Algorithms in Abstract DNA Self Assembly

Xingsi Zhong
University of Texas-Pan American

ALGORITHMS IN ABSTRACT DNA SELF ASSEMBLY

A Thesis

by

XINGSI ZHONG

Submitted to the Graduate School of
The University of Texas-Pan American
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2013

Major Subject: Computer Science

ALGORITHMS IN ABSTRACT DNA SELF ASSEMBLY

A Thesis
by
XINGSI ZHONG

COMMITTEE MEMBERS

Dr. Robert Schweller
Chair of Committee

Dr. Zhixiang Chen
Committee Member

Dr. Bin Fu
Committee Member

August 2013

ABSTRACT

Zhong, Xingsi, ALGORITHMS IN ABSTRACT DNA SELF ASSEMBLY. Master of Science (MS), August, 2013, 59 pp., 1 table, 45 figures, references, 18 titles.

For the past two years, I have always been working on the topic of Abstract DNA Tiles Self-Assembly. This is a very new area, driven by the interest of DNA molecules. The feature that the system composed by DNA molecules can be a highly parallelized system, make it much more powerful when comparing with the traditional methods. This thesis will introduce the concept of abstract DNA self-assembly models as well as some interesting problems and their solutions.

DEDICATION

Thank my family for supporting me all my life, especially during my two years' study on the other side of the earth.

ACKNOWLEDGMENTS

I will always be grateful to Dr. Robert Schweller, chair of my thesis committee, for all his mentoring and advice. As well as Dr. Matthew Pattiz.

I would also like to thank my colleagues at the UTPA computer science department.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Self-assembly is a process that a set of relatively simple components automatically attach to an assembly, which start from an initial assembly, under local interactions and eventually become a large, complex, ordered structure. The process is highly automated and a system usually contains a set of very few rules. The process of self-assembly can be easily found throughout the world, such as the grown of crystals, the biological growth and replication. People are now attempt to create artificial self-assembling systems by learning from the nature. One of the goal in this research area is to perform atomically large scale precise manufacturing desired nanoscale products. This research area is very new and promising, successes examples such as using self-assembly techniques to produce processors, and using self-assembly techniques to create drug containers that can accurate deliver drugs to the targets inside the body. Pioneers in the area also realized the capacity of the compute power of this system, the system can be guided by predesigned rules to perform algorithms and programs, and the feature that self-assembly actions are driven by local interactions make the system highly parallelism. There for, the study of how to design an efficiently self-assembly systems can be essential to the future study and engineering of nanotechnology.

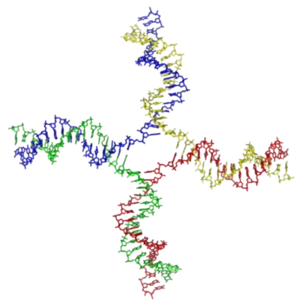
The process of assembly and its final product present execute and output of a predesigned self-assembly system. A tile assembly system can take full advantage of its feature of parallelism and preforming parallel algorithms that usually difficult to achieve in traditional architectures and computational models. For example, the parallel adding and parallel sorting, which the run times are always limited by the hardware, but there is literally no limit by using a tile assembly system, because the computing unites in this system are simply DNA molecules and the supply for computation can always be more than enough. However, the process of self-assembly will use time and geometric space, and the communication between two distant tiles will take large amount of time. So, how to design self-assembly systems to perform programs quickly, which means to design a system that contain only constant tile types and execute the program in a relatively short amount of time, is one of the most interesting direction in this area.

In this thesis, I will mainly talking about two type of self-assembly systems that I mostly focused on during my previous two years working, which are Abstract Tile Assembly Model (ATAM), and Signal Tile Assembly Model (STAM). In the second chapter, I will talk about the physical basis of each model, and the definition of their abstract models. In the third chapter and fourth chapter, I will introduce several interesting problems and their solutions by using these two models, talking about structures and the time complexity. I will also give a brief review of some other works I have done during the past two years in the appendix.

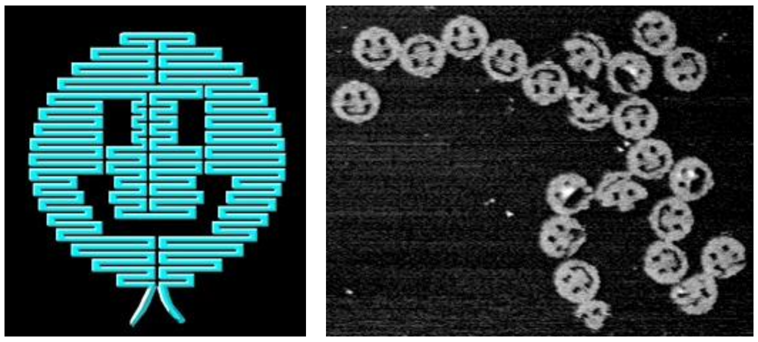
CHAPTER II

PHYSICAL BASIS AND ABSTRACTED MODELS

The general basic rules for DNA self-assembly is based on the DNA base pairing. A pair of DNA base can attach to each other as long as they are match, however, one pair of connected DNA pair may not strong enough to keep two strand hold together, but with more pairs match, two DNA strands can easily attach and strong enough to stay connected. By carefully design DNA strands, people can create desired shapes not only just double strands DNA. One example is the “Holliday Junctions”, shown as figure 1 a), by properly design four DNA strands, the first half of each strand will match the second half of another strand. Holliday Junctions can be created by put these four type of strands into the same test-tube, every four strands will connect together and create a Holliday junctions. Other examples like shapes or maps using the technique called DNA origami, shown as figure 1 b). This two structures are also the basic module used in the ATAM, which we will talk about later.



a)



b)

Figure 1 a) Holliday junction b) DNA origami

By carefully design the DNA strands, people can create all kinds of structures with interesting features. In this chapter, I will introduce the two widely used techniques, the Abstracted DNA Tiles Self-Assembly Model and Signal Tiles Model. I will introduce the basic ideas and physical foundations of each model as well as their abstracted model.

Abstracted DNA Tiles Self-Assembly Model

Abstracted DNA Tiles Self-Assembly Model (ATAM) is one of the earliest and widely used abstracted models in this area, it offers a defined process that an initial structure can get attached by predesigned monomers and grow into a target structure.

