DNA Storage and Computation

Zoe Dormuth
Electrical Engineering Master's Plan C Project

Abstract

This document explores a form of encoding binary data in DNA using nicks and DNA strand displacement to applications in logic gates, quantum computing and associative memory. A logical AND operation implemented in DNA is analyzed and simulated using Visual DSD. A model for in-memory computing with DNA is applied to the Toffoli quantum gate. A synthesis of previous research about implementations of associative memory in DNA is performed. Encoding data with nicks is applied to a previous implementation of DNA associative memory. A linear threshold circuit that implements a Hopfield neural network is studied and the fundamental building blocks of the DNA reactions are simulated using Visual DSD.

1. Introduction

This project is part of research studying DNA computation and storage being done at the University of Minnesota, University of Illinois at Urbana Champaign (UIUC) and University of Texas - Austin. The University of Minnesota group is exploring how DNA can be used to perform computations and store data. UIUC and UT Austin are finding new techniques to encode data in DNA and performing experiments in a laboratory setting. The rise in the amount of data we can collect has led to research different ways to store and compute data. The benefits of using DNA as an avenue for storage and computation is the high information density, small area and easy access to DNA resources. This research focuses on encoding and storing data in DNA with an emphasis on using native DNA versus synthesized DNA. Native DNA is naturally occurring DNA found in living organisms. The experiments performed by the research groups in Texas and Illinois are using E. coli DNA as native DNA. Synthesized DNA is the result of artificial creating of DNA molecules. This is costly and not as readily available compared to native DNA. A technique to encode binary bits in native DNA using nicks has been discovered and will be applied to previous research using synthesized DNA sequences. Performing computations on data stored in DNA is also being studied. Simple computational concepts such as Boolean logic gates, circuits and fundamental computer science concepts are being implemented using DNA and the tools of molecular biology.

1.0.1. DNA Strand Displacement

DNA strand displacement is an important tool in molecular biology where one strand of DNA is exchanged with another strand of DNA through branch migration. Hybridization between complementary DNA strands based on the Watson-Crick base pairings (A, C, T and G's) is the fundamental concept of DNA strand displacement. DNA strand displacement occurs when there is exists a double stranded DNA with a toehold. A toehold is a gap in the top strand of the double stranded DNA exposing the base pairs on the bottom strand seen in Figure 1. The input to a DNA strand displacement reaction is a single strand of DNA that is complementary to the base pairs exposed by a toehold on the double stranded DNA. If the single stranded DNA is also complementary to the base pairs in the double stranded DNA next to the exposed toe hold, branch migration will occur. Branch migration is when a single stranded DNA binds to the complementary base pairs on a double stranded DNA and displaces the top strand of the double stranded DNA seen in Figure 1. The output is a double stranded DNA that has the input strand bound to the bottom strand. The top strand that was originally bound to the double stranded DNA is now a single strand of DNA ready to perform DNA strand displacement on another double strand of DNA with an exposed toehold. These reactions will continue as long as there are exposed toeholds with complementary base pairings to single stranded DNA. When there are no exposed toeholds as seen in the output in Figure 1, the double stranded DNA is stable and no further reactions can take place. The presence and absence of toeholds play a key part in whether a reaction is a forward-only reaction or a reversible reaction.

1.0.2. Nicking

A double stranded DNA has nucleotide bases that are bonded together to create a stable DNA structure. A nick is a cut in one of the strands in the double stranded DNA between two base pairs. A nick can be placed anywhere in

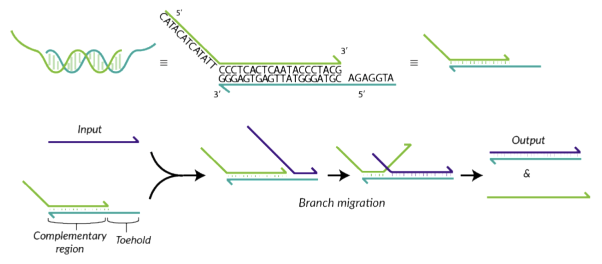


Figure 1. Toehold and DNA Strand Displacement.

the double stranded DNA. When a double stranded DNA is nicked, it breaks the phosphodiesterase bond between 2 adjacent nucleotides. This causes a discontinuity in the double stranded DNA. The nicks in the double stranded DNA will not affect the integrity of the bonds or DNA strand unless heat is applied. When multiple nicks are placed close together (3-5 base pairs in length) and the double stranded DNA sequence is heated up, the bonds between base pairs where the nicks are located will be broken and a section of the double stranded DNA will come off (Wang et al.). This creates toeholds in the double stranded DNA as shown in Figure 1. These toeholds will be used to perform DNA strand displacement.

Nicking is used to encode binary bits (1's and 0's) in a double stranded DNA. A bit will encompass a defined number of base pairs on the double stranded DNA. For example, 6 base pairs will encode a bit of information. A 1 can be encoded if the first two base pairs out of the 6 total base pairs are exposed toeholds. This would be done by placing 3 nicks in the double stranded DNA (Wang et al.). The three nicks would be placed before the first base pair, before the second base pair and before the third base pair. When heat is applied to the double stranded DNA the nicks will not hold the top strand together and the base pairs will unbind from the bottom strand of DNA creating an opening in the double stranded DNA, a toehold. A zero can be encoded in a similar way where two toeholds would be placed at the end of the 6 base pairs instead of the beginning (Wang et al.).

Once a double stranded DNA has been nicked, reading the binary bits encoded in the DNA requires heat and reporter strands. Heat is applied to break the bonds where nicks are located and expose the toeholds. Reporter strands are single stranded DNA that are synthesized to match specific sequences in the double stranded DNA. Each bit location will have its own, unique reporter strand. Based on where the toeholds are located in that bit location, the reporter strand will bind to a front toehold or a back toehold if the bit is 1 or 0, respectively. This causes a DNA strand displacement reaction to occur. The strand originally bound to the double stranded DNA will be released and the reporter strand will bind to the double stranded DNA (Wang et al.). This erases the bit stored at that location so memory is corrupted after a read unless the released strands are reintroduced to the solution at a later point. The strand that was released will be sequenced and knowledge about the sequence at each bit location must be known to identify if the strand is a 1 or 0.

1.0.3. Visual DSD

Visual DSD (DNA Strand Displacement) is a web-based interface tool that allows users to code different DNA reactions without having to manually construct the reaction network by hand. It implements a domain-specific language to allow different encodings of the species in a DNA strand displacement reaction (Lakin et al., 2011). Different levels of abstraction are available to allow for low-level detailed views and higher level simplified views (Lakin et al., 2011). Visual DSD has a stochastic and deterministic simulation tool as well as a chemical reaction network (CRN) that makes it easy to visualize and understand the reactions occurring between DNA strands. DNA logic gates and systems involving low-level species populations have been designed and analyzed with Visual DSD (Lakin et al., 2011).

2. AND Gate

2.1. Introduction

DNA is used to build logic gates that use binary bits as inputs and outputs. The AND gate was built by the UT Austin group where binary bits, 1's and 0's, are encoded in a double strand of DNA. The bits are read at specific addresses, an AND operation is performed and the result is written to another location in the double strand of DNA. An AND gate takes in a two inputs and outputs a 1 if both inputs are 1, otherwise, it outputs a 0. An example of reading, writing and performing the AND operation are shown in the example below.

2.2. AND Gate with Addressing

A double strand of DNA is used to encode a string of 1's and 0's. Each bit is represented by 6 base pairings. Each base pair can be considered an address. A bit is represented as 1 by nicking the top strand of the double stranded DNA before the first base pair, before the second base pair and before the third base pair seen in Figure 2. A bit is represented as a 0 by nicking before the fourth base pair, before the fifth base pair and before the sixth base pair seen in Figure 2. This procedure is repeated to encode a string of bits along a double stranded DNA. When heat is applied to the double stranded DNA, the first two base pairs out of six will be exposed where a 1 was encoded and the last two base pairs out of six will be exposed where a 0 was encoded.

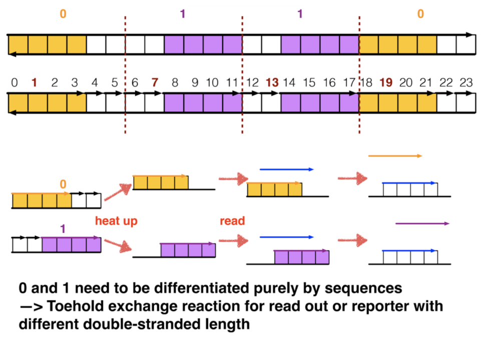

0
1
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
heat up
read
0 and 1 need to be differentiated purely by sequences
—> Toehold exchange reaction for read out or reporter with different double-stranded length

Figure 2. Encode and read binary bits at specific locations.

2.2.1. Encoding a String of Bits

A bit can be read as a 0 or 1 by supplying a reporter strand. The reporter strand is unique to each location of each bit meaning the A, T, C, and G pairing will be ordered specifically for each location. The reporter strand for addresses 0 through 5 will be different than the reporter strand for addresses 18 through 23. The reporter strand for each bit is complementary to the base pairing of the middle 4 addresses out of the total six addresses available. This allows for the reporter strand to bind to an exposed toe hold for a bit encoded as a 0 or 1.

If a zero is encoded at a specific bit location, the reporter strand will bind to the fifth base pairing out of six because it is an exposed toe hold. DNA strand displacement will occur and the reporter strand will displace the strand at the first 4 addresses and bind to the middle four base pairs that are complementary. If a one is encoded at a specific bit location, the reporter strand will bind to the second base pairing out of six because it is an exposed toe hold that is complementary to the reporter strand base pair. The reporter strand will bind to that address and perform DNA strand displacement. The strand at the third through the sixth base pairings will displace from the double stranded DNA and the reporter strand will bind to the second through the fifth base pairings at the specific bit location because it is complementary to them.

After a read, the bit location will no longer encode a 0 or 1 once the reporter strand performs DNA strand displacement. The bit location will have two exposed toe holds, one at the first base pair and one at the sixth base pair. This does not follow the encoding of a 0 having two exposed toe holds at the last two base pairs of the bit location and a 1 having two exposed toe holds at the first two base pairs of the bit location. Therefore, the data is lost when a reporter strand is introduced at the specific bit location. If another reporter strand for that specific bit location is introduced, it would not bind to that location since there is not an exposed toe hold that is complementary to that reporter strand and DNA strand displacement would not occur.

