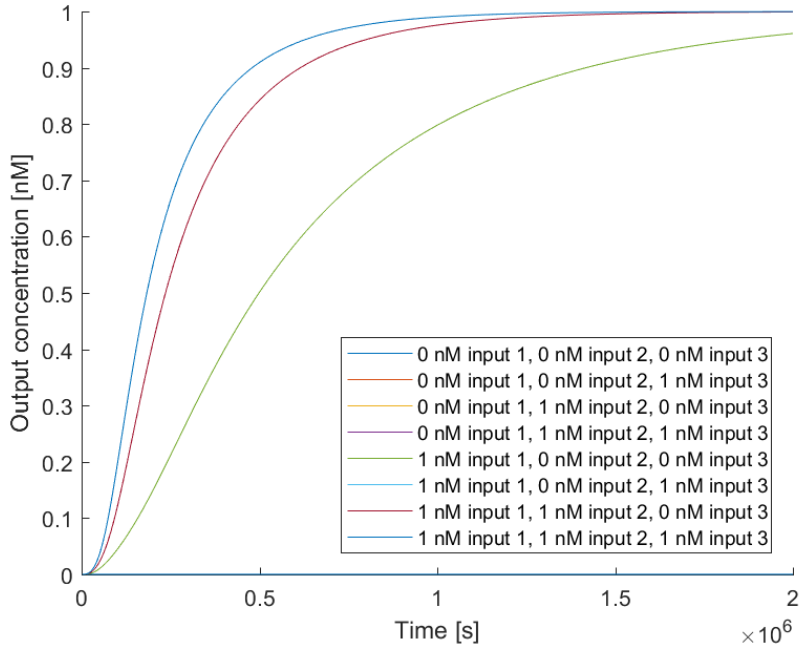
4.1.4 3-input 1-OR

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(a) Truth table for the 3-input 1-OR gate.



(b) Diagram of the 3-input 1-OR with correct weights and threshold.



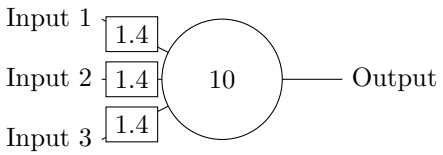
(c) Time analysis of the 3-input 1-OR.

Figure 4.4: Simulation results of the trained 3-input 1-OR gate. The network is trained to activate when at least 1 of the inputs is active. The correct output was obtained after 16 iterations of the training algorithm, with a weight of 1.3 for all inputs, and a threshold of 10.

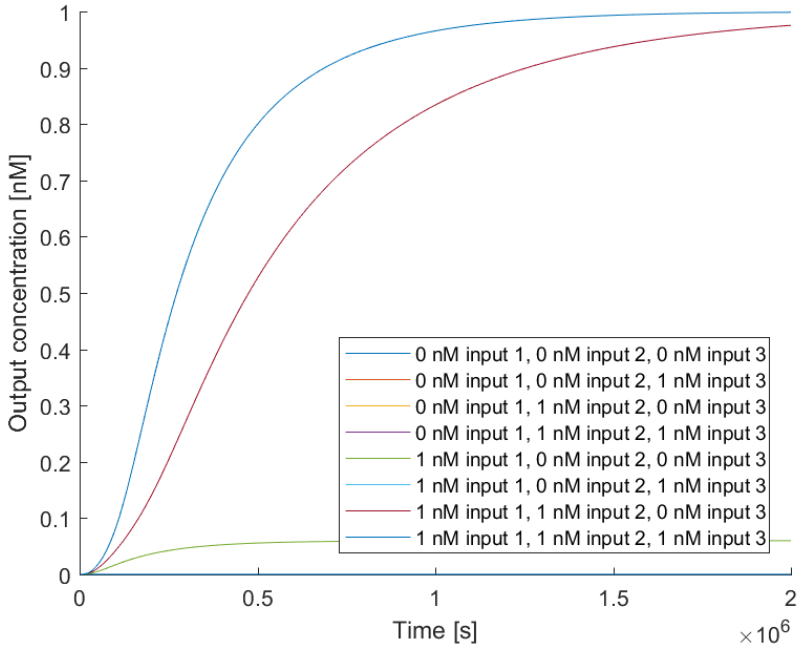
4.1.5 3-input 2-OR

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a) Truth table for the 3-input 2-OR gate.



(b) Diagram of the 3-input 2-OR with correct weights and threshold.



(c) Time analysis of the 3-input 2-OR.

Figure 4.5: Simulation results of the trained 3-input 2-OR gate. The network is trained to activate when at least 2 of the inputs is active. The correct output was obtained after 15 iterations of the training algorithm, with a weight of 1.4 for all inputs, and a threshold of 10.

