Leak Reaction Case Study - The Seesaw Circuit Architecture

Mrinank Sharma

August 31, 2017

1 Introduction

In this case study, we simulate some of the key reactions from the Seesaw circuit architecture, devised by Qian and Winfree [1]. This circuit architecture allows the implementation of digital logic gates, catalysis cycles and more. We also simulate leak reactions - unintended reactions which result in spurious output production and hence compromise the reliability of such circuits.

2 Background Information

2.1 Basic Definitions

The seesaw circuit architecture is a composable circuit architecture based upon the principle of toehold exchange, named so due to its back and forth motion. The seesaw circuit architecture is defined by the following [1]:

1. The *wire*, defined by a two *recognition* domains (15 nucleotide long domains to prevent undesired strand displacement) and a toehold.
2. The *gate*, defined by a strand with a recognition domain flanked by two toeholds. The gate is always bound to a wire. A gate complex is denoted as a gate:wire complex where wire is the wire bound to the gate.
3. The *threshold*, a duplex which serves to consume all of the input wire strand until the threshold level is met. As a result, small amounts of input strand present when unintended can be tolerate (i.e. increases robustness to leak).
4. The *universal toehold* - each toehold in this architecture has the same toehold domain. As a result, it is the recognition domains which control whether a certain strand displacement reaction will occur rather than matching toeholds.

These features are abstracted as may be seen in Fig. 2, copied from [1].

2.2 Intended Reactions

There are several intended reactions within the Seesaw Circuit Architecture. They include:

- Output Production: An input wire and a gate:output complex react to form an output wire and a gate:input complex. Note that this is a toehold exchange reaction and as such the reaction may proceed in either direction.
- Fuel Input Regeneration: A gate:input complex reacts with a fuel wire to release an input wire and form a gate:fuel complex. The gate:fuel complex has no purpose within the architecture. As above, this reaction may proceed in either direction.
- Thresholding: An input wire reacts with a threshold complex in a fast, irreversibly manner causing the input to be consumed. This increases the reliability of the circuit.

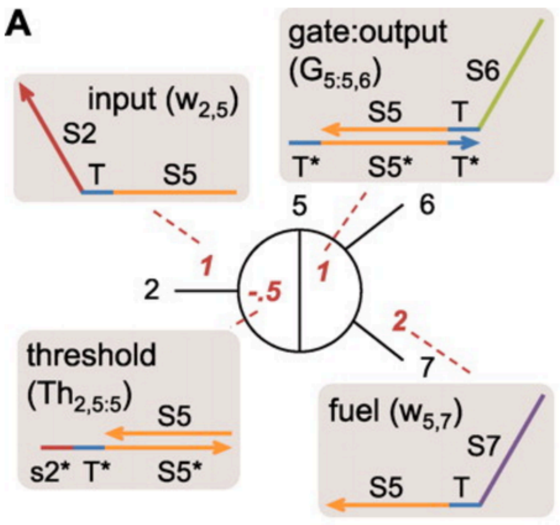

Figure 1: Abstraction using to representing seesaw networks. Positive numbers indicate concentrations whilst negative numbers indicate threshold concentrations. Taken from [1].

2.3 Leak Reactions

In addition to the reactions listed above, there also exist leak reactions which are unintended and undesired. An example of a leak reaction would be the release of output strand in the absence of input strand (see below for more details). These occur at rates which are several orders of magnitude slower compared to the intended reactions but even though there exist thresholds in order to mitigate the effects of leak, given enough time, the thresholds of the gates will be depleted causing erroneous results.

Common leak reactions which occur in seesaw circuits include but are not limited to the following [2] :

1. Gate - Gate Leak : zero toehold strand displacement can occur between two gates (e.g. due to polarity of complementarity) resulting in strand displacement. The mechanism by which this reaction occurs is not particularly well understood and it hypothesized that blunt end stacking has a significant contribution to this leak. A solution to this problem exists in the form of clamp domains. [1]

2. Fuel - Gate leak: unintended strand displacement also occurs between a gate and its respective fuel in which a fuel wire reacts directly with the gate:output complex, forming the output wire without input wire. See Fig. 4.