The example in Figure 3 shows a string of bits, 01100, that are encoded in a double strand of DNA using the nicking scheme to nick the last two base pairs of out six for a 0 and the first two base pairs out of six for a 1. There are 30 addresses (base pairs) in the double stranded DNA which is also called a deck. Each bit is allocated six addresses. The first bit in the string 01100 is allocated addresses 0 through 5, the second bit is allocated addresses 6 through 11 and so forth. The double stranded DNA is heated up to expose the toeholds where the top strand was nicked.

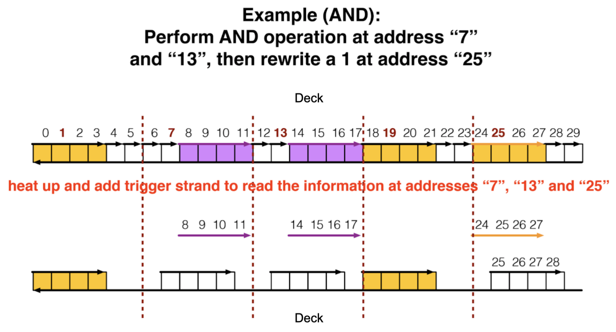

Example (AND):
Perform AND operation at address "7" and "13", then rewrite a 1 at address "25"
Deck
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
heat up and add trigger strand to read the information at addresses "7", "13" and "25"
8 9 10 11
14 15 16 17
24 25 26 27
25 26 27 28
Deck

Figure 3. Read bits at addresses 7 and 13 and write result to address 25.

2.2.2. Reading a String of Bits

The goal is to read the bits at addresses 7 and 13 and perform the AND operation on those bits. The result is written at address 25. Two reporter strands are supplied to read the bits at address 7 or the second bit of the string 01100 and address 13 or the third bit of the string 01100. The reporter strand to read the bit at address 7 is complementary to the base pairs at address 7 through 10. The bit at address 7 is a 1 and the reporter strand will bind to the open toehold at address 7 and displace the strand at addresses 8 through 11. The reporter strand will then bind to the addresses 8, 9 and 10. This will leave the second bit location with two exposed toe-holds, one at address 6 and one at address 11. The output of the read is the strand complementary to the base pairs at addresses 8 through 11. The second reporter strand will perform the same procedure at address 13. The reporter strand is complementary to the base pairs at addresses 13 through 16. The third bit is a 1 and the reporter strand will bind to the exposed toehold at address 13 and displace the strand complementary to base pairs 14 through 17. The reporter strand will then bind to the double stranded DNA at addresses 13 through 16. The third bit location will have two exposed toeholds, one at address 12 and one at address 17. This location no longer holds a 0 or 1. The output of the read will be the strand with complementary base pairs to 14 through 17.

The result of the AND operation for the input bits at addresses 7 and 13 will be written to the fifth bit of the string 01100 at address 25. A reporter strand must be bound to the location of the fifth bit at address 25 to write the result at that location. If the result of the AND gate is 1 the result strand will bind to the open toehold at address 29 and displace the reporter strand bound to addresses 25 through 28. The result strand would encode a 1 at the fifth bit location by binding to addresses 26 through 29 after DNA strand displacement. Otherwise, the result would encode a 0 at the fifth bit location by binding to the exposed toehold at address 24 and displacing the reporter strand bound to addresses 25 through 28. The result strand would bind to addresses 24 through 27 after DNA strand displacement.

A reporter strand that is complementary to the base pairs at addresses 25 through 28 is introduced to the deck. The reporter strand binds to the exposed toehold at address 28 and displaces the strand complementary to the base pairs at addresses 24 through 27. The reporter strand then binds to addresses 25 through 28 and two exposed toeholds are at the fifth bit location at addresses 24 and 29. The lower diagram in Figure 3 shows the results after reading at addresses 7, 13 and 25.

2.2.3. Performing the AND Operation

An AND gate is constructed using a double stranded DNA with two strands separated by a nick on the top and an exposed toe hold at its first base pair seen in Figure 4. The two strands are four base pairs long and six base pairs long. The strand that is six base pairs long has three base pairs that are not bound to the double stranded DNA. The second strand will be displaced if both inputs are encoded as 1s. The first strand will be displaced if either of the input strands is encoded as 1. In this example, the exposed toe hold on the AND gate has to be complementary to the first inputs base pair at address 8. The strand on the AND gate that is four base pairs long is complementary to addresses numbed 9 through 11 on input 1 and complementary to the first address of the second input numbed address 14.

The first input strand (addresses 8 through 11) will bind to the exposed toe hold at the first base pair of the AND gate. The first input strand is complementary to the first four base pairs of the AND gate. The first input strand will bind to the AND gate at the first four base pairs. The result of the DNA strand displacement reaction is an exposed toe hold at the fifth base pair of the AND gate seen in Figure 4. The top strand of the AND gate that is four base pairs long was displaced by DNA strand displacement. The exposed toehold is complementary to the first base pair, numbered address 14, of the second input strand (addresses 14 through 17). The second input strand will bind to the exposed toehold and displace the second top strand on the AND gate. The second input strand is complementary to the fifth through the eighth base pairs on the AND gate. This allows the DNA strand displacement to occur and for the second input strand to bind to the AND gate. The second top strand of the AND gate is released as a single strand.

The result of the AND operation replaces the AND gate with a double stranded DNA with no toeholds. The double stranded DNA is stable because there are no exposed toeholds. It is waste product because no further reactions can occur. The second top strand that was released from the AND gate and is now a single stranded DNA. This is the output of the AND gate reactions. This strand was sequenced to have six base pairs in which the last two base pairs, colored in purple, are complementary to addresses 26 and 27 on the deck. The first four base pairs, colored in green, are complementary to the reporter gates first three base pairs seen in Figure 4.

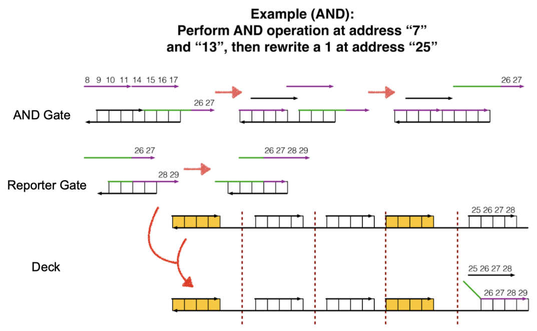


Figure 4. AND operation on DNA strands.

2.2.4. The Reporter Gate

The reporter gate is a double stranded DNA that was constructed to release an output that will write a 1 to address 25 on the deck if the correct output from the AND gate is released. The output from the AND gate cannot directly write a 1 to address 25 on the deck, however, it is the correct input to the reporter gate that will release the strand that will write a 1 at address 25 on the deck. If the top strand that is six base pairs long was designed to write a 1 to the address 25, the second input strand would not be complementary to the double stranded DNA. It would not bind to the exposed toehold and release the top strand that is six base pairs long. The reporter gate was constructed and the output of the AND gate will be the input strand to the reporter gate to output the strand to write a 1 at address 25 in the deck.

The reporter gate is constructed to have an exposed toehold that is complementary to the second base pair on the 6 base pair output strand from the AND gate. The second and third base pairs on the reporter gate are complementary to the third and fourth base pairs on the AND gate output strand. The third through the sixth base pairs on the reporter gate are complementary to the base pairs at addresses 26 through 27. The top strand on the reporter gate has the same base pairs as the third and fourth base pairs of the AND gate output strand. It also has base pairs complementary to addresses 26 through 29 of the deck. This strand will be displaced through DNA strand displacement.

The AND gate output strand will bind to the exposed toehold and displace the strand that is bound to the reporter gate using DNA strand displacement. The result is a double stranded DNA with a base pair that is not bound to the reporter gate but attached to the top strand of the double stranded DNA reporter gate. The output strand from the reporter gate is a 6 base pair long strand that is has its last four base pairs complementary to addresses 26 through 29 and the first two base pairs not complementary to addresses 24 and 25.

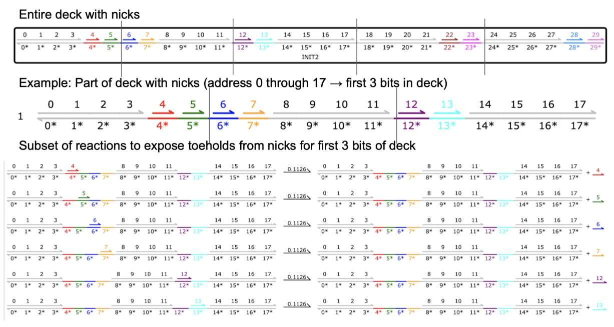
2.2.5. Writing to a String of Bits

The output from the reporter strand is able to write to the deck. The current deck has the reporter strand bound to addresses 25 through 28 at the fifth bit location. There are exposed toeholds at addresses 24 and 29. The output strand from the reporter gate will bind to the exposed toehold at address 29 and displace the reporter strand through DNA strand displacement since the output strand is complementary to addresses 26 through 29. The first two base pairs on the output strand from the reporter gate will not bind to addresses 24 and 25 since they are not complementary to those base pairs. These base pairs will act as an overhang.

The encoding at the fifth bit or address 25 is now a 1 since there are two exposed toeholds at the beginning of the bit location and a strand of 4 base pairs at the end of the location. The final encoding of the deck will 0XX11. The second and third bits are not encoded as a 0 or a 1 since the reporter strands are still bound to those locations and do not read a 1 or 0. The last bit switched from being a 0 to a 1 since the second and third bits were 1s so a 1 was written to the last bit of the string to reflect the AND gate operation.