Name	Short	Sequence	Length
T7 promoter	0	GGTAATACGACTCACTATAG	20
Short 1	1	CCTCAAGGAGCTTCAGTCTAGCCCTATAGTGAGTCGTATTACC	43
Short 2	2	CTCCTTGAGGCACATAACTCCCCTATAGTGAGTCGTATTACC	42
Short 3	3	CACATAACTCTACTAAATCTCCCTATAGTGAGTCGTATTACC	42
Short 4	4	GAGTTATGTGCCTCAAGGAGCCCTATAGTGAGTCGTATTACC	42
Short 5	5	AGATTTAGTAGAGTTATGTGCCCTATAGTGAGTCGTATTACC	42
Long 1	6	GTCAATTCGCCTCAAGGAGCTTCAGTCTAGCCCTATAGTGAGTCGTATTACC	52
Long 2	7	GCTCCTTGAGGCGAATTGACCCATCTTCATTCTACTCCTACCCTATAGTGAGTCGTATTACC	62
Long 3	8	CCATCTTCATTCTACTCCTATACCTCAATCCCCTATAGTGAGTCGTATTACC	52
Long 4	9	TAGGAGTAGAATGAAGATGGGTCAATTCGCCTCAAGGAGCCCCTATAGTGAGTCGTATTACC	62
Long 5	10	GATTGAGGTATAGGAGTAGAATGAAGATGGCCCTATAGTGAGTCGTATTACC	52
Beacon fluorophore	11	CGGCTAGACTGAA	13
Beacon quencher	12	CCTCAAGGAGCTTCAGTCTAGCCG	24

Table 4.1: Sequences and names of the DNA strands used for transcription.

Name	Short	Sequence	Length
Short 1	1	GGCUAGACUGAAGCUCCUUGAGG	23
Short 2	2	GGGAGUUAUGUGCCUCAAGGAG	22
Short 3	3	GGAGAUUUAGUAGAGUUAUGUG	22
Short 4	4	GGCUCCUUGAGGCACAUAACUC	22
Short 5	5	GGCACAUAACUCUACUAAAUCU	22
Long 1	6	GGCUAGACUGAAGCUCCUUGAGGCGAAUUGAC	32
Long 2	7	GGUAGGAGUAGAAUGAAGAUGGGUCAAUUCGCCUCAAGGAGC	42
Long 3	8	GGGAUUGAGGUAUAGGAGUAGAAUGAAGAUGG	32
Long 4	9	GGGCUCCUUGAGGCGAAUUGACCCAUCUUCAUUCUACUCCUA	42
Long 5	10	GGCCAUCUUCAUUCUACUCCUAUACCUCAAUC	32

Table 4.2: Sequences and names of the transcribed RNA strands.

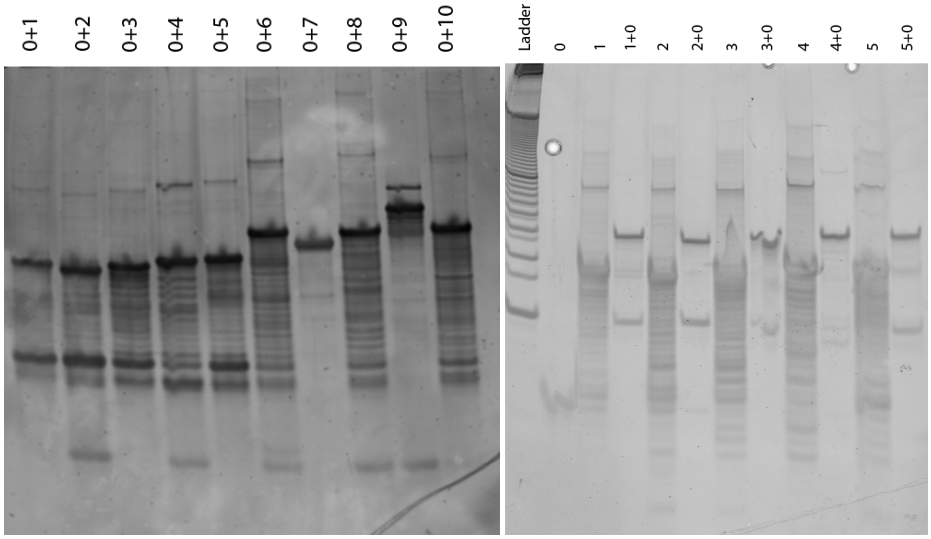
4.2 Translation of RNA sequences

4.2.1 Translator design

The sequences for the RNA translator was designed to output CUAGACUGAAGCUCCUUGAGG, through two strand displacement reactions. The design was done in Nupack using the code in Code A.1 on page 38 for the short translator and Code A.2 on page 38 for the long translator. The resulting RNA sequences from Nupack can be seen in Table 4.2. The RNA was converted to the required DNA template for transcription using a custom program [18], which implements the process shown in Figure 3.3 on page 16. The template DNA strands required for the transcription can be seen in Table 4.1. Refer to Figures 3.1 and 3.2 on page 16 for the naming of the strands.