Physical Basis

DNA Tiles. DNA tiles or Monomers are the most basic individual components of self-assembly. In DNA self-assembly, a tile is an abstracted module of molecules constructed from DNA strands. In 2D, a tile can be created by slightly modify a Holliday Junctions. Shown as figure 1 a), by design four DNA strands, and the first half of each strand will match the second half of previous strand, four strands can connected and create a Holliday junction. Similarly, we can design a set of DNA strands that match each other but also leave a few unmatched fragments

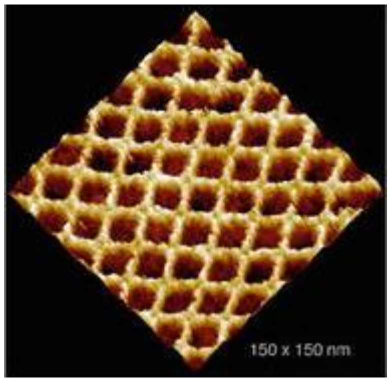


Figure 2 A mesh created by Holliday Junction

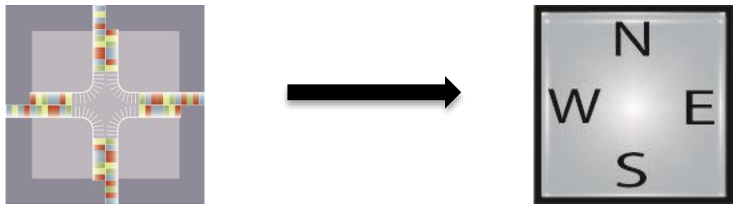


Figure 3 Abstract from the Holliday Junction to an abstracted square tile

at the end of each strands, then the DNA strands can still stick to each other and also leave a short fragments exposed at the end of each strand shows as the left figure in figure 3. Then, the structure can potentially connect to another structure with the corresponding fragments exposed. Shown as figure 2, a mesh can be created by DNA Holliday Junction. We abstract this structures as a *tile*, shown as figure 3, and each end of the strand presents a side of the tile. The exposed fragments on each side can only attach to the corresponding sequence, so we call the fragments on each side of the tile as *glue*, and the type of sequence of the fragments called the *glue type*, so the glue can bound to another tile with corresponding glue type. The length of exposed sequence will determine the ease of attach, so we also abstract the length of an exposed sequence to an integer value called *glue strength*. The value of a glue strength is usually less than 4, and equal or less than the value of *temperature*, which I will introduce later. With the same principle, by using six DNA strands, we can easily build a three dimensional cube tile, which will further contains the Up and Down side. When designing the DNA strings used in tiles, it is easy to insert a piece of florescence label in the tile, thus, people can easily distinguish the tiles under the electron microscopy. Also, it can be used as the *label* of each tile, so we can give the tiles a meaningful tag, for example 0, 1 or empty.

Seed. Seed is a fundamental structure in a DNA tiles self-assembly system, which can allow other monomers in the self-assembly system initially attached with, just like the dust played as the core in the snow flake. There is only one seed in a self-assembly system. A seed in DNA tile self-assembly system is usually created by using the technique called DNA origami. Shown as figure 2 b), a string of DNA can be folded at specified positions by pre design the sequence that part of the sequence can match another part of sequence on the same strand nearby. Also, we could insert some unmatched strings in the origami, so that the unmatched string will not bound any part of the origami but exposed to outside. Then, these exposed strings will allow the corresponding matched strings on other monomers in the system to attach. The format of the seed can be used as the input of the system. With different seed, the output can be different, even with the same tile set.

Temperatures. Temperature τ is an environment variable. It describe the minimum strength with which glues must bind for a tile to attach. It is an integer value equals to 1 or 2 or 3 and usually no larger than 4. For example, if in an abstract DNA tile self-assembly system with τ =2, then only the tiles that the total matched glue strength is equal or larger than 2 can attach to the assembly, if the total band strength is less than 2, the tile cannot band. Figure 4 shows an example of the cooperative binding. The tile can attach to the system only when the total matched glue strength is equal or large than the temperature.

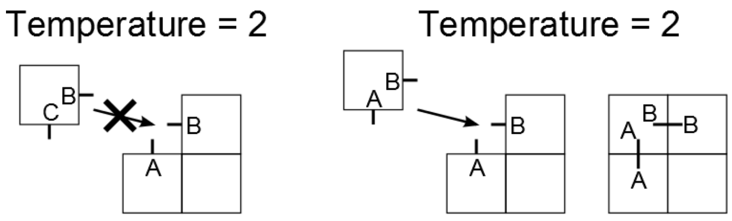


Figure 4 Example of cooperative tile binding

Abstracted Definition of the Model

Tiles. A tile is a square (with four directions N, S, E, W) or cube (with two additional directions U and D) with each side of the shape could be assigned with some glue types. The glue type has some non-negative integer strength. The tile can also been tagged with some label for identification. Once a tile is attached to an assembly, the location of the tile will be get by using integers coordinate. A tile cannot be rotated.

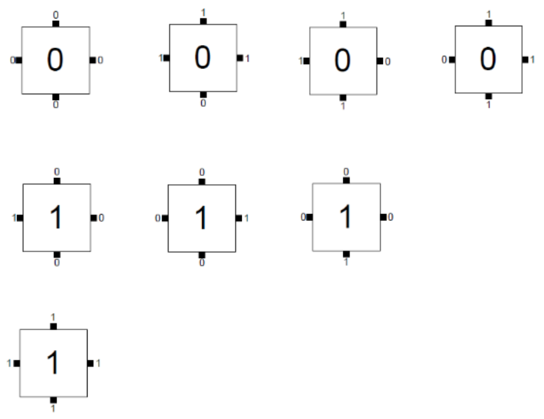
Assemblies. An assembly is a finite set of tiles connected by matched glues and do not overlap with each other, and assign the center of each tile with and integer coordinate or triplet. In other words, each tile in the assembly has a unique coordinate. Glues between two adjacent tiles will be bound and contribute a positive weight only if the two glues have the same glue type. Define the bond graph to be the weighted graph, each element in the graph is a vertex. The weight of and edge between two neighboring tiles is the strength of the matched glues. An assembly is τ-stable, where τ is the temperature value, only if the total weight on the minimum cut of the weighted graph is larger than τ.

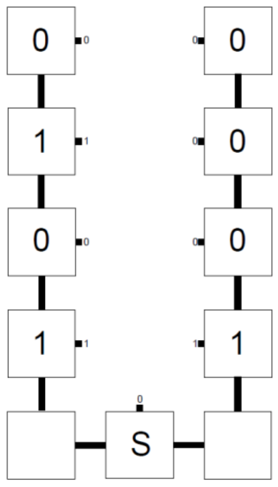
Tile Attachment. Given an integer τ as the temperature, a tile t, and a τ-stable assembly A'. Then, tile t may attach to A' if $A = A' \cup t$ and A is τ-stable.

Tile Systems. A tile system $\Gamma = (T, S, \tau)$, where T is the tile set of the system, S is the seed assembly, and τ is a positive integer as the system's temperature.