3. Gate Crosstalk: for gates with multiple inputs, each wire shares a recognition domain with the gate. As a result, it is possible that the input wire can bind to the threshold intended for a different input. This can be remedied using design constraints.

4. Threshold inhibition: a fuel wire may bind to a threshold molecule rendering it inactive, increasing the probability that an input wire activates the gate rather than being thresholded. Similar to gate-crosstalk, this can be remedied by adding design constraints.

The occurrence of leak reactions can be attributed to dynamic nature of DNA complexes; in reality, once a base pair has formed, there is a probability that it will disassociate even though this is energetically unfavorable. The dissociation of terminal base pairs is known as fraying and studies have indicated that the terminal base pair can be 50% open at room temperature with the penultimate base open around 10-20% given no dangling ends. [3]. For instance, take a typical gate complex. Fig. 5 outlines the base pair formation probabilities individually for each base pair, generated using the NUPACK web application [4]. The base pairs are less open than earlier suggested due to the stabilizing contributions from the dangling ends. Regardless, this fraying action exposes base pairs, causing undesired reaction pathways i.e. leak.

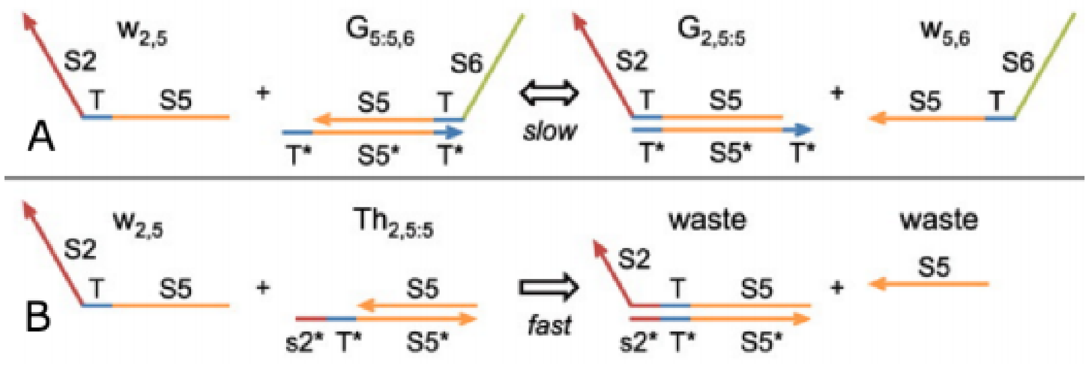

Figure 2: Fundamental reactions within the seesaw architecture. Reproduced from [1]. (A) The seesawing reaction in which toehold exchange occurs with the input wire displacing the output wire from the gate:output duplex. This is slow and reversible. (B) The thresholding reaction in which the input wire is consumed. This is a quick reaction as it results in a reduction in the free energy of the system (net increase in the number of base pairs).

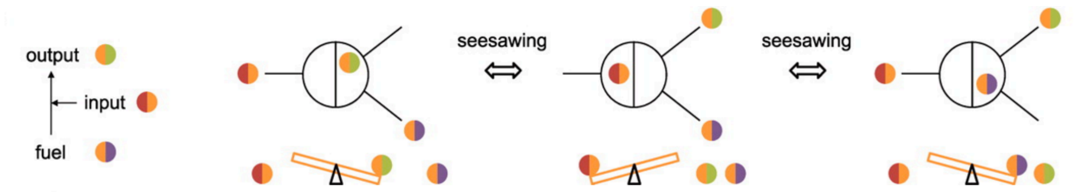

Figure 3: Seesaw architecture 'seesawing'. When a wire (represented by two colors, denoting its two recognition domains) is in the gate, that wire is bound to the gate. As the diagram indicates, when the input wire 'jumps' onto the 'seesaw' in the first reaction, the output is released. The fuel wire can then 'jump' onto the 'seesaw', regenerating the input and allowing that wire to react with a different gate. Reproduced from [1]