4.2.2 Transcription

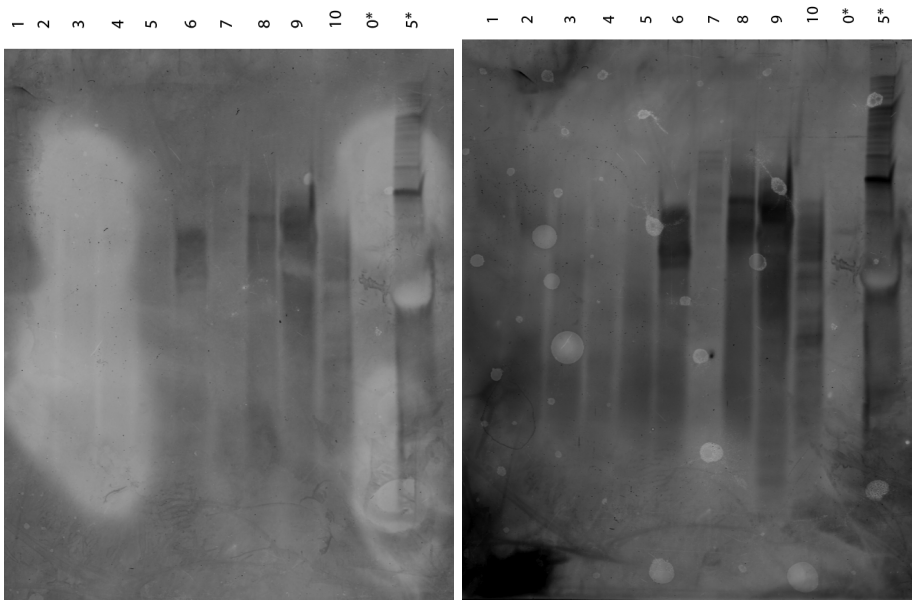
The result of the DNA template annealing can be seen in Figure 4.6a. The darkest bands are the annealed samples. The samples for the short translator runs at about the same size, while there is bigger variation in the long translator samples, as expected based on Table 4.1 on page 27. The exact positions of the long translator samples does not match with the sequence length, though this can be explained by secondary structures of the single-stranded part of the sample. The shorter bands visible below, are probably excess promoter, other secondary structures, and shorter sequences from synthesis errors. No lane with ladder was run, so the exact position of the bands can't be commented on.



(a) (b)

Figure 4.6: **(a)** Typhoon scan of the annealed templates and promoter strands. The lanes are labelled by which strands are annealed (see table 4.1). **(b)** The annealed strands again, together with the single strands as controls and a 10 nt ladder. The lanes are labelled with the strand names given in Table 4.1 on page 27. The plus symbol denotes which strands are annealed.

The transcription result of the annealed templates can be seen in Figure 4.7 on the following page. The lanes 1-10 does not show distinct bands. There seems to be more product from the long translator transcriptions (lanes 6-10), than in the short ones (lanes 1-5). Even the controls (lanes 11 and 12) which were loaded in equal amounts does not show up in equal strength. Since the gel from the DNA annealing was run without any controls, it was difficult to see if there was any errors in the annealing. To simplify the experiment while trying to find the error, only the short translator sequences were used, and a new annealing gel with proper controls was run (Figure 4.6b).

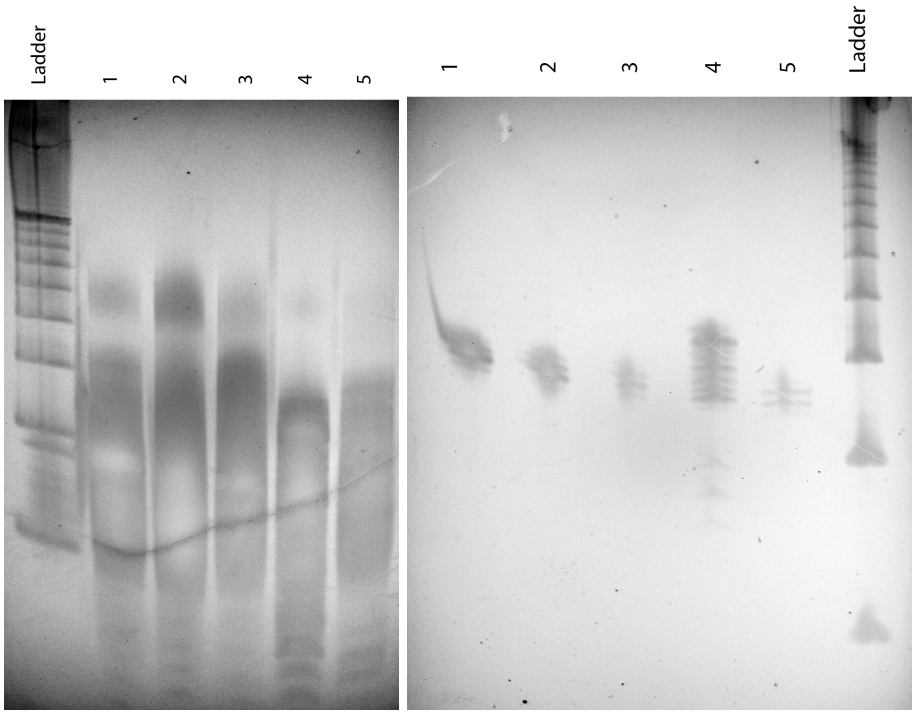


(a) (b)

Figure 4.7: **(a)** Typhoon scan of the transcribed RNA strands in lanes 1-10, and the original DNA strands as controls in 11 and 12. The lanes are labelled by which strands are annealed (see Table 4.2 on page 27). The asterix refers to the DNA strands in Table 4.1 on page 27. **(b)** The same gel after further staining in SYBR Gold.