Examples

Two number addition. Here is a very simple example of adding two numbers together using the traditional method. The figure below shows the input seed 0101 and 0001, and the tile set on its left. The system temperature is 3. .Figure 5 shows the 4 steps to adding these two numbers. In chapter 3, I will introduce a much more powerful algorithm of adding two numbers in a much faster speed





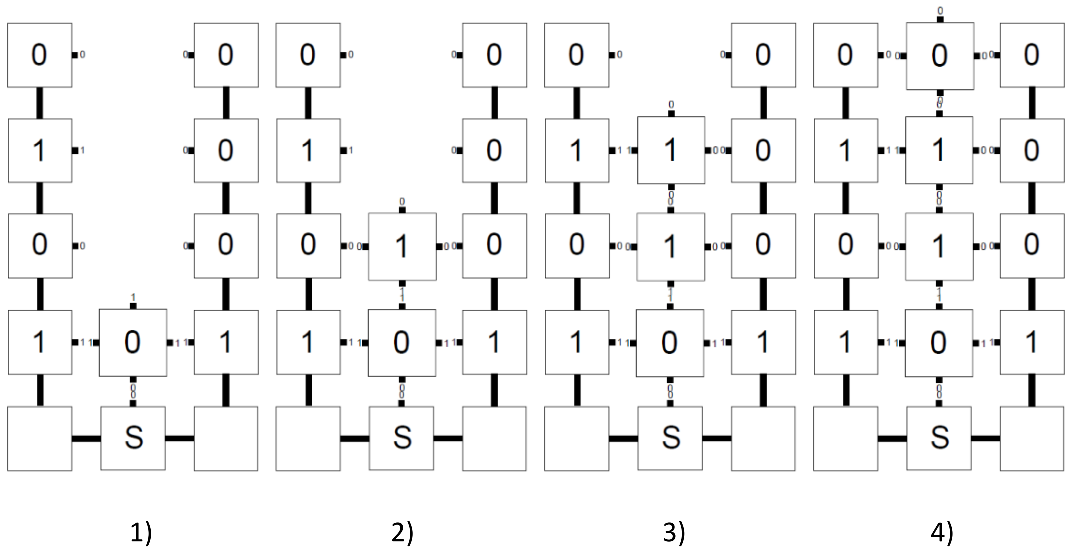


Figure 5 Four steps of adding two numbers. This example shows 101+1=110

Signal Tiles Self-Assembly

Signal tiles has all the physical basis that basic DNA tiles has. The different is, more than all the features that DNA tiles already have, signal tiles can sending signals between each tiles through the bounds. After the input side of a signal tile attached with a correspondence glue, the output glue can be turned on from latent, or turned off from either on or latent, and the process is

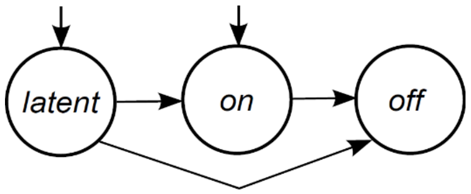


Figure 6 The three stages of the glue of signal tiles

irreversible. This feature makes the Signal tiles self-assembly system much powerful than the basic DNA tiles system. The seed and temperature of a signal tiles self-assembly system has exactly the same features with the DNA tiles self-assembly system, so here I only introduce the signal passing functions in this system.

DNA Walker

DNA walker is the physical technique used in the signal tiles. Today, there are several type of techniques can perform the DNA walker technique, but they are all follows the same idea, here I introduce one of the most easy to understand technique. Instead of using simple single strand DNA sequences, the signal tiles use the strands of DNA structures that attached with an array of comb teeth liked structures, so we can imagine that the signal tile is not assembled by few DNA strands, but a few DNA structure bands. On the input side of this band, a short piece of DNA strand half attached at the glue, and half attached at one comb teeth, and this

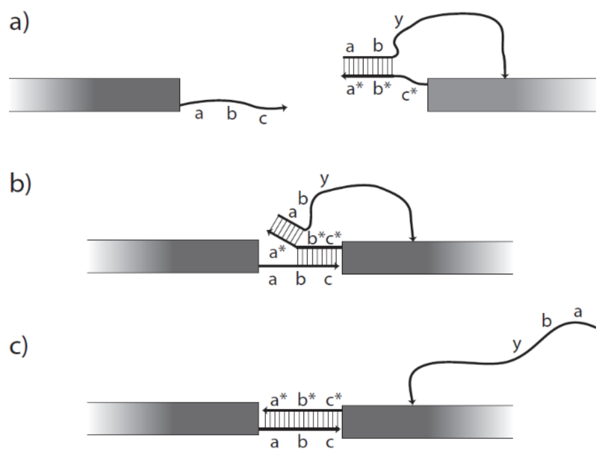


Figure 7 The DNA walker (shows as the strand with sequence of a, b and y) is been pushed up by the attached input glue on the left (shows as the strand with sequence a, b and c).

short piece of DNA strand is called the DNA walker. The input glue of this band is half attached with the walker and half exposed, shown as figure 7 a). When a correspondence glue that matches the input side glue, it will first attach the exposed half part of glue on the input side, shown as figure 7 b), and compete with the DNA walker until fully attached on the input side glue and push the walker away, shown as figure 7 c). During the competition, the half part that used to connect with the walker will be attached by either part of the walker or part of the correspondence glue and their chance to attach are equal, so they will move back and forth just like a seesaw. However the rest part of the input glue is always connect with the input glue, so the input glue will never fail in the competition, but it is possible for the walker to be fully detached. After the walker detached from the input glue, same principle will be used on the comb tooth track. Two type of hairpin structures will be used as fuel to push the walker all the way to the other side of the track. Eventually, on the other side of the track, by using the same idea, the output glue can be turned off, if the walker fully matches the output glue, or turn the output glue on. The using of the fuel and irreversible features makes the signal tiles can send any signals only once, and never been reused.