2.3. Simulation with Visual DSD

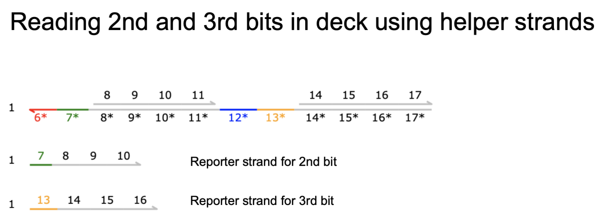
The AND gate example seen in Figures 2-4 was simulated using Visual DSD to better understand the reactions. Initially the deck was encoded as five bits, 01100, seen in Figure 5. Six base pairs are used to encode one bit of information. Nicks were placed around the last two base pairs to encode a 0 and around the first two base pairs to encode a 1. The reactions for exposing the toeholds by applying heat were shown for the first three bits of the deck, addresses 0 through 17. The reactions show that each single strand that will expose a toehold is released separately. The products of the reaction have exposed toeholds at addresses 4 and 5, 6 and 7 and 12 and 13 to encode the bits 011.



Entire deck with nicks
Example: Part of deck with nicks (address 0 through 17 → first 3 bits in deck)
Subset of reactions to expose toeholds from nicks for first 3 bits of deck

Figure 5. Encoding a string of bits into the deck.

The initial strands for reading the bits at addresses 7 and 13 are seen in Figure 6. The reporter strands have the same addresses for the middle four addresses out of the six addresses used to encode a bit. Address 7 on the reporter strand for the second bit will bind to the 7* exposed toehold on the deck. Address 13 on the reporter strand for the third bit will bind to the 13* exposed toehold on the deck.



Reading 2nd and 3rd bits in deck using helper strands
Reporter strand for 2nd bit
Reporter strand for 3rd bit

Figure 6. Reading bits at addresses 7 and 13.

The reactions that occur when reading the bits at addresses 7 and 13 are shown in Figure 7. The first and second reactions show the first reporter strand binding to the 7* exposed toehold and displacing the strand covering addresses 8, 9, 10 and 11. The reactions show that this is done in two steps since the output of the initial bind does not release the 8, 9, 10 and 11 strand right away. These reactions are reversible since there are two arrows pointing opposite directions. The output of this read is a strand with addresses 8, 9, 10 and 11 and a double stranded DNA with an exposed toehold at address 11. The third through the fifth reactions in Figure 7 show both helper strands reading from the deck at the same time. The outputs are the 8, 9, 10 and 11 address strand and the 14, 15, 16, and 17 address strand which are the strands that encode the bits used for the AND gate operation. The deck is also an output that can have a reverse reaction to restore the data encoded in the deck if those strands are not used in a later reaction.

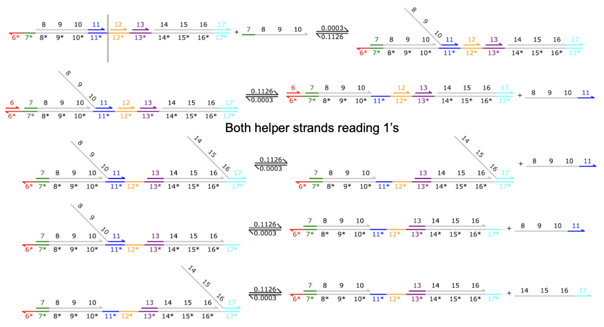


Figure 7. Reactions from reporter strands reading bits at addresses 7 and 13.

The AND gate and reporter gate operations are simulated in Figure 8. The initial strands are a double stranded reporter gate, two inputs that were read in Figure 7 and the double stranded AND gate. The first reaction simulates the 8, 9, 10 and 11 address input strand binding to the AND gate at the 8* exposed toehold. The output is the 9, 10, 11, 14 strand being displaced through DNA strand displacement. The second reaction shows the reporter gate and the second input strand with addresses 14, 15, 16 and 17 binding together which is not what we want, however, this reaction is reversible so it will restore to the second input strand and reporter gate for later use. The third reaction is the AND gate from the first reaction after DNA strand displacement occurs. The second input strand with the addresses 14, 15, 16 and 17 binds to the 14* exposed toehold on the AND gate and displace the strand with addresses 15, 16, 17, 18, 26 and 27. This is a forward-only reaction since there is one arrow and the AND gate after the reaction does not have any exposed toeholds. The fourth reaction shows the reverse reaction that occurs due to the first reaction in this figure. The output is the original AND gate and the first input strand. The fifth reaction is the output of the AND gate seen in the third reaction and the reporter gate. The AND gate output strand binds to the 16* exposed toehold and displaces the strand with addresses 17, 18, 26, 27, 28 and 29. This strand will be used to write the result to address 25 in the deck. This reaction is also forward-only because the reporter gate does not have any exposed toeholds after this reaction.

The output strand from the reporter gate is used to write the result from the AND operation to address 25 on the deck. Figure 9 shows the reaction and final result for the deck from the AND operation. The last two bits and the output from the reporter gate seen in Figure 8 are inputs to the reaction. The output strand from the reporter gate binds to

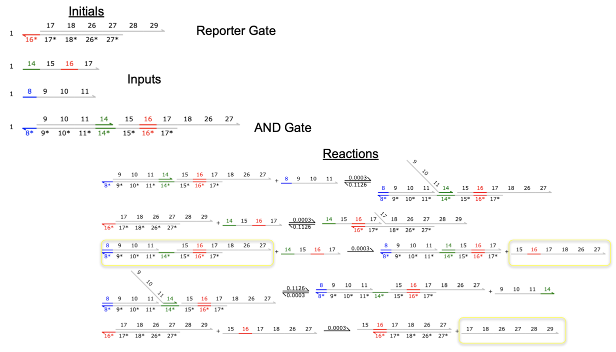


Figure 8. AND and Reporter gate reactions.

the 28* and 29* exposed toeholds on the deck and displace the reporter strand that has addresses 24, 25, 26 and 27. The result (not shown) has the addresses 17 and 18 as overhangs since they are not complementary to addresses 24 and 25. Therefore, the end result has two exposed toehold in the front of the six base pairs used to encode a bit result in a 1 written to address 25. The final result shows the string 0XX01 where the X's are the reporter strands bound to the second and third bit locations where the data was read from the deck to perform the AND operation.

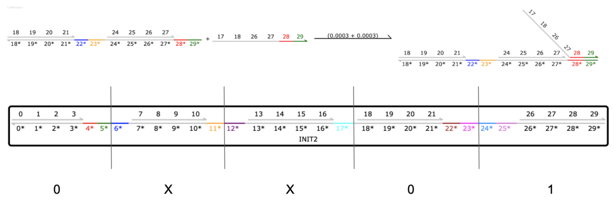


Figure 9. Writing result from AND operation to address 25 in deck.

2.4. Discussion

The motivation behind this example is to show that a string of bits can be encoded in DNA and an AND operation can be performed using specific bits of the bit string. An important observation to note is that the result from the AND operation between 2 specified bits can be written to a specific address or bit location in the string. The AND operation does not need to be performed with two bits in a row in the bit string nor does it need to be written to the bit location directly following the two bits chosen as inputs for the AND operation from the bit string. This allows for flexibility in choosing which data the AND operation should be performed on and where the result is written.

This example was simulated using Visual DSD. The reactions that occur during the AND operation were studied to determine which reactions are forward-only versus reversible. This provides a better understanding about DNA strand displacement. The limitation to Visual DSD in this example was the web-based interface. This example has a long data strand for the number of reactions Visual DSD can simulate due to the number of exposed toeholds. The example was performed in separate steps to understand encoding data, reading data, performing the AND operation and writing the result back to the deck. Ideally this would be simulated in one program, however, each part was performed separately to have the reactions run with the resources of Visual DSD. Future work would be to model this example with sequences that might allow Visual DSD to simulate the entire example in one program.

3. SIMDNA and Quantum Gate

3.1. Introduction

DNA computing has been researched extensively with the potential to be a new avenue of storage and computation for the increasing amount of data. This section explores the idea of building a quantum computer using DNA by introducing a scheme to implement a Toffoli (CCNOT) gate. The Toffoli gate is implemented using an in-memory computation model proposed by the research group at UT Austin in their paper, SIMDNA: Single Instruction, Multiple Data Computation with DNA Strand Displacement Cascades (Wang et al.). SIMDNA proposes an in-memory computation model that is different from previous research in that synthesizing new DNA is not required to transform stored data. This model avoids the time consuming and expensive synthesizing and sequencing steps seen in previous DNA storage schemes (Wang et al.). A sequential procedure is introduced using magnetic beads which allows for an easier transition from electronic concepts to DNA models. Parallelism is achieved which saves computation time. SIMDNA presents two examples, a Rule 110 and a binary counter, to demonstrate their model (Wang et al.).

3.1.1. Quantum Computing

A Toffoli (CCNOT) gate is a universal reversible logic gate. Any circuit that is reversible can be constructed solely using Toffoli gates. It is an important gate in the realm of quantum computing. Quantum computers are built out of a set of universal quantum gates. They are able to compute concepts that classical computers cannot. All Boolean logic gates can also be implemented using the Toffoli gate. This allows for classical logic operations to be performed on a quantum computer.

The motivation of implementing the Toffoli gate using DNA is to eventually scale up and construct a quantum computer using DNA. The Toffoli gate is not enough to construct a universal quantum computer so other quantum gates such as the Fredkin gate would need to be constructed as well. Quantum computing takes qubits as inputs. Qubits can be encoded as DNA to have a probability of 1 or 0 that is determined by the sequence of their base pairings. This is a start to open up an avenue into DNA quantum computing.

3.1.2. How SIMDNA works

The research group at UT Austin has created a model to perform in-memory computations on stored data. They use a domain level abstraction for DNA strands that focus on DNA strand displacement and not the sequences of base pairs (Wang et al.). Data is stored in a multi-stranded complex called a memory register. A memory register holds multiple bits where each bit is stored in a cell that contains a constant number of domains or base pairs (Wang et al.). Bits are encoded by the location of toeholds in each memory cell. Nicking is used to separate strands to create the multi-stranded complex. The bottom strand is synthesized and a magnetic bead is attached to it (Wang et al.). The memory registers encoding different strings of bits are placed in a master solution.