The results of Figure 4.6b on page 28 still shows that the templates have annealed with the promoter, and runs as about 40-50 bp. The expected size is around 60 bp if the template was fully double-stranded (sum of promoter and template), but it is difficult to say how a partly annealed structure will run on a native gel. The new gel does however show that the bands in the annealed lanes below the assumed product, might not be excess promoter. The promoter is seen in the second lane, and lies below the bands in the annealed structure thought to be excess promoter. The bands below the product might be due to secondary structures of each of the template strands, but comparing with Figure A.4 on page 40, the band in the 5+0 lane would be expected to be less visible, as strand 5 has no secondary structure.

Despite the unexplained bands from the annealing, a new transcription was run on the short template strands to see if better results could be obtained. The UV shadowing still did not show any visible product, so the gel was scanned to check for bands (Figure 4.8a on the following page).



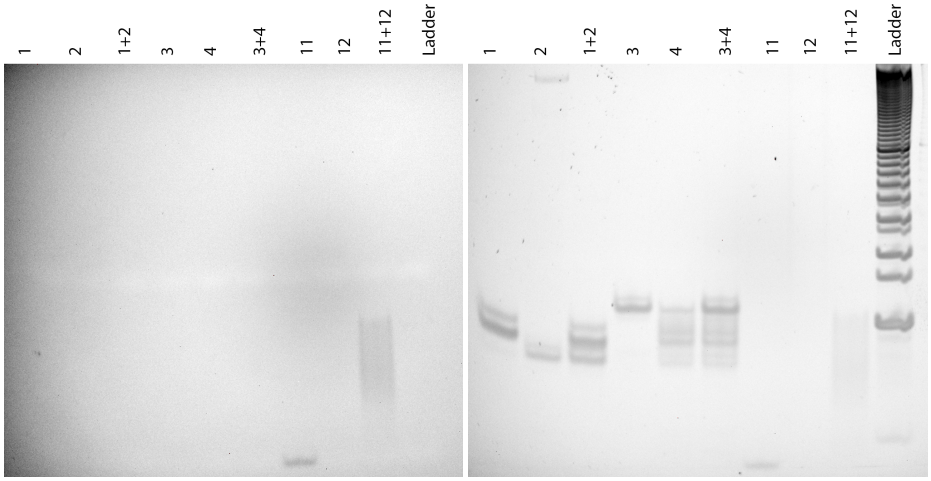
(a) (b)

Figure 4.8: **(a)** The transcribed short translator sequences and a 10 nt ladder. The lanes are labelled with the strand names given in Table 4.2 on page 27. **(b)** The purified short translator RNA strands with a 10 nt ladder.

As seen in Figure 4.8a, still no clear bands with RNA product was visible. After checking all buffers and materials, it was learned that the T7 polymerase that was used was from 2013, and might have been too old to still function properly. A fresh T7 polymerase was used for a new transcription. The UV shadowing did show some distinct bands this time, so the bands were cut out for purification.

Figure 4.8b shows that the RNA has been isolated, but is not very pure. It is possible that when cutting out the RNA from the gel before purification, too large an area was taken. The target RNA sequences were still expected to be in the purified samples (albeit not very pure), and it has previously been shown that strand displacement reactions can work with unpurified components [21], so the experiment continued with the current RNA.

The result of the annealed RNA translators and the reporter can be seen in Figure 4.9. It was expected that the fluorophore would be visible at the same wavelength as SYBR Gold, so a scan without staining should show the free fluorophore (and not the quenched fluorophore in the annealed reporter).



(a) Not stained (b) Stained

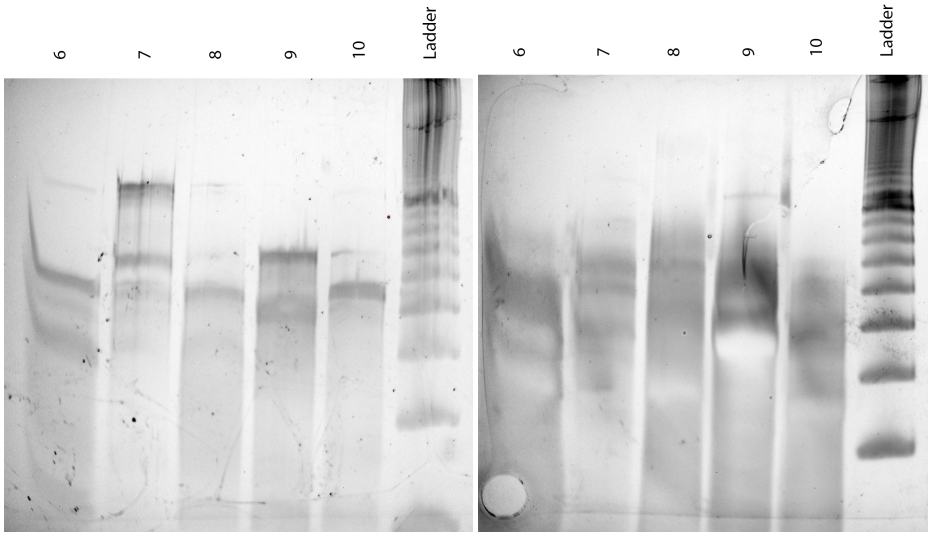
Figure 4.9: The annealed short translator subunits and reporter, with the single strands as controls, and a 10 nt ladder.