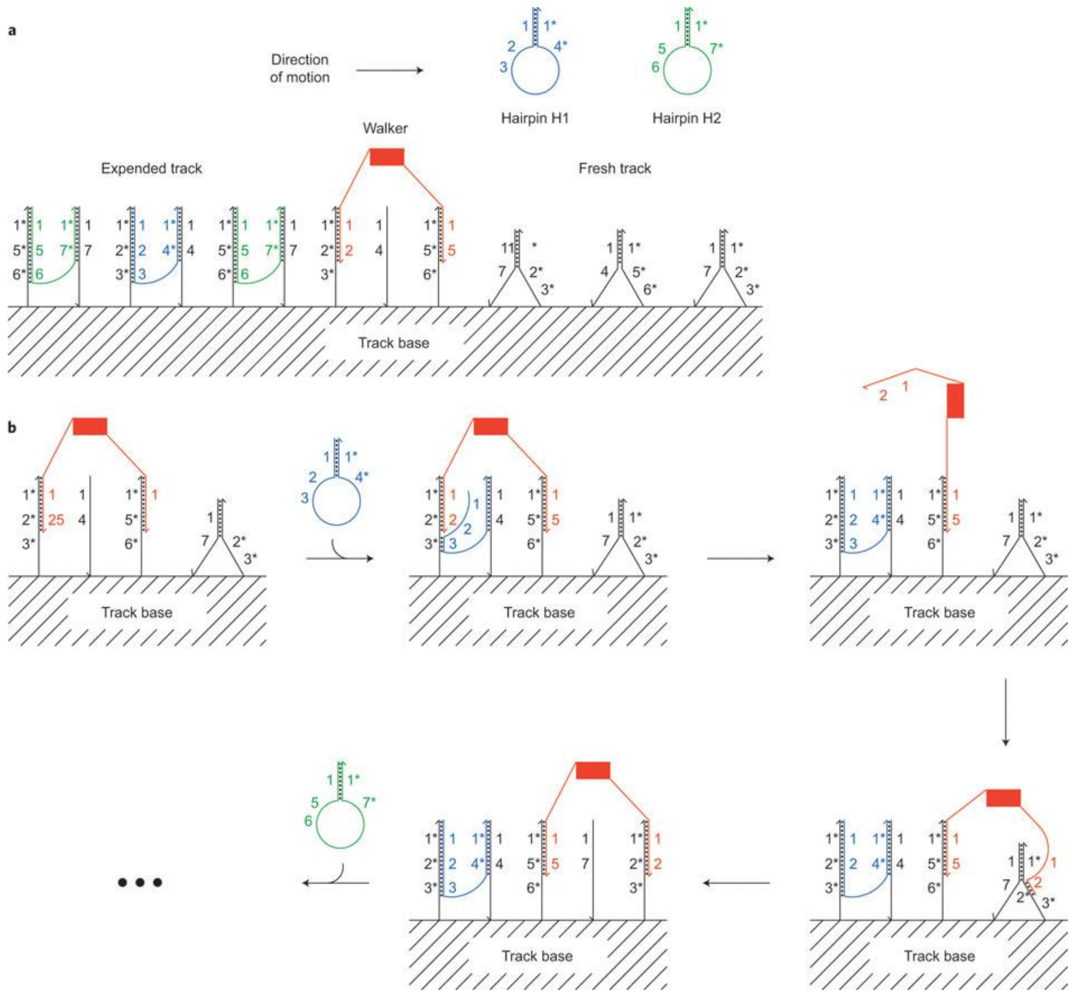


Figure 8 A schematic diagram to show how the walker is walking on the track

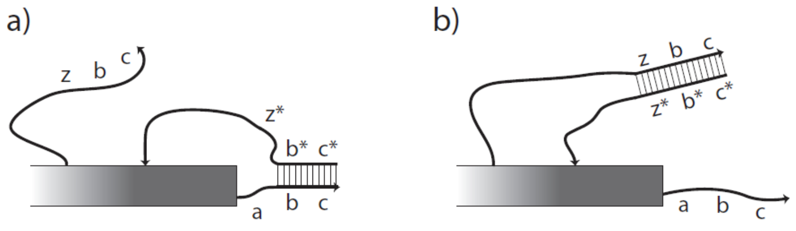


Figure 9 Example of a glue been turned on by the walker shows as the strand with sequence z, b and c

Signal Tiles Model

Here we describe the signal tile assembly model (STAM) by define the concepts of a signal tile, glue slots, as well as three stages of a glue slot. The definition of assembly is same with the ATAM, so we do not discuss it again.

Glue Slots. Glue slots are similar with the definition of glue in the ATAM, but more than a single glue with glue type and strength, the glue slots is always in one of the three states, which are on, off, or latent.

Active Tiles. Active tile is also have the similar definition with the tiles in ATAM. Only the glue slots in the on state on an active tile will be used for the attachment. In the section using STAM, we will use the term *tile* and *active tile* interchangeably.

Reactions. More than just have tiles attached in ATAM, STAM can change the state of the glue to transform to another stage.

Break reaction. A assembly will perform a break reaction if the bond graph of the assembly has a cut with the combined strength less than temperature τ, then the current assembly a will be separate follows the cut into assemblies b_1 and b_2.

Glue-flip reaction. A reaction if assembly b can be achieved from changing the state of any glue form on or latent to off, or from latent to on.

Batches. A batch is a set of assemblies. A batch B can be transform to batch B' if one of the reactions can be applied at temperature τ to get B' from B. A *batch sequence* at temperature τ is any sequence of batched from $a_1$to a_r such that $a_i$is transformed from a_{i-1}, where i is a positive integer less than r.

Signal Tile System. A Signal Tile System is an ordered pair (B, τ) where B is the initial batch or seed, and τ is the temperature. For any batch B', it is producible by (B, τ) only if there is a batch sequence from $B_1$to B_r such that $B' = B_r$.

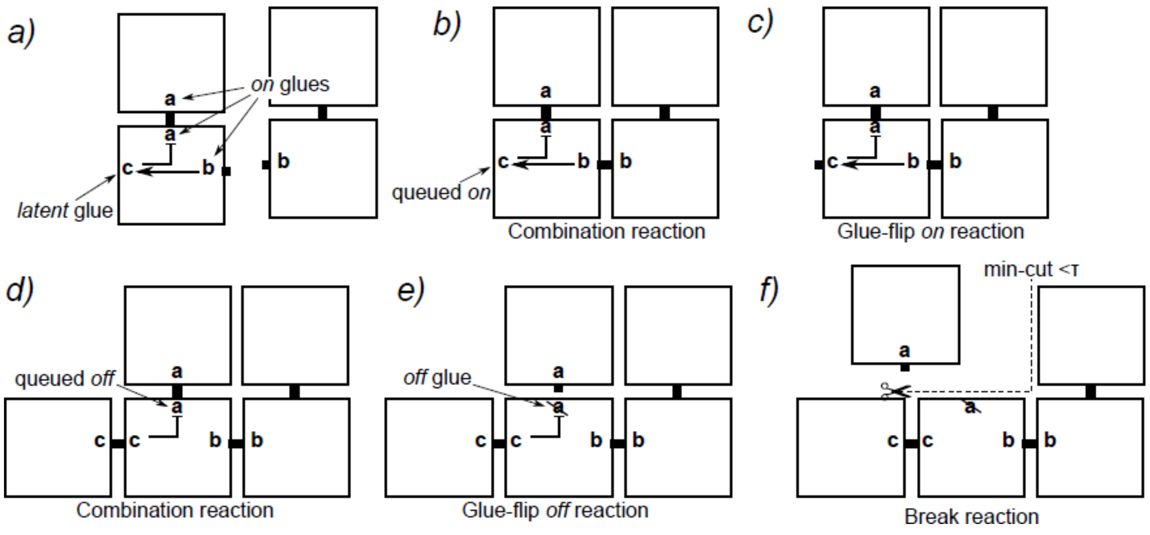


Figure 10 An example sequence of reactions.