3 Case Study Files

3.1 Simulation Method

The simulations will be carried out using *first step mode*. The simulator generates a Boltzmann sample (using NUPACK) for each of the starting complexes and creates a virtual box containing one molecule of each of the starting complexes. The simulator then triggers an initial base pair formation between the two complexes and then watches for stop conditions to be triggered (e.g. a condition set which could indicate the reaction has failed) and will simulate until either a stop condition has been met or the maximum reaction time has been reached.

The initial complexes, starting states and stop conditions are all set by the experiment type and configured automatically for ease of use.

NB: Supersampling is enabled. Here, whilst we retain Boltzmann sampling we re-use each sample multiple times (25). This is to reduce the time required to sample the initial complexes as this requires significant computation time. This may have unintended consequences in result biasing.

3.2 Case2.py

3.2.1 What we are looking at...

This case study file investigates several reactions of the Seesaw Circuit Architectures. The following reactions are investigated:

- The productive reaction which results in the release of an output wire and a gate:input molecule in the presence of input and a gate:output molecule.

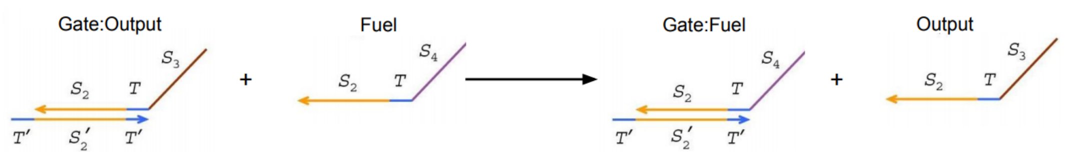

Figure 4: Gate-Fuel - strand displacement occurring leading to unintended output . Modified from [2]

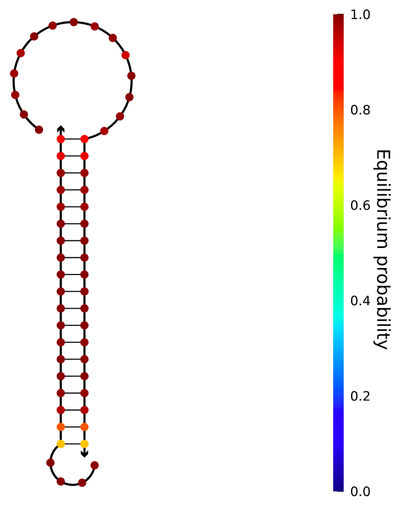

Figure 5: Base pair formation probabilities for a typical gate complex. Generated using the NUPACK web application. [4]

- The productive reaction which results in the release of an input wire and a gate:fuel molecule in the presence of a fuel wire and a gate:input molecule.
- The leak reaction known as gate-gate leak, in which two gate:output molecules react and result in the spurious release of a output wire.
- The leak reaction known as gate-fuel leak, in which a gate:output molecule reacts with its fuel wire, resulting in the release of an output wire.

3.2.2 Using this file

The main method calls `runSimulations(1000)` which is used to run a number of simulations, incrementing 1000 trials until *enough successful trials are observed*.

Run simulations is defined as follows

```
def runSimulations(trialsIn=1000):
    # Uncomment for automatic logging
    # myMultistrand.setOutputFile("case2")
    # Here we say we are going to use 2 threads, storing only succesful data.
    # We will require at least 2 succesful trials with a maximum number of trials of 2500000
    # We will bootstrap, re-sampling our results 1000 times.
    setupSimulationOptions(2, True, 2, 2.5e6, 1000)

    # Here we create two clamped seesaw gates, according to the defined interface.
    # These are typical sequence designs, taken from published papers
    # whose lengths are reasonable for the clamped seesaw circuit architecture.
    gateA = ClampedSeesawGate(*CL_LONG_GATE_A_SEQ)
    gateB = ClampedSeesawGate(*CL_LONG_GATE_B_SEQ)

    # In order to run different reactions, use a different enum for the experiment type!
    runExperiment(trialsIn, gateA, Experiment.GATE_OUTPUT_PRODUCTION)
```