There are three different instruction events that are used to compute the data in-memory. These instruction events are illustrated in Figure 10. Attachment adds a new strand to the pre-existing strands in a memory cell. It does not displace any pre-existing strands. It can partially displace pre-existing strands if both the new strand and the pre-existing strand both have at least two domains bound to the bottom strand (Wang et al.). Displacement adds new strands that detach pre-existing strands through DNA strand displacement (Wang et al.). Detachment removes strands from the memory cell without adding new strands. New strands that are complementary to pre-existing strands are added but do not bind to the memory register. The use the overhang of pre-existing strands as a toehold to bind and remove the pre-existing strand from the memory register (Wang et al.).

The magnetic beads allow for sequential elution operations (Wang et al.). After each instruction event, the master solution is rinsed and the waste products are washed away. The magnetic beads keep each memory register in the master solution and more instructions are able to be performed on each memory register.

3.2. Toffoli Gate

A Toffoli gate was built using the three instruction events seen in Figure 10 (Wang et al.). A total of eight instructions are used to achieve a bit flip of the third input bit if the first and second input bits are 1. Three bits are used to represent the input bits. The output will be represented using three bits after all instructions are executed. The bits are encoded using 5 base pairs. A bit encoded as 1 has an exposed toehold at the first base pair out of five. A bit encoded as 0 has an exposed toehold at the last base pair out of five. The three bits used in the memory registers can be encoded as any input combination in the Toffoli gate truth table seen in Figure 11. If the first two bits are not both 1's, the output will be the same as the input. This scheme demonstrates the DNA strand displacement reactions if the first two bits of the deck are encoded as 1's. The instructions will be performed on a 111 input and a 110 input simultaneously to demonstrate that the same sequence of instructions will work for both inputs. These two memory registers will be placed in a master solution and the strands for each instruction will be added to it and washed away after each instruction is executed.

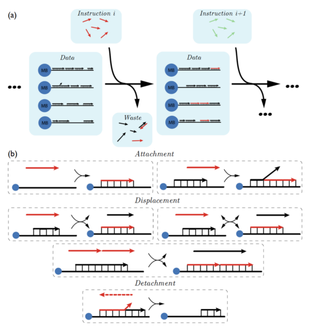
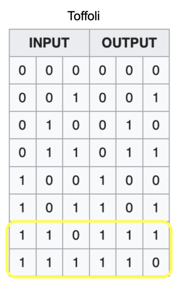


Figure 10. Three operations in SIMDNA paper.



Toffoli

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Figure 11. Truth table for Toffoli gate.

Figure 12 shows the initial strands of DNA for the two cases in the Toffoli gate truth table that will flip the third input bit. The first case is when the input is encoded as 111 and outputs 110, this will be referred to as the 111 memory register. The second case is when the input is 110 and outputs 111, this will be referred to as the 110 memory register.

The first instruction protects the first and second bits that need to remain 1's. A long single stranded DNA that is complementary to the base pairs used to encode the first and second bits is added to the memory vessel seen in Figure 13. It has an overhang that will not bind to the exposed toehold at the third memory cell for the 111 memory register because it is not complementary to that base pair. The first instruction strand is synthesized to match the base pair ordering of the first and second bits, but not the third bit. The reaction from instruction 1 is seen in Figure 14. The strands encoding the first and second bits on both memory registers are displaced by the long input strands.

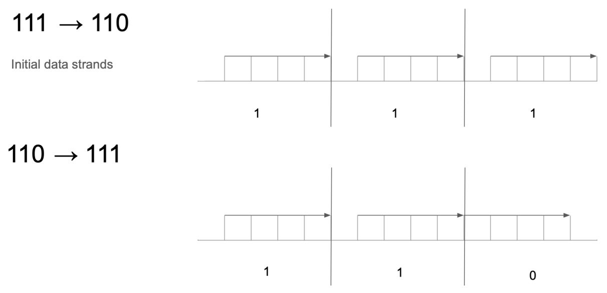


Figure 12. Toffoli gate initial strands.

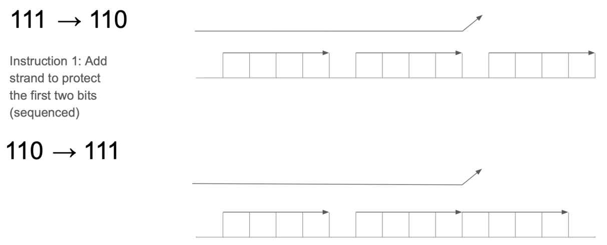

111 → 110
Instruction 1: Add strand to protect the first two bits (sequenced)
110 → 111

Figure 13. Instruction 1.

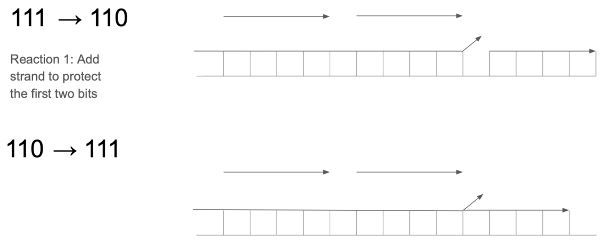


Figure 14. Reaction 1.

The second instruction protects the third bit for the 110 memory register. An input strand that is complementary to the fourth and fifth base pairs of the third bit location is added to the memory register seen in Figure 15. The reaction for the second instruction is seen in Figure 16. The input strand cannot bind with the 111 memory register because there are no exposed toeholds at those base pairs. The input strand binds with the exposed toehold on the 110 memory register and detaches one base pair from the strand that encodes that location as a 0. This occurs because both the input strand and the pre-existing strand still have at least two base pairs bound to the double strand. The pre-existing strand at the third bit location on the 110 memory register now has an overhang. The input strand also has an overhang that will be used as a toehold later.

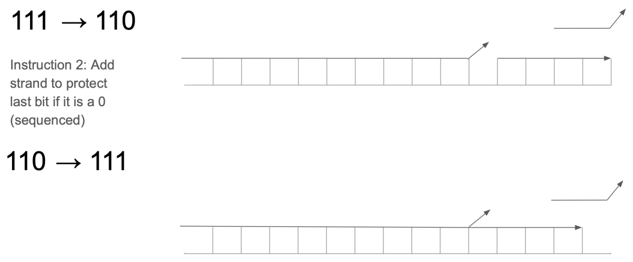

111 → 110
Instruction 2: Add strand to protect last bit if it is a 0 (sequenced)
110 → 111

Figure 15. Instruction 2.

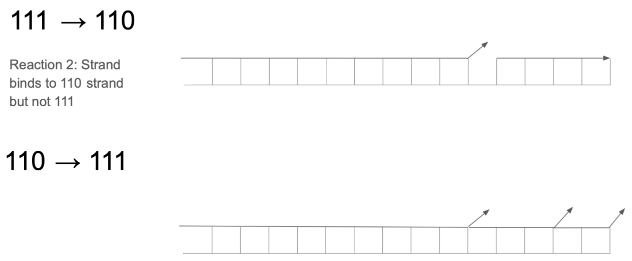


Figure 16. Reaction 2.

The third instruction flips the third bit to 0 if it is encoded as a 1 seen in Figure 17. The only exposed toehold is on the 111 memory register at the third bit location which is encoded as a 1. The input strand at that location will bind to the exposed toehold and displace the strand encoding the bit as a 1. The reaction for instruction 3 is seen in Figure 18. The result of the DNA strand displacement is the third bit on the 111 strand being encoded as a 0 instead of a 1. The 110 memory register and the first two bits of the 111 memory register were not affected by the instruction 3 reaction because there were no exposed toeholds.

The fourth instruction adds a protector strand to protect the third bit of the 111 memory register since it has been flipped to 0 which is the goal. This protector strand is different from the protector strand added in instruction 2 because it is shorter in length. The protector strand is complementary to the last base pair of the third bit locations seen in Figure 19. The reaction for the fourth instruction is seen in Figure 20. The 110 memory register does not have any exposed toeholds and the protector strand will not be able to bind

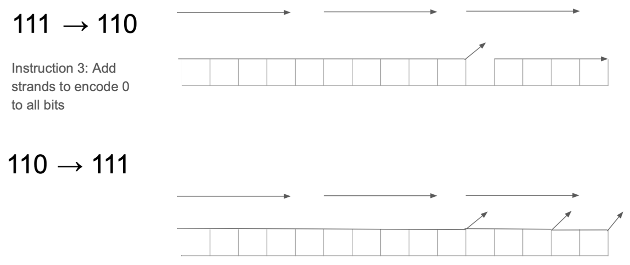


Figure 17. Instruction 3.

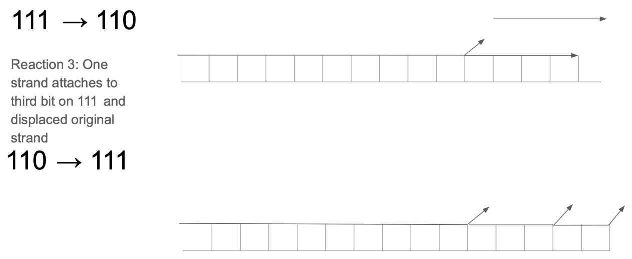

111 → 110
Reaction 3: One strand attaches to third bit on 111 and displaced original strand
110 → 111

Figure 18. Reaction 3.

to it. The protector strand binds to the fifth base pair of the third bit location of the 111 memory register.

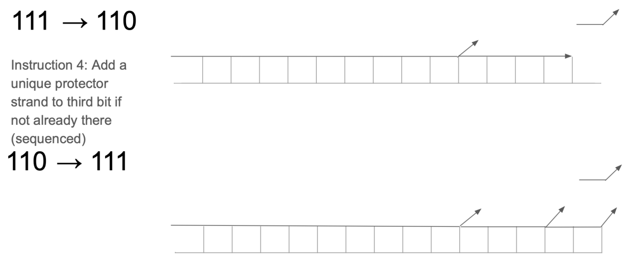

111 → 110
Instruction 4: Add a unique protector strand to third bit if not already there (sequenced)
110 → 111

Figure 19. Instruction 4.