CHAPTER III

FAST ARITHMETIC

The solution for the most fundamental problems can always be tricky. An efficient solution for the fundamental problems is not easy for traditional algorithms and hardware. In this chapter, I will focus on the complexity of arithmetic primitives using the Abstracted Tile Self-Assembly Model (ATAM). By using this model, we study the time complexity of adding two n-bit numbers, and achieved a runtime with lower bound of $\Omega(\sqrt{n})$ in 2D assembly and $\Omega(\sqrt[3]{n})$ in 3D assembly. I will also show that we further designed algorithms to achieve an $O(\log n)$ average case time, and a combined algorithm to achieve an $O(\log n)$ average $\Omega(\sqrt{n})$ worst case addition in 2D. Then, we further designed a multiplication algorithm using those results and achieve a runtime of $O(n^{\frac{5}{6}})$. The results here are dramatically faster than any previous result and almost unbeatable in this area. The results table for chapter Fast Arithmetic as well as the previous best known result is shown as Table 1. In addition to this analytical results, the data analysis of the result of ATAM simulations were shown to visualize the fast algorithms presented in this chapter, as well as the comparison with previous results.

	Worst Case	Average Case
Addition(2D)	$O(\sqrt{n})$	$O(\log n)$
Addition(3D)	$O(\sqrt[3]{n})$	$O(\log n)$
Multiplication(3D)	$O(n^{\frac{5}{6}})$	
Previous Best Addition(2D) Previous Best Multiplication (2D)	$O(n)$ $O(n)$	

Table 1 Summary of results of fast arithmetic.

Problem Description

Here we will use the notion of *Tile Assembly Computer* (TAC) to present a tile set, temperature and the input and output templates. The input template will be serves as a seed with an arrangement of "*" to present the locations of input bits, which could be one of "0" or "1", and will be used in the seed. The output templates an arrangement of locations that present the positions for the TAC grown from a fully filled template, and will be replaced with tiles tagged as "0" or "1" that shows the output bit string. We denote that a TAC will compute a function f when the seed is arranged with a bit string b, the terminal assembly will encode the value of $f(b)$ on the locations of the output bits in the output template. Please note that it is possible that when mapping the input string to desired locations, it could involve part of the function f and served as a pre computation of the function, and that is what we are trying to prevent from in this research, but sometimes it is hard to notice.

Run Time Models

Figure 11 shows the run time simulate results comparisons by using two well established run time model for ATAM, *Parallel Time Model,* and the *Continuous Time Model*. The parallel time model, in short, simply count the parallel steps. Start from the seed, attach all attachable tiles to the current assembly, count as one parallel step, and repeat the process on the newly created assembly according to the previous step, keep counting until the terminate assembly is got. The continuous time model, very different from parallel time, will accumulate the time for each tile to attach. Start from the seed, a location is randomly selected from all attachable locations for the current assembly, then calculate the time for the appropriate tile to attach. The time for one tile to attach to an assembly is the expected time for the randomly selected location with the appropriate tile attach, which would be an exponentially distributed random variable, which the rate is relative to the number of attachable locations and the type of tiles. It would be easy to understand that with more available positions on an assembly, the time for one tile to attach will be shorter, however, when the TAC contains more tile types, the chance for the right tile to attach would be smaller and the time would be longer. After the appropriate tile attached, repeat the process for the newly created assembly, and accumulate the time until the last tile attached and the assembly become the terminate assembly.

Comparing this two runtime models, the parallel timing model is simple and straightforward and can clearly shows “how many steps” it take for the assembly to terminate; the continues timing model, on the other hand, more complicate but more realistic for considering the time instead of steps.

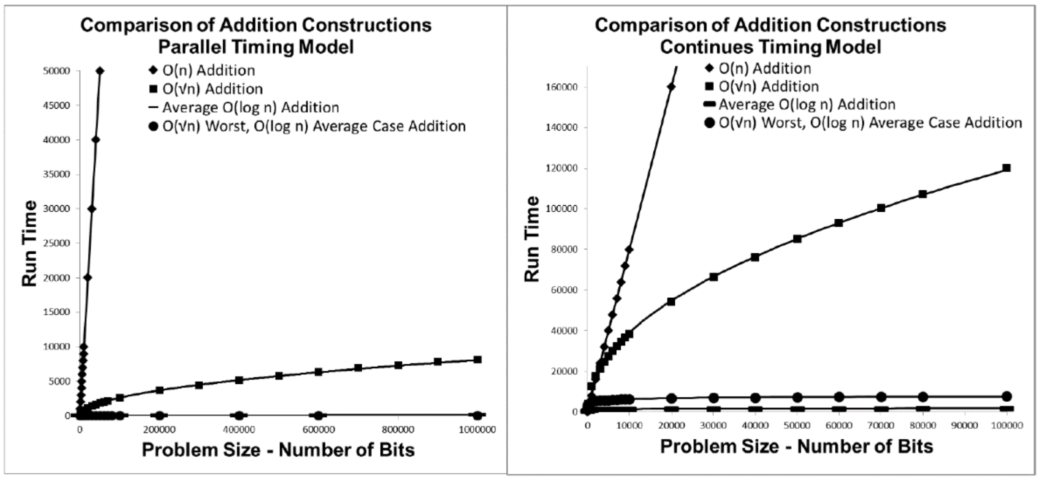


Figure 11 Run time simulate results comparisons of Addition TACs by using Parallel Timing Model and Continues Timing Model

Addition In Average Case Logarithmic Time

In this section, I will introduce an adder TAC algorithm that could reach an average $O(\log n)$ runtime to compute the addition of two n-bit numbers. When adding two binary numbers, the tradition algorithm is to start adding the bits pair from the least significant bit of two addends, calculate the result of current bit and the carry for next bit's computation, repeat the process in the next bit's calculation, until the most significant bit of the result is got. Obviously the tradition algorithm will achieve a runtime of $O(n)$. The reason we usually calculate from the last bit to the first bit is that we need to send the carry to the next bit and get the result. However, not all the carries for next bit's calculation is depend on the previous carry, more specifically, in some digit of the two addends, the two bits could be 1 and 1, the carry for next bit's calculation is always 1, and it is also true for bits combinations of 0 and 0, the carry for next bit's calculation is always 0. So, it is possible to get part of the results without looking all the way back to the first bit's calculation. If the bits combination are 1 and 0, or 0 and 1, then the carry for next bit's calculation will be rely on the previous carry. So, it is also possible that the result for a bit could have to wait until all the results before it is get. So the algorithm will perform an average case $O(\log n)$ runtime and worst case $O(n)$ runtime.

Construction

Input And Output Template. The input template, or seed, for the average case $O(\log n)$ runtime and worst case $O(n)$ runtime TAC is shown as figure 12. The input template is assembled by arranging n chunks, each chunk contains three tiles, which are the corresponding bit of A and B, and a function initializing tile. The input numbers are arranged in the way that the same bits of each numbers are allocated in same chunk and became a bits pair, the bits pairs are arranged in sequence from small to large from east to west, the least significant bits is on the easternmost chunk and the most significant bit is on the westernmost chunk. The easternmost and the westernmost input tiles in the seed are different from the one in others chunks, the easternmost input is the least significant bits pair, no previous carry will be get, so it will send a fake “0” carry signal to the current result position. And similar with the last significant bits pair, there doesn’t exist a higher bits pair to accept the carry it send out, so the last tile will send a signal to accept the carry from the most significant bits pair as the last bit of the result.