As seen in Figure 4.9a, the fluorophore clearly shows up in the unstained scan, and gets partly quenched and moves up when annealed to its quencher. The smear in the annealed reporter might be due to synthesis errors in the quencher strand.

In the stained scan (Figure 4.9b), the annealed half-translators does not seem to have annealed at all. Comparing with the controls, the strands that were supposed to have annealed, merely looks like the sum of the single strands. There is a few reasons why this could have happened. The RNA that was purified from gel was not the correct sequences, the annealing was not carried out correctly, or the strands simply doesn't anneal very well.

To check the strength of the annealing, the strands were analyzed in Nupack. The free energy analysis (Figure A.1 on page 37) shows that the short translator sequences does not anneal as well as the long ones. Due to time constraints, the focus moved towards the long translator sequences, as they were expected to provide better results.

The long translator sequences were transcribed using the first transcription protocol. The result can be seen in Figure 4.10a on the next page.



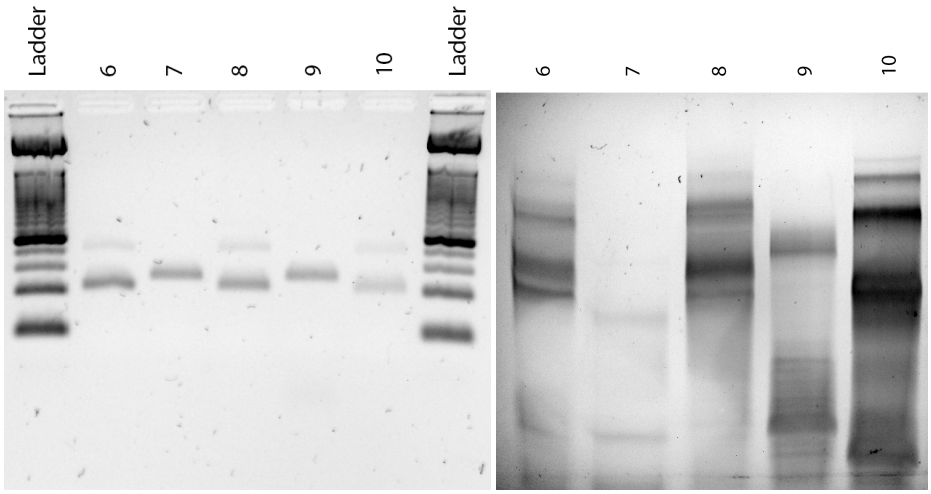
(a) (b)

Figure 4.10: **(a)** Transcription of the long translator sequences with a 10 nt ladder. **(b)** Transcription of the long translator sequences with a 10 nt ladder, using an adapted protocol.

As seen in Figure 4.10a, some RNA was produced so the concentration of nucleotides in the reaction was increased to get enough RNA to be visible in UV shadowing. The transcription protocol was also changed to one recommended from the supplier of the T7 polymerase [20], to ensure that the polymerase was not the problem. The new transcription revealed a single band (strand 9) in UV shadowing, but the rest of the templates had not produced enough RNA to be purified. A scan revealed once again that RNA had been produced, but the right product could not be isolated (Figure 4.10a).

The problem in getting pure RNA might be that the templates are only partly double-stranded with the T7 promoter sequence. To get fully double-stranded templates, some reverse primers for the template strands were ordered from IDT, and the T7 promoter could be used as the forward primer in a PCR reaction. The result of the PCR reaction can be seen in Figure 4.11a on the following page.

The result in Figure 4.11a on the next page shows the expected PCR product sizes compared with Table 4.1 on page 27. Another transcription reaction was carried out on the double-stranded long translator.



(a) (b)

Figure 4.11: **(a)** PCR product of the long translator sequences. **(b)** Transcription of the double-stranded long translator sequences.

Figure 4.11b shows strand 10 clearly, but still a lot of other products, and strand 7 has hardly been transcribed at all. The results might be explained by secondary structures of the RNA strands. If the strands form strong hairpins, the transcription can be aborted early, and produce unintended products. The RNA strands were checked in Nupack for secondary structures. Comparing the Nupack analysis in Figure A.5 on page 41 with the gel in Figure 4.11b, the amount of product seems to follow the free energy of the secondary structure of the RNA. Strand 10 which was visible in UV shadowing has no secondary structure with the highest probability, and strand 7 which shows the least amount of product has the lowest free energy.