The fifth instruction removes the first protector bit from the 110 memory register. This is the detachment instruction event where the strands added during the procedure can be detached if their complementary strand is added to the memory register. The arrows are faced in a different direction seen in Figure 21. The reaction for instruction 5 is seen in Figure 22. The 111 memory register is not affected by the input strand because it is not complementary to any strands at that location or in the memory register. The 110 memory register has the protector strand bound to the fourth and fifth base pairs of the third bit location removed using the detachment process. The original strand that encodes the third bit location attaches the overhang to the double stranded DNA

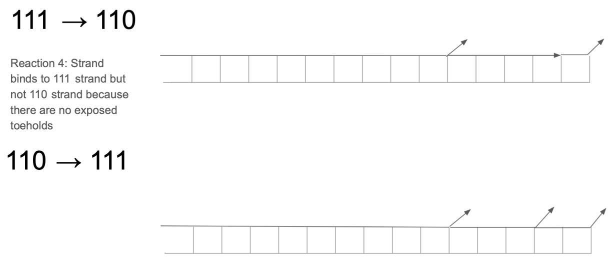

111 → 110
Reaction 4: Strand binds to 111 strand but not 110 strand because there are no exposed toeholds
110 → 111

Figure 20. Reaction 4.

and restores the bit to be encoded as a 0.

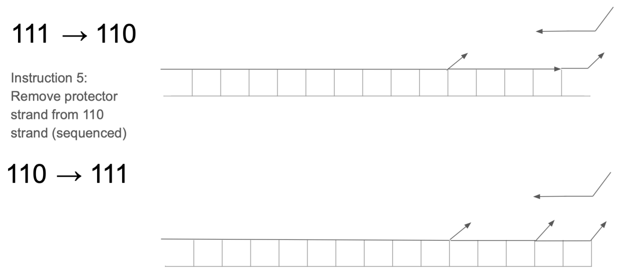

111 → 110
Instruction 5: Remove protector strand from 110 strand (sequenced)
110 → 111

Figure 21. Instruction 5.

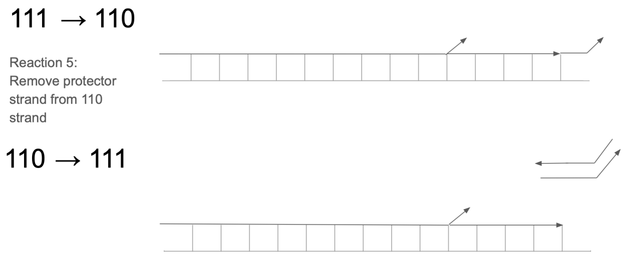


Figure 22. Reaction 5.

The sixth instruction removes the strands that protect the first two bits of both memory registers. The detachment process is used again. Two long strands that are complementary to the input strands added during the first instruction are added in Figure 23. The reaction for instruction 6 is seen in Figure 24. The long input strands displace the long strands bound to the first and second bit locations for both memory registers. This leaves the first ten base pairs exposed.

The seventh instruction adds a 1 to all bit locations on both memory registers. Three input strands are added to each memory register that are complementary to the last four base pairs for each bit location seen in Figure 25. The reaction

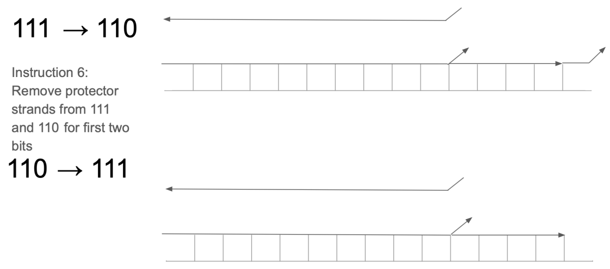

111 → 110
Instruction 6: Remove protector strands from 111 and 110 for first two bits
110 → 111

Figure 23. Instruction 6.

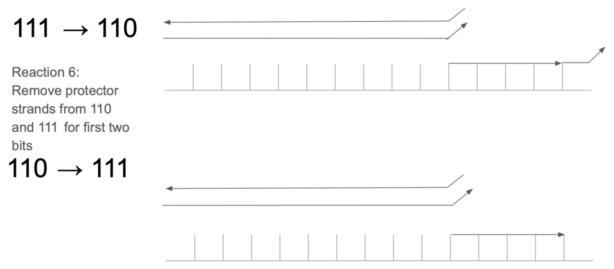

111 → 110
Reaction 6: Remove protector strands from 110 and 111 for first two bits
110 → 111

Figure 24. Reaction 6.

for instruction 7 is seen in Figure 26. The input strands bind to the first two bit locations for both memory registers. The third bit locations differ for each memory register. The 111 memory register does not have an exposed toehold at the third bit location and no reaction occurs. The 110 memory strand has an exposed toehold at the fifth base pair where the input strand binds and displaces the strand encoding the bit as a 0. The DNA strand displacement changes the third bit from a 0 to a 1 which is the goal.

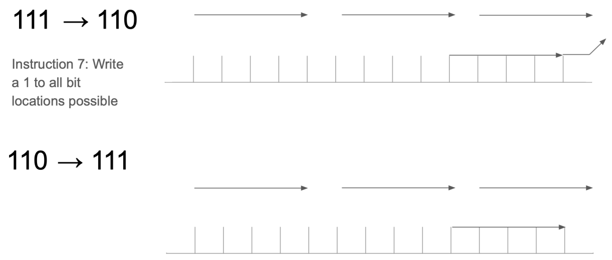


Figure 25. Instruction 7.

The eighth instruction removes the protector strand from the 111 memory register. The detachment process is used to detach the protector strand seen in Figure 27. The input for instruction 8 is complementary to the protector strand added during instruction 4. The reaction for instruction 8 is seen

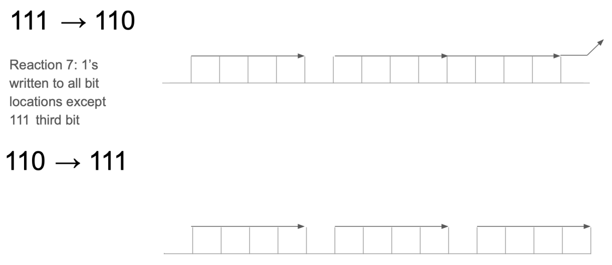

111 → 110
Reaction 7: 1's written to all bit locations except 111 third bit
110 → 111

Figure 26. Reaction 7.

in Figure 28. The input strand removes the protector strand from the 111 memory register. The 110 memory register is not affected because the input strand is not complementary to the strand at that location.

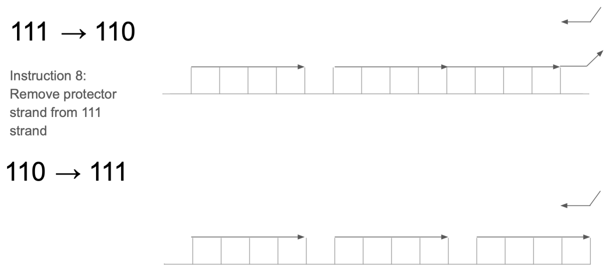


Figure 27. Instruction 8.

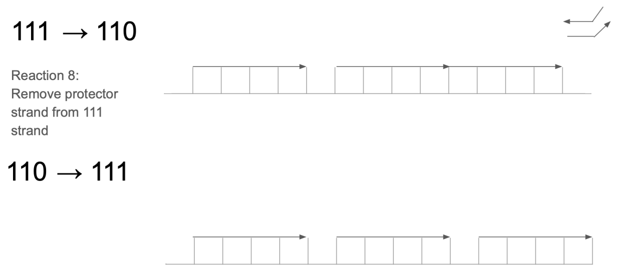


Figure 28. Reaction 8.

The final result is achieved after instruction 8 executes seen in Figure 29. The 111 memory register now encodes the bits 110. The 110 memory register now encodes the bits 111. The input 111 has the output 110 and the input 110 has the output 111 seen in the Toffoli gate truth table in Figure 11. The correct output was achieved by using the same instructions for two different inputs.

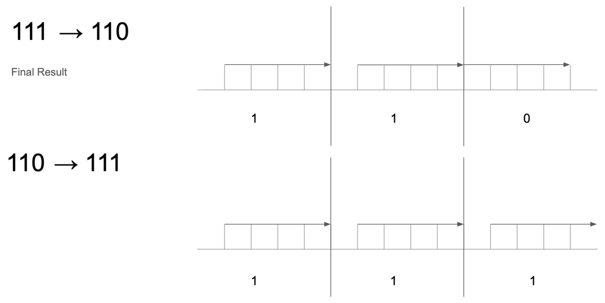


Figure 29. Final result of Toffoli operation.

3.3. Discussion

The Toffoli gate was implemented using the SIMDNA model of in-memory computation. The case where the inputs to the gate are 111 and 110 were modeled to demonstrate the bit flip on the third input bit. The inputs to the Toffoli gate that did not have 1's as the first two bits were not modeled. The next step to this process would be to sequence the first instruction as an if statement to check if the the first two bits are 1's. If they are, performed the sequence of instructions seen above, else return the inputs without modifications.

The attachment instruction event requires two free domains to be present which might conflict with instruction 2. The input strand for instruction 2 is complementary to the fourth and fifth base pair on the third bit location. It binds to the exposed toehold and partially displaces the pre-existing strand on the fourth base pair in this example. If there are not enough base pairs to allow the bind, the encoding scheme can be increased to include 6 base pairs per bit instead of 5. This would provide more area to allow the input strand to bind to the memory register.

This procedure provides an example of how to flip a bit from 0 to 1 and 1 to 0 which can be used in other functions. Future work would include implementing this scheme in a laboratory setting to confirm it is correct. In addition, more quantum gates can be implemented using the SIMDNA model. Eventually a quantum computer could be constructed using DNA. The probability of a qubit reading a 1 or 0 should be studied in regard to the sequence of base pairs in a strand of DNA.

4. Associative Memory

4.1. Introduction

Nicks and DNA strand displacement are used to store and retrieve data in associative memory implementations. Associative memory is an important component of human intelligence (Baum, 1995). The combination of using DNA and associative memory allows for data to be stored that is comparable to the human brain capacities (Baum, 1995). Content addressable memory is a form of associative memory where an input that has partial knowledge of the contents of the memory can retrieve the correct data. An example is storing words or strings of bits into memory and retrieving a specific word or set of words from an input that has partial knowledge of the content of the words.