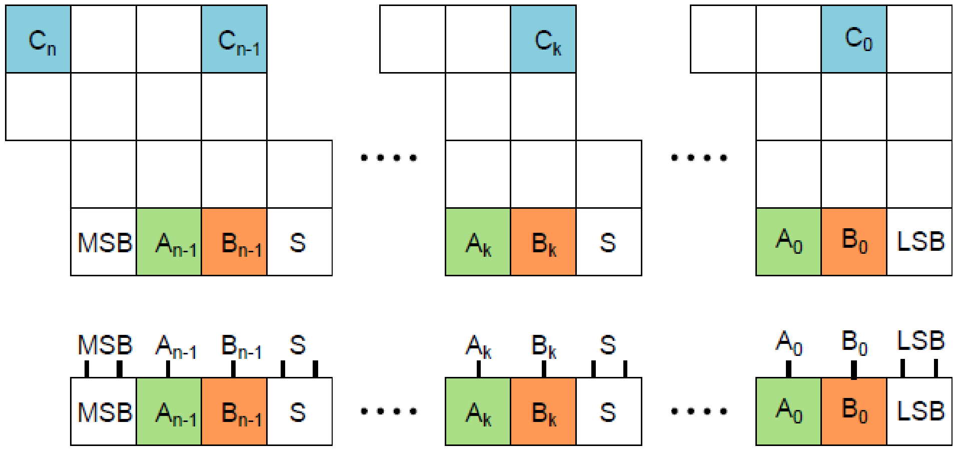


Figure 12 The input and output template for the average case $O(\log n)$ and worst case $O(n)$ runtime adder TAC

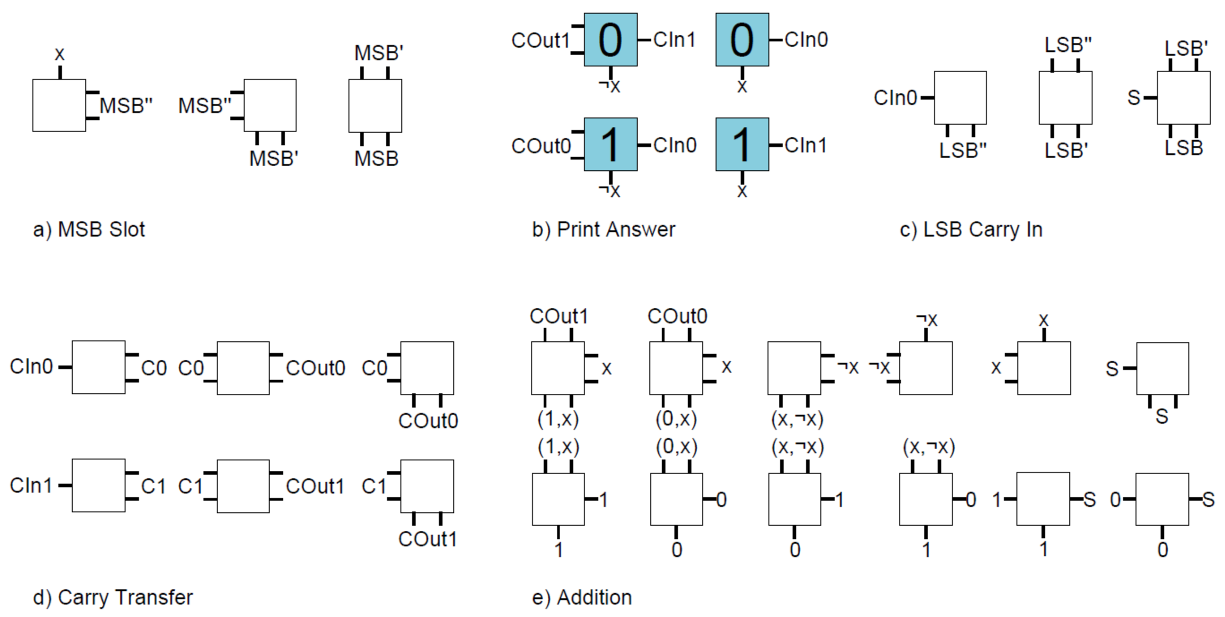


Figure 13 Tile set for implement the average case $O(\log n)$ runtime and worst case $O(n)$ runtime addition

Computing Carry Out Bits. The tricky part to apply the algorithm is to send out the carry to next bits pair immediately if the current bits pair's carry does not depends on the previous bits pair. There are totally 4 combinations of the bits pairs can exist in the input, which are (0 0), (0 1), (1 0), and (1 1). The carry to send to the next bits pair is already known for the combinations (0 0) and (1 1), and the carry of the combinations (0 1) and (1 0) will be depend on previous pairs. An example listed all four combinations is shown as Figure 14. In each chunks,