The only way to avoid this problem is to create another design which doesn't have secondary structures, but since the translator sequences are locked by the desired input and output sequences, this is not a possibility. To actually get the RNA strands, they should have been ordered from IDT, synthesized and purified. This is more expensive, and was thought to be unnecessary in the beginning of the experiment. There was not enough time left to order the RNA sequences and carry out the fluorescence measurements on the reporter, so no results was obtained from the experiment.

5. Conclusion

The RNA strands for the translator couldn't be transcribed, possible due to secondary structures. It is not possible to avoid the secondary structures due to the restrictions of the translator design, so the RNA will have to be synthesized to create translators in RNA. The experiment was almost identical to the one in Picuri et al 2009, and there is nothing to suggest that the strand displacement shouldn't have worked using RNA instead of DNA [5]. The authors even use RNA as the input for the DNA translators.

The simulations of the neural network did show some promising results. It was possible to create multiple perceptrons of varying input size, and train them to different truth tables (Figures 4.1 to 4.5). The system requires no input from the user after having defined the desired truth table, and should theoretically be able to realize any truth tables of any size as long as they don't require negative weights, and are linearly separable.

The perceptron in this experiment is still only a stripped down version of the networks in Qian et all 2011, as it was not possible to apply the dual-rail logic needed for avoiding negative weights. Furthermore, the perceptrons trained in this experiment, could have been made much simpler by the logic gate system the same authors designed recently [21]. The only novel thing that came out of this experiment was the simplified training algorithm for the perceptrons without dual-rail logic, and the automatic generation of the perceptron based on a truth table.

There is also the problem of applying the simulations to in vitro reactions. Qian et all 2011 found some general guidelines for adjusting the concentrations that seemed to work for some networks, but it can't be expected that the sequences and concentrations found in the simulation will always work in vitro. They also express concern for moving to in vivo.

Still, the preliminary work done in this experiment could in theory be used to create a method for designing RNA/DNA detection kits. With some more work, the training algorithm and input translation could be combined, and tested in vitro.

Bibliography

[1] Yong Peng and Carlo M Croce. The role of MicroRNAs in human cancer. *Signal Transduction and Targeted Therapy*, 1:15004, jan 2016.

[2] Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-Free Nucleic Acid Logic Circuits. *Science*, 314(5805):1585–1588, 2006.

[3] W Engelen, B M G Janssen, and M Merkx. DNA-based control of protein activity. *Chemical communications (Cambridge, England)*, 52(18):3598–3610, 2016.

[4] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475, 2011.

[5] John M. Picuri, Brian M. Frezza, and M. Reza Ghadiri. Universal translators for nucleic acid diagnosis. *Journal of the American Chemical Society*, 131(26):9368–9377, 2009.

[6] David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103–113, 2011.

[7] David Yu Zhang and Erik Winfree. Control of DNA Strand Displacement Kinetics Using Toehold Exchange.

[8] Matthew R Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *BIOINFORMATICS APPLICATIONS NOTE*, 27(22):3211–321310, 2011.

[9] Jack V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, nov 1996.

[10] Richard P Lippmann. An Introduction' to Computing with Neural Nets.

[11] Zhao Yanling, Deng Bimin, and Wang Zhanrong. Analysis and study of perceptron to solve XOR problem. In *The 2nd International Workshop on Autonomous Decentralized System, 2002.*, pages 168–173. IEEE Comput. Soc.

[12] Lulu Qian and Erik Winfree. A Simple DNA Gate Motif for Synthesizing Large-Scale Circuits.

[13] S.I. Gallant. Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191, jun 1990.

[14] http://dsd.azurewebsites.net/beta/manual-beta.pdf.

[15] https://github.com/danielsvane/neuralnetworkcompiler.

[16] Joseph N. Zadeh, Conrad D. Steenberg, Justin S. Bois, Brian R. Wolfe, Marshall B. Pierce, Asif R. Khan, Robert M. Dirks, and Niles A. Pierce. NUPACK: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 32(1):170–173, jan 2011.

[17] John F. Milligan, Duncan R. Groebe, Gary W. Witherell, and Olke C. Uhlenbeck. Oligoribonucleotide synthesis using T7 RNA polymerase and synthetic DNA templates. *Nucleic Acids Research*, 15(21):8783–8798, 1987.

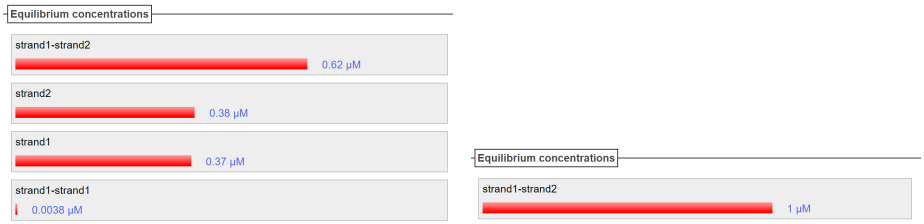
[18] http://nupackorder.herokuapp.com/.