The benefit of content addressable memory is that a specific address does not need to be known or given to retrieve data. A subset of the data to be retrieved can be given and content addressable memory will be able to match the partial input to data stored in memory and output the matching data. This is advantageous in regards to memory encoded in DNA since they do not follow the same structure as computer memory. Memory encoded in DNA is not in the form of an array or vector and thus is not able to be addressed with a location as seen in standard computer memories.

4.1.1. Memory in DNA

Memory encoded in DNA follows a structure where a vessel or solution holds DNA sequences (Baum, 1995). These sequences are encoded to represent data such as words or strings of bits. There are multiple words or strings of bits in the memory vessel. This is how different information is stored in memory. Each word can be thought of as a memory cell (Baum, 1995). Different kinds of associative memory models in DNA would structure a memory cell in different ways.

4.2. Methods

A synthesis of previous research on associative memory implemented in DNA is performed. Three papers are analyzed to show how associative memory applications are achieved using DNA sequences. Baum encodes data in single stranded DNA using the ordering of a DNA sequence (Baum, 1995). Nowak has each memory cell hold an address and a data portion that also uses the DNA sequences to encoded data. Nowak uses both single stranded and double stranded DNA sequences to store information (Nowak et al., 2006). Qian uses DNA strand displacement to implement a seesaw gate out of DNA and builds linear thresholding circuits from those reactions to implement a Hopfield neural network (Qian et al., 2011).

Nicking is applied to Nowak's implementation of associative memory. A DNA strand displacement tool, Visual DSD, is used to simulate the seesaw gate reactions. This will provide a means to better understand artificial neural networks and DNA strand displacement cascades.

4.3. Baum

One of the first approaches to content addressable memory in DNA was shown by Baum in 1995 (Baum, 1995). He was researching how to build an associative memory larger than the human brain. His approach used single stranded DNA sequences and the ordering of the base pairs to encode data. Multiple approaches of implementing associative memory in DNA were explored using hybridization, parallelism and a native DNA encoding.

4.3.1. Encoding and Storing Data in DNA

Baum described the storage of information as having a memory vessel containing words that are represented as single stranded DNA sequences. Words are strings of bits that are 0 or 1. A 0 or 1 is encoded using a unique single stranded sequence of DNA that is known. These sequences representing 1's and 0's are concatenated together to form words in the memory vessel. Each bit in the word can be thought of as a component.

4.3.2. Retrieving Data in DNA

The retrieval of a word given partial knowledge of the word is as follows. A cue is made up of multiple components that is a subsequence of a word. For each component in the cue, a complementary subsequence can be introduced that matches the bit represented by the component in the DNA sequence word. The input complementary sequence will bond to the complementary sequence on the DNA strand that contains that component and memory. The input complementary sequences are affixed with a magnetic bead.

The magnetic beads are extracted to read the data. The molecules that have bonded with the DNA sequence would also be extracted because they are bonded with the magnetic bead that is attached to the complementary sequence that has bonded with the DNA sequence in the memory vessel. The words represented as single stranded DNA sequences that matched the cue exactly would be the result. Thus, given partial knowledge, different cues, the entire data strand matching those cues would be retrieved.

4.3.3. Restoring Data in Memory

After reading the data, the components are removed from memory and the data is no longer stored in the memory vessel. To restore the data in memory for another read, the DNA molecules would be reintroduced into the memory vessel. The data began as single stranded DNA sequences. After reading, the data are represented as double stranded DNA sequences since the input strands that have complementary sequences to the component values bonded by hybridization. These complementary sequences have a biotin bead attached. The output double stranded DNA strands are split into single stranded DNA strands. The resulting strands are the original strand that was stored in memory before the read and the complementary strands with the biotin beads attached. The strands without beads attached to them would then be added back into the memory vessel and the original data is preserved for the next read.

4.3.4. Achieving Parallelism

Baum describes an approach where he introduces a parallel implementation. The same data encoding is used as described above. Complementary subsequences for each component value would be introduced to the memory at one time. A marker is attached to each complementary subsequence such that when the subsequence bonds to a memory it is complementary to, the data in memory would have an additional marker attached (Baum, 1995). The data in memory with the most markers would be extracted since it matched the input the most. A different technology than biotin beads would have to be used as the markers are relatively small compared to the biotin beads.

4.3.5. Encoding Data in Native DNA

The previous approaches Baum stated are using synthesized DNA to encode binary data. Specific sequences of nucleotide bases are ordered to represent a 1 or 0 and these sequences are concatenated to form strings of bits or words. Baum suggested using native DNA which is found in living organisms. Although native DNA sequences cannot be ordered in a specific manner since it is naturally occurring, it is cheaper and easier to use than synthesized DNA. Native DNA would require a start and stop sequence that would be encoded and concatenated onto native DNA to represent a 1 and use the idea of sparse vectors. A similar procedure to those described above would be performed to retrieve the correct data.

4.4. Nowak

A paper by Nowak published in 2006 describes an associative memory based on DNA strands built in a laboratory. They use a slightly different approach where each memory strand that represents a different piece of information stored in memory has an address portion and a data portion (Nowak et al., 2006). Therefore, given an input, the output would be extracted based on whether the input is associated with the address portion of the data. For example, an input can retrieve zero, one or multiple memory strands in the memory if the input matches the address associated with the memory strand. A binary relation is used to represent the pairings of data and cues (Nowak et al., 2006). This research is performed in a laboratory setting where the DNA strands are built synthetically. The techniques to store data in memory and recall specific strands of data are performed using molecular techniques such as hybridization, denaturation, ligation, cutting and polymerase chain reaction (PCR) (Nowak et al., 2006).

4.4.1. Encoding and Storing Data in Memory

Data is stored into a memory vessel similar to Baums approach. Data is represented using double stranded DNA sequences. The double stranded DNA sequences hold the data which is encoded using the sequences of A, T, C and Gs. Single stranded DNA sequences that represent two cues are attached to either side of the double stranded DNA representing the data portion as seen in Figure 30. The cue at the beginning of the strand is the address cue and the cue at the end of the strand is used later during PCR suppression. Every memory cell has the same cue at the end of the memory cell and it is not used for addressing. The cues are also encoded and addressed based on their sequence of A, T, C and G pairings. Each strand that contains a data portion and two cues is called a memory cell.

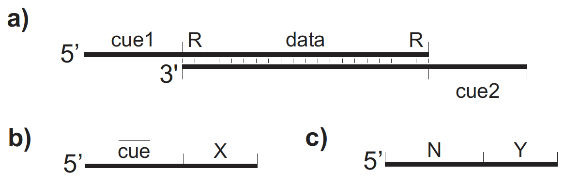


Figure 30. Memory cell (a), input signal (b), temporary signal (c).

The cues have a constant length of N nucleotides. The number of possible cues for a length N is 4 to the power of N cues (Nowak). The data portion does not have a definite length. In Nowaks experiment he used a data length ranging from 65 base pairs to 517 base pairs. The number of different words stored in memory is represented by the number of memory cells in the memory vessel. Once the memory cells are encoded and in the vessel, reading and retrieving of data may begin.

4.4.2. Retrieving Data in DNA

An important part of reading data from memory is the input and temporary signal strands added to the memory vessel. These signals determine the data to be retrieved. An input signal and a temporary signal are single stranded DNA that are synthesized to match the cues attached to the memory cells seen in Figure 30.

The input signal consists of two portions. One portion is a sequence complementary to the cue that is associated with the data to be retrieved. The other portion is a random sequence that can be called X. This sequence will be the same for all input signals and it will not be complementary to any cues in the memory vessel. The sequence X does not have to be unique from the data portion of any of the memory cells in the vessel since they are contained in a double stranded DNA and cannot bind with single strands. The cue portions of the memory cells are single stranded DNA which are able to bind to new strands introduced.

Temporary strands also consists of two portions. Input strands are built specifically to retrieve any data that is associated with one cue. Temporary strands encompass all the other cue sequences in the memory vessel. Thus, there may be different sequences of temporary strands constructed and added to the vessel. The temporary strands have a cue portion that is complementary to any of the cues in the memory vessel that are not the cue associated with the data to be retrieved. The second portion of the temporary signal is the sequence Y that is different from the sequence X and not complementary to any cues in the memory vessel. Again, the sequence Y does not need to be unique from the data portions in the memory cells since they are bounded to a double stranded DNA and cannot bind to any single stranded DNA.

Reading from memory is performed by adding the input signals and temporary signals to the vessel and performing ligation and PCR suppression. First, the input signals are added to the memory vessel. The input signals will contain the complementary sequence to the cue that is attached to the data to be retrieved. The number of input signals will be greater than or equal to the number of memory cells in the vessel in case all memory cells have the same given cue and all should be retrieved. Once the input signals are added they will bond to the complementary cues if they exist during ligation.

Next, the temporary signals are added to the memory vessel. The number of temporary signals will be greater than the two times the number of memory vessels multiplied by the number of possible cues. This will ensure that all remaining cues will be bonded with the correct temporary signal.

After both the input and temporary signal strands are added, ligation occurs. Ligation is the joining of two nucleic acid fragments through an action enzyme, ligase (Nowak et al., 2006). Ligation forms covalent bonds between the complementary cues such that every memory cell has a sticky end of the X sequence or the Y sequence and the other end containing the Y sequence. All memory cells will contain the Y sequence on at least one of their ends due to the second cue not being used for addressing, thus, no input signals would be able to bond to both ends of a memory cell. The memory cells that do not contain the correct data portion to be retrieved will have Y sequence on both ends because the X sequence is only associated with the input signals. This is important for the next step which is PCR with suppression.

PCR with suppression is a molecular biology technique that will amplify the DNA strands with different sequences on the ends and suppress the DNA strands with the same sequences on the ends into pan-like structures (Nowak et al., 2006). The data associated with the input signal is amplified and can be identified after PCR with suppression using a gel electrophoresis diagram seen in Figure 31. Each column represents which cue is being searched and the lines represent the output of the data associated with the cue. Column 7 is a control to test suppression, therefore, no lines are present which indicates it works correctly.