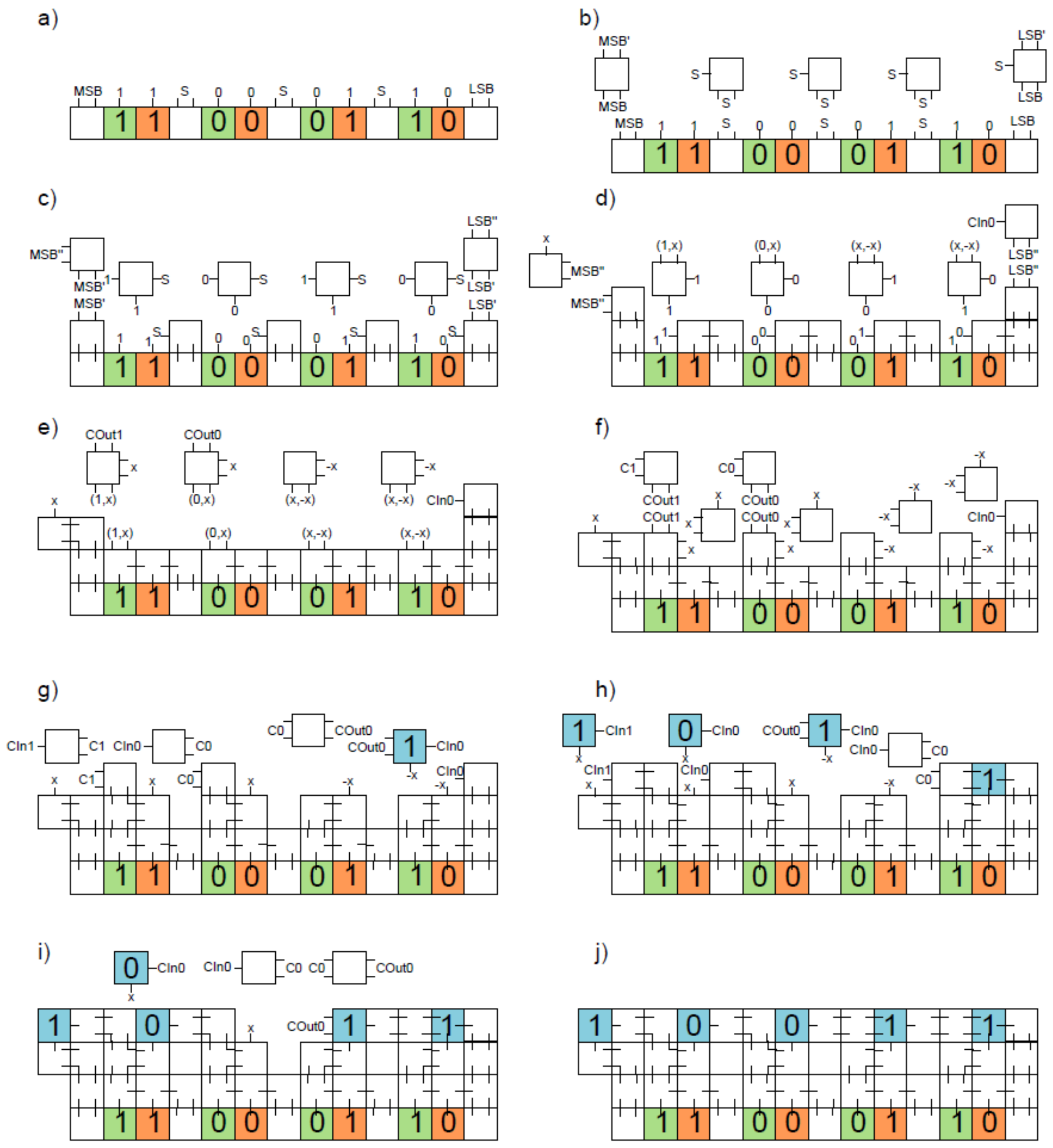
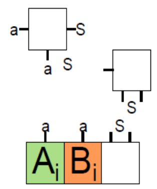
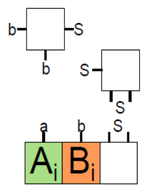


Figure 14 Example of $O(\log n)$ average case addition 1001+1010=10011

a)

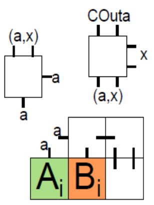


A_i and B_i are both 0s or both 1s.

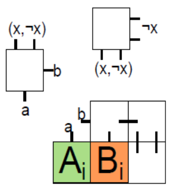


A_i and B_i are different; a=0, b=1 or a=1, b=0.

b)

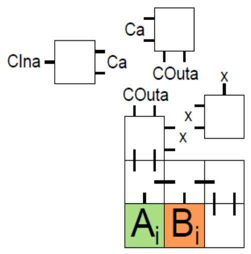


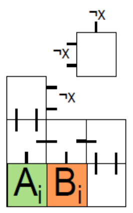
The (a,x) means that the value of the carry generated for next block is *a* and the result in this block will be the value of the carry-in, x, from previous block.



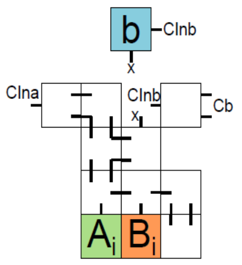
The (x,¬x) means that the value of the carry which will be propagated to the next block is x, which is same as the carry-in from the previous block. Also, the result in this block will be NOT x.

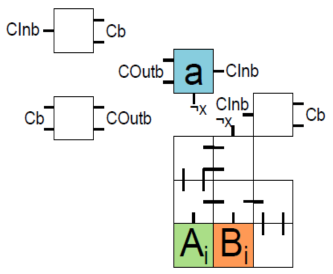
c)





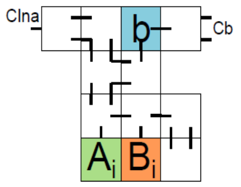
d)

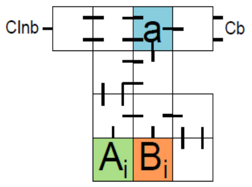




Here we see that if the input values are equal, the carry that was just generated will be sent to the next block immediately. However, if the input values are not equal, the carry for next block will be propagated after the carry-in is accepted.

e)





A_i=B_i=a; carry=b; A_i+B_i+carry=ab

A_i=¬B_i; a=¬b; carry=b; Ai+B_i+carry=ba

Figure 15 The figures on the left side shows the case that a carry could be generated before the addend-pair has received a carry in. The figures on the right side shows the case that a carry out is dependent upon a carry in having been received by the addend-pair.

the TAC start to grow from the right most initial tile. Then after 4 parallel stapes, the assembly will looks like Figure 14 f). Now, the very top of the tile in each chunk will now contains both input bits and decide to send the carry to next bits pair if it can, or simply grows to east and waiting for the income carry from previous bits pair. As mentioned before, the first chunk doesn’t have previous chunk, so the first tile will send a “fake” 0 carry, to the first chunk of bits pair. Similarly, the carry send out from the last chunk will be catch by the tiles growing from the last tile because there is no further bits pair chunks.

Computing the Sum. A carry out bit tile has been computed and send from the lower bits pair chunk to the neighbored higher bits pair chunk. Also the result of the summation of current bits pair without the carry coming in, by simply add the carry from the previous bits pair, TAC can now get the current bit of result, and then send out the carry to next chunk if the carry form current pair is relied on previous chunk.

Time Complexity-Parallel Time

Worst Case. First we show the worst case O(n) runtime. Consider a binary sequence with length 2n presenting two n-bit binary numbers A and B. Denote thatA_{k} B_k presents the k-th bits of A and B. Then, assume for any k from 0 to n, $\mathrm{A}_{\mathrm{k}} \neq B_k$, which means $\mathrm{A}_{\mathrm{k}} = 0$ $B_k = 1$ or $\mathrm{A}_{\mathrm{k}} = 1$ $B_k = 0$. So, the carry of any bits pair will not be get until the previous bits pair send out the carry. For any chunk, after 5 parallel steps, the result will be get after the previous chunk get the result and send out the carry, which will take 3 parallel steps. So, for the last chunk, the n-1 bit will get the result after 3n+5 step, and for the last bit, 3 more step which is 3n+8. So the total parallel worst case time is O(n).