[19] http://nanodropimport.herokuapp.com/.

[20] https://www.neb.com/protocols/2015/03/09/protocol-for-standard-rna-synthesis.

[21] Anupama J. Thubagere, Chris Thachuk, Joseph Berleant, Robert F. Johnson, Diana A. Ardelean, Kevin M. Cherry, and Lulu Qian. Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nature Communications*, 8:14373, 2017.

A. Appendix



(a) Equilibrium concentration of the short translator strands 1 and 2.

(b) Equilibrium concentration of the long translator strands 6 and 7.

Figure A.1: Nupack analysis of the equilibrium concentrations of the first subunit of the short and long translator.

	Concentration
Tris-HCl pH 8.0	10 nM
EDTA	1 mM
Nuclease-free water	

Table A.1: Contents of 1X TE buffer.

	Concentration
Tris pH 8.0	10 nM
NaCl	50 mM
EDTA	1 mM
Nuclease-free water	

Table A.2: Contents of 1X annealing buffer.

```
structure state1 = ......................
structure state2 = ............((((((((((+............))))))
    ))))
structure state3 = ............((((((((((+............))))))
    )))))
structure state4 = ......................

domain a = N10
domain b = N10
domain c = CUCCUUGAGG
domain d = CUAGACUGAAG
domain e = GG

state1.seq = e a b
state2.seq = e c a e b* a*
state3.seq = e d c e a* c*
state4.seq = e d c

prevent = AAAA, CCCC, GGGG, UUUU, KKKKKK, MMMMMM, RRRRRR,
    SSSSSS, WWWWWW, YYYYYY
```

Code A.1: Nupack code for the short translator

```
structure state1 = ..............................
structure state2 =
    ......................((((((((((((((((((+............))
    ))))))))))))))))
structure state3 =
    ............((((((((((((((((((+......................))
    ))))))))))))))))
structure state4 = ..............................

domain a = N20
domain b = N10
domain c = GCUCCUUGAGG N9
domain d = CUAGACUGAA
domain e = GG

state1.seq = e a b
state2.seq = e c a e b* a*
state3.seq = e d c e a* c*
state4.seq = e d c

prevent = AAAA, CCCC, GGGG, UUUU, KKKKKK, MMMMMM, RRRRRR,
    SSSSSS, WWWWWW, YYYYYY
```

Code A.2: Nupack code for the long translator

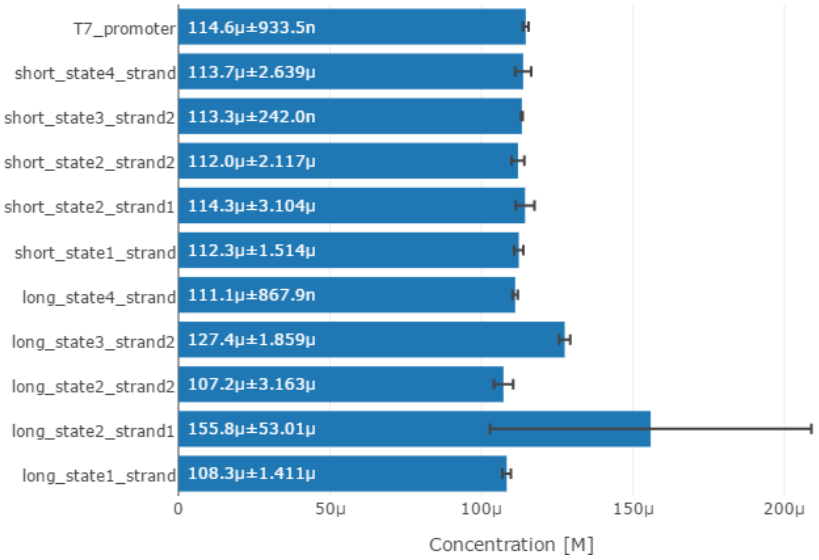


Figure A.2: Concentrations of the dissolved DNA oligos ordered from IDT.

	Concentration
Xylene cyanol	0.04%
Bromophenol blue	0.04%
Glycerol	20%
Nuclease-free water	

Table A.3: Contents of native loading buffer.

	Concentration
Xylene cyanol	0.04%
Bromophenol blue	0.04%
Glycerol	20%
Urea	5.4 M

Table A.4: Contents of denaturing loading buffer.