In order to run different experiments, the `runExperiment(args)` can be modified. For instance, in order to run the gate fuel regeneration reaction followed by a gate gate leak experiment, we can modify run simulations to the following.

```
def runSimulations(trialsIn=1000):
    # Uncomment for automatic logging
    # myMultistrand.setOutputFile("case2")

    # Here we say we are going to use 2 threads, storing only succesful data.
    # We will require at least 2 successful trials with a maximum number of trials of 2500000
    # We will bootstrap, re-sampling our results 1000 times.
    setupSimulationOptions(2, True, 2, 2.5e6, 1000)

    # Here we create two clamped seesaw gates, according to the defined interface.
    # These are typical sequence designs, taken from published papers
    # whose lengths are reasonable for the clamped seesaw circuit architecture.
    gateA = ClampedSeesawGate(*CL_LONG_GATE_A_SEQ)
    gateB = ClampedSeesawGate(*CL_LONG_GATE_B_SEQ)

    # In order to run different reactions, use a different enum for the experiment type!
    runExperiment(trialsIn, gateA, Experiment.GATE_FUEL_REGEN)
    runExperiment(trialsIn, gateA, Experiment.GATE_GATE_LEAK, 25, gateB)
```

The results for each experiment will be outputted to the console and logged to `data.txt`. If `myMultistrand.setOutputFile("case2")` is uncommented, the results for the batch of experiments will be stored in a text file which also includes a timestamp e.g. `case2-25-08-17+15-32-12`.

3.3 Case3.py

3.3.1 What we are looking at...

In this case study, we will take a look at the effect of changing the amount of super-sampling on the time taken per trial for an experiment. This case study reuses a significant amount of code and provided that `matplotlib` is installed, will automatically produce a graph.

The reaction we have chosen here is the productive reaction which results in the release of an output wire. This is a intended reaction for the seesaw circuit architecture and as such these simulations should not take a significant amount of time to compute.

3.3.2 Using this file

In order to use this file, the main method can be run and a graph producing the desired plot will be produced automatically. We use the same interface as in `case2.py`.

3.3.3 Comments

In this example, we calculate the time taken per trial which is perhaps not the best measure of time. Due to the stochastic nature of chemical reaction networks, any straightforward measure will be limited when there are *"not enough"* reactions - perhaps increasing the minimum number of successful trials will produce different results (but take longer of course).

There is clearly a limit to the number to which it is reasonable to super sample or not. For instance, if terminating after 1 successful reaction it is not acceptable to super-sample 10^8. It is hard to formulate reasonable bounds for the amount of super-sampling.

4 Closing Comments

These case study files have been designed to be as simple to use as possible. If you have any questions regarding the use of these case study files, please do not hesitate to contact me at `ms2314@cam.ac.uk`

Mrinank Sharma,
September 2017

References

[1] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades.

[2] Lulu Qian and Erik Winfree. A simple DNA gate motif for synthesizing large-scale circuits. *J. R. Soc. Interface*, 8:1281–1297, 2011.

[3] X. Olson, S. Kotani, J.E. Padilla, N. Hallstrom, S. Goltry, J. Lee, B. Yurke, W.L. Hughes, and E. Graugnard. Availability: A metric for nucleic acid strand displacement systems. *ACS Synth. Biol.*, 6:84–93, 2017.

[4] J. N. Zadeh, C. D. Steenberg, J. S. Bois, B. R. Wolfe, M. B. Pierce, A. R. Khan, R. M. Dirks, and N. A. Pierce. Nupack: analysis and design of nucleic acid systems. *J. Comput. Chem.*, 32:170–173, 2011.