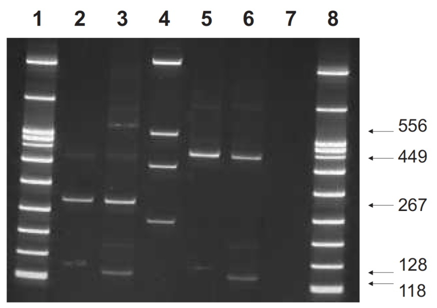


Figure 31. Experimental result from PCR with suppression.

4.4.3. Restoring Data in Memory

When the sequences are amplified the number of memory cells increases which makes it easy to identify the correct data associated with the input signal. However, the memory is corrupted after this process. Each memory cell has input and temporary signals bonded to them from the ligation step. Nowak created a scheme to restore the data in memory after it has been read to prepare it for the next reading. The input and temporary signals are removed from all memory cells by cutting using restriction enzymes (Nowak et al., 2006). The input and temporary signals are then separated from the memory cells using electrophoresis gel (Nowak et al., 2006). The memory vessel is then restored to the original encoding and information as was present in the beginning of the experiment.

4.4.4. Modifications to Approach

One major drawback with this approach is the retrieval of data. Binary relations are used to define the structure of this experiment. For example, there exist a data set D which contains data d1, d2, d3, d4 and a cue set C which contains cues c1, c2, c3. The binary relation for an example set of memory cells would be the relation R which contains (c1, d1), (c2, d3), (c3, d1), (c3, d2). Therefore, the data being retrieved is measured as the presence of the data from the correct given cue rather than reading the data. This form of an associative memory is slightly incomplete as content addressable memory would have data stored in memory and it would only be retrieved as the entire set of data given an input that has partial information of the data. This scheme uses cues and addresses which do not match data portion of the associated memory cell and do not retrieve the data that corresponds to the given cue.

This problem can be fixed by encoding the data specifically for reading. Nicks can be used to encode the binary bits in the data portion of the memory cells. This would allow for reading after the correct memory cells were addressed from a given cue. A couple of steps would be added to the existing procedure stated above. If PCR with suppression only provides readouts of data in an electrophoresis gel diagram, a separate memory vessel/solution would need to be created to hold copies of the data. This would allow the data to be read using the nicking procedure explained below.

The existing procedure would be modified by adding a step during the encoding of data. The data portion of the memory cells would be nicked at specific nucleotide base locations to encode a 1 or 0. The double stranded DNA would not be affected during the rest of the procedure since heat is not applied to the memory cells, thus the integrity of the strand will hold. The memory cells could have data portions that are nicked as a specific string of bits to represent words. The rest of the procedure stated above will follow. The cues are unchanged and the same procedure to add input and temporary signals can be used although this does not fix the problem addressing with partial data as an input.

The input and temporary strands would be added and ligation and PCR with suppression would be performed. After this occurs the correct memory cells that match the input signal would be amplified. Instead of stopping at this point, the data associated with that memory cell would be read. Heat is applied to the memory vessel that contains the copy of the memory cells for the nicking procedure. The result is memory cells with toeholds in the data portion that represent the string of bits. This vessel would have helper strands added to the vessel to read out the bits. Helper strands are synthesized single stranded DNA that are complementary to each bit location in the data portion. Thus, the helper strand will perform DNA strand displacement when reading the bit at a specific location in the data portion of the memory cell. This will release the strand that represents the bit and the helper strand will bind to that bit's old location. Reading the data using helper strands will corrupt the memory and it would not be able to be restored unless the strands representing the bits read were reintroduced back into the memory vessel and a reverse reaction occurred.

4.5. Qian

Qian applies DNA strand displacement and DNA computing concepts to an artificial neural network model to develop a Hopfield associative memory. Linear threshold circuits are an artificial neural network that mimics a biology neural network (Qian et al., 2011). The human brain has neurons that communicate with other neurons through synapses. A neuron has an electric potential that when a threshold is met, the neuron sends a pulse to other neurons (Qian et al., 2011). The linear threshold gate works in a similar way. It takes in inputs that are 1 or 0 and also have analog values for weights. The linear threshold gate turns on when the weighted sum of the inputs is greater than or equal to a given threshold (Qian et al., 2011). This is similar to the electric potential in a neuron and when it sends a pulse. This paper explores different functions that can be constructed using the linear threshold gate, specifically building a recurrent linear threshold circuit to act as a neural network to implement a content-addressable associative memory.

4.5.1. Seesaw Network

A seesaw gate motif is use to build linear threshold circuits. The seesaw gate motif has been developed in previous research by Qian and is implemented using DNA and toehold-mediated DNA strand displacement (Qian et al., 2011). The seesaw network has three basic reactions that occur, seesawing, thresholding and reporting.

The seesawing initial state has a single stranded DNA as input and a DNA gate that has an exposed toehold as well as a complementary sequence to part of the single stranded input as seen in Figure 32. The input strand and DNA gate bind at the toehold and DNA strand displacement occurs. The seesawing final state contains an output that was originally bound the the DNA gate and an input gate that is similar to the DNA gate in the initial state, however, the input strand is bound to it instead of the output strand. This reaction is reversible since there is always an exposed toehold and the input and output sequences are complementary to the DNA gates.

The thresholding initial state has a single stranded DNA input and a threshold gate. The threshold gate has two exposed toeholds that will only bind to a specific input that match the toehold sequences. The input DNA strand matches both toeholds and binds to the threshold gate and performs DNA strand displacement as seen in Figure 33. The thresholding final state contains two waste products that cannot be used to bind to strands because there are no exposed toeholds and thus they are stable outputs.

The reporting initial state has a single stranded DNA with an exposed toehold which is an output from a previous reaction and a reporter gate. The reporter gate contains an exposed

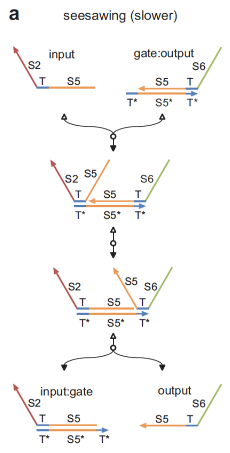


Figure 32. Seesawing (slower).

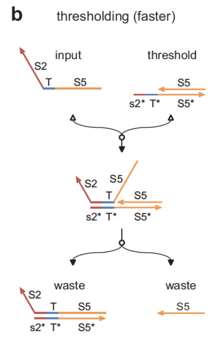


Figure 33. Thresholding (faster).

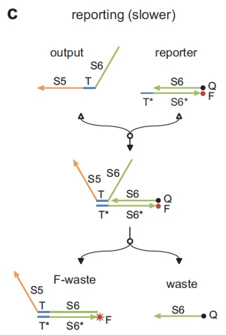


Figure 34. Reporting (slower).

toehold and a specific sequence attached that strands that could bind to it would have to match the sequence. If a strand binds to the reporter gate a florophore is released indicating that a match has been found. The output matches the reporter gate and performs DNA strand displacement which causes the florophore to light up as seen in Figure 34. The reporting final state contains two waste products, one which lights up as a florophore, with no exposed toeholds which indicate they are stable and can no longer react.

4.5.2. Visual DSD: Seesaw Gate Motif Reactions

Visual DSD was used to code the three basic reactions in the seesaw gate motif network. Visual DSD allows a better understanding DNA strand displacement and provides a way to build and simulate linear threshold gates and circuits out of the fundamental building blocks of seesaw gates.

The network for the seesawing reaction was implemented in Figure 35. The input is a single stranded DNA with two different sequences connected by an exposed toehold. The gate starts out bonded to the output strand seen in Figure 35 and has an exposed toehold as well as a sequence complementary to the sequence in the input strand. The double headed arrows indicate the reaction is reversible. The middle box shows DNA strand displacement occurring between the two strands in the initial state. The bottom two boxes are the outputs from the DNA strand displacement. Figure 36 shows the reactions that occur while seesawing. The reaction between the two inputs is used to release the two outputs where the DNA gate is now bound to the output releasing the input strand as a single strand.

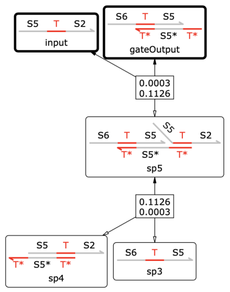


Figure 35. Visual DSD network implementing seesawing (slower).

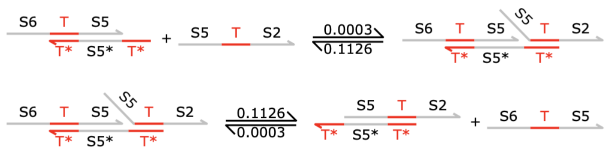


Figure 36. Reactions for seesawing (slower) implemented in Visual DSD.

The network for thresholding is implemented in Figure 37. This network only shows initial and final states of the thresholding reaction. The outputs do not have any exposed toeholds and are stable. The reactions occurring during thresholding bind to the two exposed toeholds and displace the strand S5 from the threshold gate as seen in Figure 38. The threshold gate with S2 has a portion of S2 as an exposed toehold and part of it as a normal single stranded DNA. Having two exposed toeholds indicates that this reaction occurs faster than the seesawing and reporting reactions. The single headed arrow indicates that this is a forward only reaction and cannot be reversed.

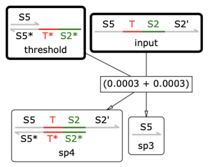


Figure 37. Visual DSD network implementing thresholding (faster).

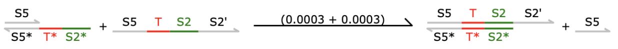


Figure 38. Reactions for thresholding (faster) implemented in Visual DSD.

The network for reporting is implemented in Figure 39. This network only shows initial and final state DNA strands. The outputs are stable and do not have any exposed toeholds. The reactions for the reporting are shown in Figure 40 where fl represents a florophore that will be released when the correct single stranded DNA binds to the exposed toehold. This strand must be the same sequence as S6 since it is attached to the florophore strand. The reaction only has one exposed toehold in the initial state so it performs slower than the thresholding reactions. The single headed arrow indicates that this is a forward only reaction and cannot be reversed.

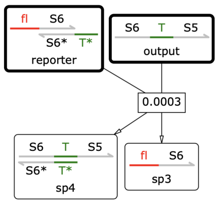


Figure 39. Visual DSD network implementing reporting (slower).