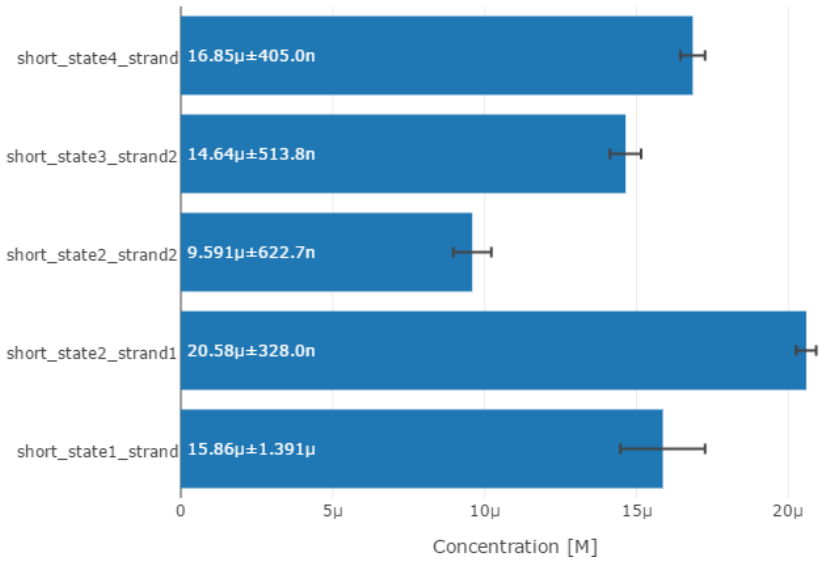
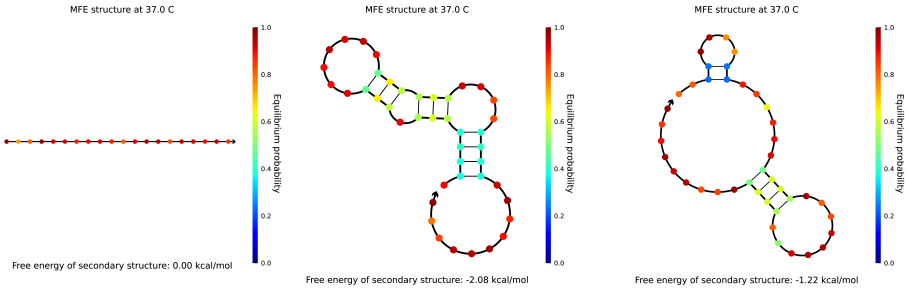


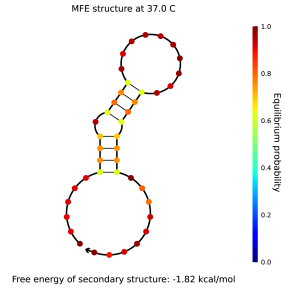
Figure A.3: Concentrations of the short translator RNA sequences.



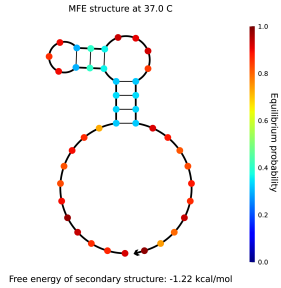
(a) T7 promoter

(b) Short 1

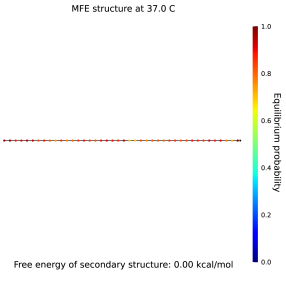
(c) Short 2



(d) Short 3

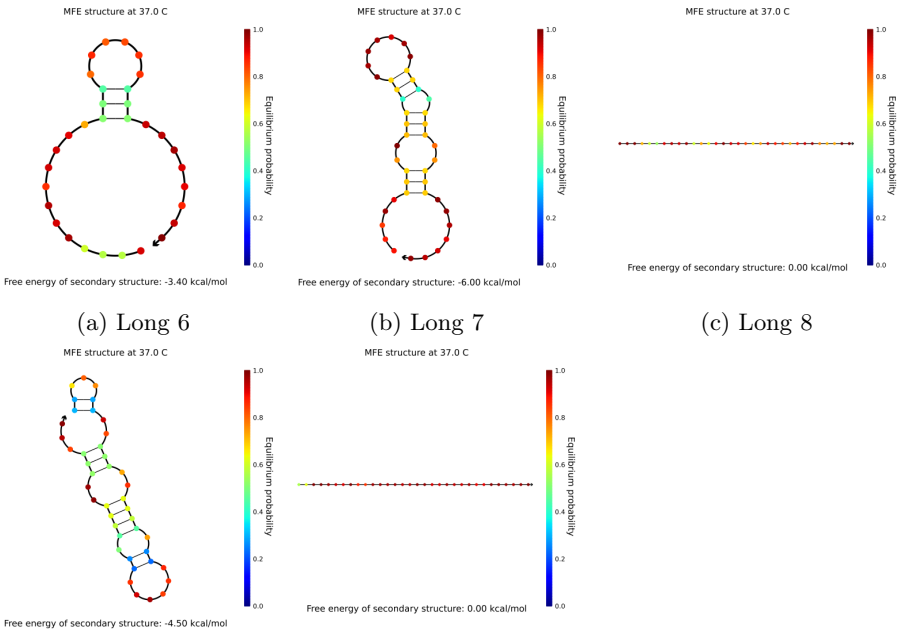


(e) Short 4



(f) Short 5

Figure A.4: Secondary structures of the DNA sequences for the short translator and the T7 promoter sequence.



(a) Long 6 (b) Long 7 (c) Long 8

(d) Long 9 (e) Long 10

Figure A.5: Secondary structures of the RNA sequences for the long translator.