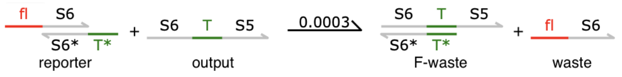


Figure 40. Reactions for reporting (slower) implemented in Visual DSD.

4.5.3. The Seesaw Catalyst

The seesaw gate catalyst can be constructed from the seesaw gate motif. The seesaw gate catalyst is a multi-step process where fuel is introduced as a catalyst. An example of the seesaw gate and DNA strands to implement the seesaw gate are shown in Figure 41. The initial state has all the strands that are in shaded boxes. The final state has all the strands that are in outlined boxes. The seesaw gate catalyst CRN was implemented in Visual DSD to understand the network and reactions. These are seen in Figure 42 and 43, respectively. There are multiple reactions occurring simultaneously through forward and reverse reactions. All reactions are reversible such that the outputs are not stable and can react with their exposed toeholds.

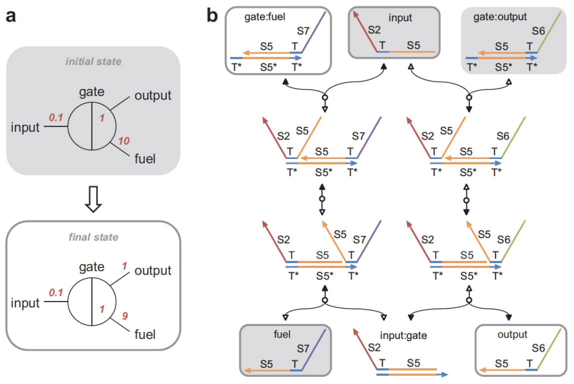


Figure 41. The seesaw catalyst.

The code shown in Figure 44 demonstrates how strands are built in Visual DSD. The concentration or the number of strands can also be specified. The simulation followed the numbers present in Figure 41 in the seesaw gate motif. The initial state includes 10 fuel strands, 0.1 input strand and 1 gate:output strand. The final state includes 0.1 input strands, 1 gate:fuel strand, 1 output strand and 9 fuel strands since one was used during the reaction.

Figure 45 shows the simulation results. The blue line represents the fuel being reduced from 10 to 9. The red line represents the gate:fuel increasing from 0 to 1 since it was not present in the initial state. The green line represents the gate:output which goes from 1 to 0 as it is not present in the final state. The yellow line which may not be visible from

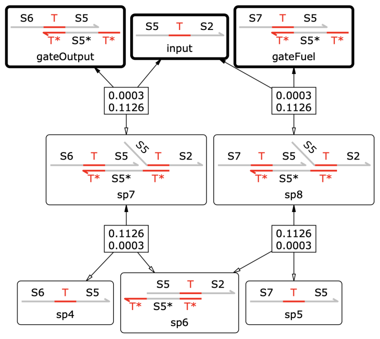


Figure 42. Visual DSD network implementing the seesaw catalyst.

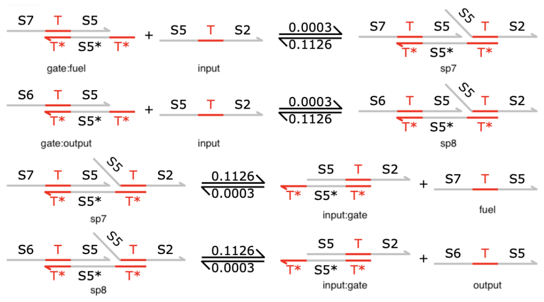


Figure 43. Reactions for the seesaw catalyst implemented in Visual DSD.

Figure 45 represents the output that goes from 0 to 1 since it was not present in the initial state. The input is represented by a pink line that roughly stays at 0.1 during the simulation. The simulation results match with the output in Figure 41.

4.5.4. Hopfield Associative Memory

The Hopfield associative memory was implemented using a recurrent linear threshold circuit. The recurrence provides feedback similar to how a brain processes information with feedback from other neurons. The three basic reactions from the seesaw gate are used to build three types of seesaw gates that are combined to construct linear threshold gates. A multiply seesaw gate have a fixed threshold of 0.2 can be encoded with concentration or number of strands in DNA. It takes in an input that is 0 or 1 and multiplies it by different output weights to produce multiple outputs. The output strand is where the weights are defined. An integrating seesaw gate takes in multiple inputs and no threshold or fuel. The output produced is the sum of all inputs. A thresholding seesaw gate has a threshold and one output. The gate takes in one input and if it is greater than or equal to the threshold the output will be set to 1, otherwise 0.

These three gates are combined to create a linear threshold

```
directive simulation {
 final=60000;
 plots=[gateFuel();gateOutput();fuel();output();input()];
}
directive simulator deterministic
directive parameters [k=0.3;u=0.1]
directive compilation infinite

dom T = {bind=k;unbind=u;colour="red"}

def gateFuel() = <S7>[T^ S5]{T^*}
def fuel() = <S7 T^ S5>
def input() = <S5 T^ S2>
def gateOutput() = <S6>[T^ S5]{T^*}
def output() = <S6 T^ S5>

( 10.0 fuel()
| 0.1 input()
| 1.0 gateOutput()
| 0.0 gateFuel()
| 0.0 output())
```

Figure 44. Code to simulate seesaw catalyst implemented in Visual DSD.

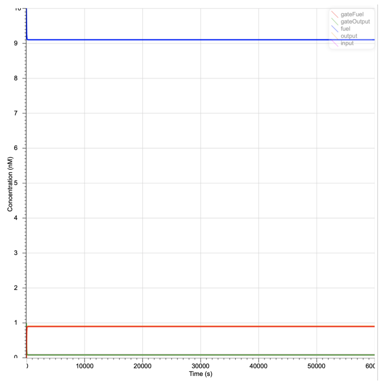


Figure 45. Simulation of seesaw catalyst implemented in Visual DSD.

circuit. A 2 input linear threshold gate starts with 2 multiplying gates that feed their input into an integrating gate. The integrating gate's output are fed into the inputs of a thresholding gate. This follows the procedure of a linear threshold gate where a set of inputs are multiplied by their weight which the multiplying gates achieve. Next, the products are added together into an overall sum which is achieved using the integrating gate. The final step is checking whether the final sum is greater than or equal to the set threshold which is accomplished by the thresholding seesaw gate. A reporter gate uses the florophore to signal if the gate is on or off.

The Hopfield associative memory uses a combination of the different types of seesaw gates and includes feedback. The weights and thresholds are chosen based on in silico learning and simulation. Four patterns of four bits were trained for the DNA to remember based on the weights and threshold values. These are the data stored in memory that will be compared to the input of partial data. Once the weights are set the experiment can be performed in a laboratory setting. Linear threshold gates are connected to each other to build a linear threshold circuit. The correct weight and threshold values enabled the neural network to retrieve the correct data given a specific partial input. The results were accurate for the associative memory and included states of unknown variables. Visual DSD was used to confirm the experiments using a stochastic simulation (Qian et al., 2011). This coding of the associative memory used specific sequences of A, T, C and G pairings. To implement a simple linear threshold gate requires 60 different strands of DNA so the associative memory simulation would be very complicated.

4.6. Discussion

Three different approaches to implementing associative memory in DNA were studied. The original idea proposed by Baum in 1995 is still being used in research today with modifications and use of better molecular biology tools. The Nowak and Qian papers demonstrated different approaches that both used synthesized DNA to achieve the correct ordering of nucleotide bases. Nowak's experiment was conducted in a laboratory setting and Qian used both simulations and a laboratory setting to conduct the experiment. A modification to Nowak's approach was provided that would be able to read the data once the correct memory cells were addressed. This would allow native DNA to be used for the data portion of the memory cells with the procedure of nicking to encode bits. This woudl reduce the cost of creating artificial DNA. Simulations were performed in Visual DSD to understand seesaw gates and linear threshold gates. Although this research has been simulated previously, there had not been a high level simulation of the different seesaw reactions or seesaw catalyst network. The three basic seesaw reactions can be combined to form four types of seesaw gates. These are simply DNA strands from the three basic reactions set to specific sequences, weights and thresholds combined together in a solution.

Future work would be to implement different Boolean gates and the Hopfield associative memory in Visual DSD using the basic building blocks of linear threshold circuits. The experiment performed by Nowak would also need to be modified to include native DNA and nicks. This would be done in a laboratory setting. There are more approaches to implementing associative memory in DNA that can be explored. The mismatch during hybridization has been used advantageously to implement learning within associative memories. The goal of this synthesis of associative memory implementations was to modify existing implementations to use nicks instead of DNA sequences to encode data. A better understanding of associative memory and DNA was achieved.

5. Conclusion

DNA concepts and techniques were studied and applied to three different concepts in computing. A simple Boolean logic gate example was analyzed and simulated using Visual DSD. A model for in-memory computation was used to build a quantum gate to explore the potential of DNA quantum computing. Associative memory implementations in DNA were analyzed to understand the different models proposed to implement one concept. An implementation of associative memory was modified to include the nicking scheme to encode and read binary data. A Hopfield associative memory implemented in DNA was studied to understand how computation and biology pair together nicely to implement a neural network using DNA. The applications of DNA storage and computation is extensive and promising for future developments in technology.

References

Baum, E. B. Building an associative memory vastly larger than the brain. *Science*, 268:583–585, 1995.

Lakin, M. R., Youssef, S., Polo, F., Emmott, S., and Phillips, A. Visual dsd. *Bioinformatics*, 27(22):3211–3213, November 2011. ISSN 1367-4803. doi: 10.1093/bioinformatics/btr543. URL `http://dx.doi.org/10.1093/bioinformatics/btr543`.

Nowak, R., J. Mulawka, J., and Pucienniczak, A. Molecular associative memory built on dna. 03 2006. doi: 10.1117/12.674904.

Qian, L., Winfree, E., and Bruck, J. Neural network computation with dna strand displacement cascades. *Nature*, 475:368–72, 07 2011. doi: 10.1038/nature10262.

Wang, B., Chalk, C., and Soloveichik, D. Simdna: Single instruction, multiple data computation with dna strand displacement cascades. unpublished.