Average Case. Similar with previous section, consider a binary sequence with length 2n presenting two n-bit binary numbers A and B. Then, given randomly give 0 or 1 to A_k *and* B_k for any k from 0 to n. Then, the for any pairs that contains both 0 or both 1, it can send out the carries immediately. However, the total run time for the TAC is based on the longest sequence of the chunks that cannot send the carry until received from the previous chunk. The chance of the combinations will lead to a carry dependency is ½, just like the chance to flip coins, it is been proved that the expected longest run of heads up in n coin tosses is $O(\log n)$. So, the expected length of the longest sequence as well as the average case runtime is $O(\log n)$.

Optimal $O(\sqrt{n})$Addition

In this section I will introduce a adder TAC that achieves a run time of $O(\sqrt{n})$. The idea is to distribute the input n bit numbers in to a $\sqrt{n}$ matrix, with $\sqrt{n}$ rows and $\sqrt{n}$ columns. The original number will be divide in to $\sqrt{n}$ sections and each contains $\sqrt{n}$ bits. The TAC will first compute the sumation of $\sqrt{n}$ bits chunks in each section in parallel. Because the current section

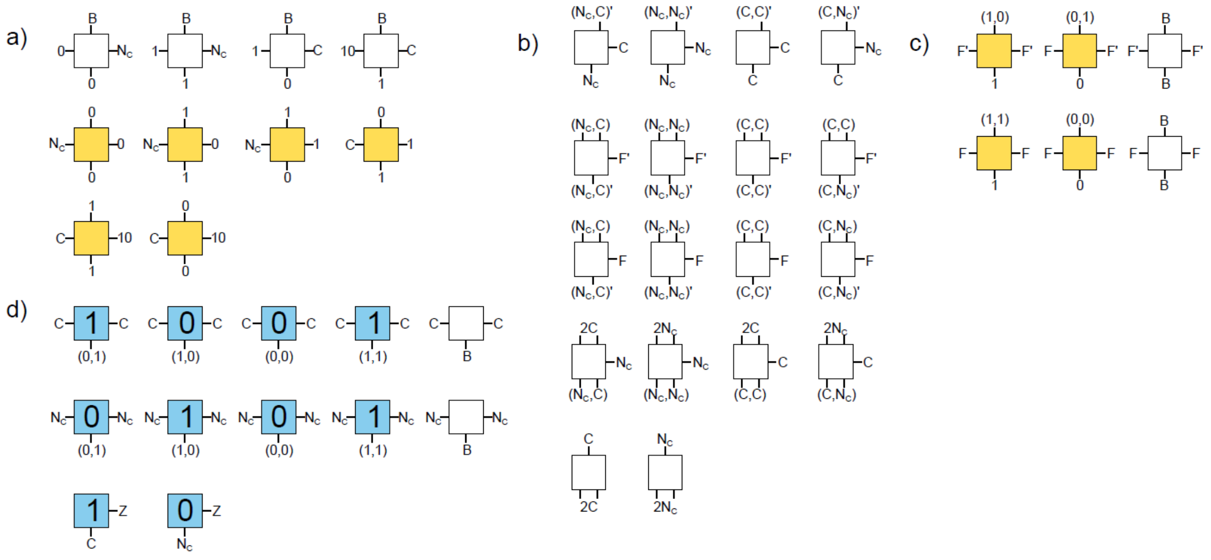


Figure 16 Tile set for$O\sqrt{n}$ addition

doesn’t know what is the value of the carry come from the previous section, so the result will now contain both the result with the carry 0 or carry 1, and also the coresponding carry will be send to next section. After the first section finished runing, it will send the carry to the next secton, and the next section will directly send the correct carry to the next section and select the coresponding result as the output in current section.

Construction

Input And Output Template. An example of the input and output template shown as figure 18**.** The least significant bit of A is shown as the A_0, and the least significant bit of B is shown as the B_0. Relatively, the most significant bit of A is shown as the A_8, and the most significant bit of B is shown as the B_8. C is the result which has 10 bits, and the most significant

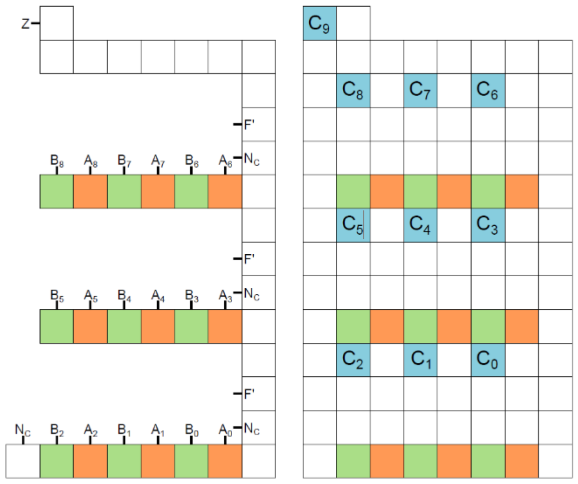


Figure 17 The Input template on the left and Output template on the right

bit of C is shown as the C_9.

Step One: Addition. Step one is pretty straightforward. Start from the lowest bit in each column, perform the addition of each sections with the predicted carry 0. First, start from the lowest bit a tile will copy the first bit input, then, the second tile will calculate the result and the carry and send it to the next pair. Then, the second pair will receive the carry, as well as copy the first bit in this pair to the next bit. Then repeat the process and get the first layer of result.

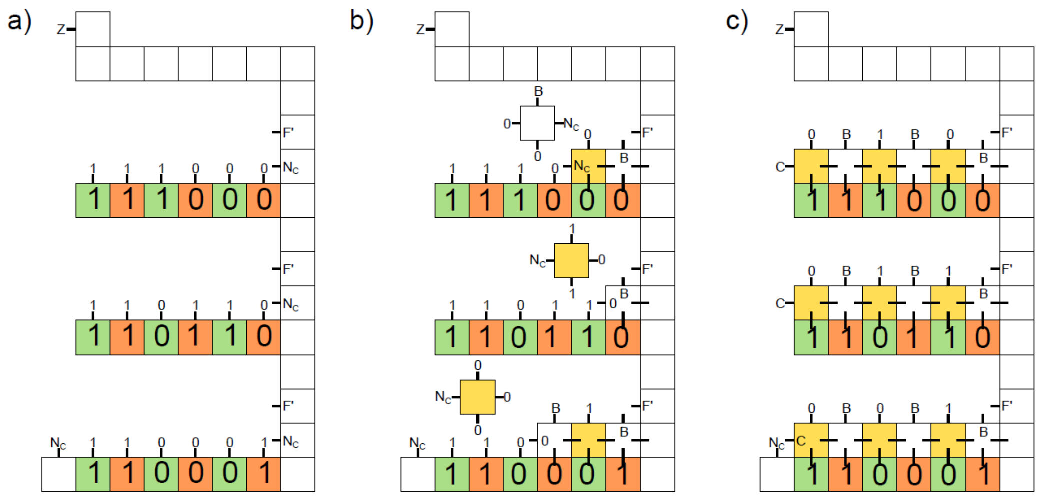


Figure 19 Step one, addition.

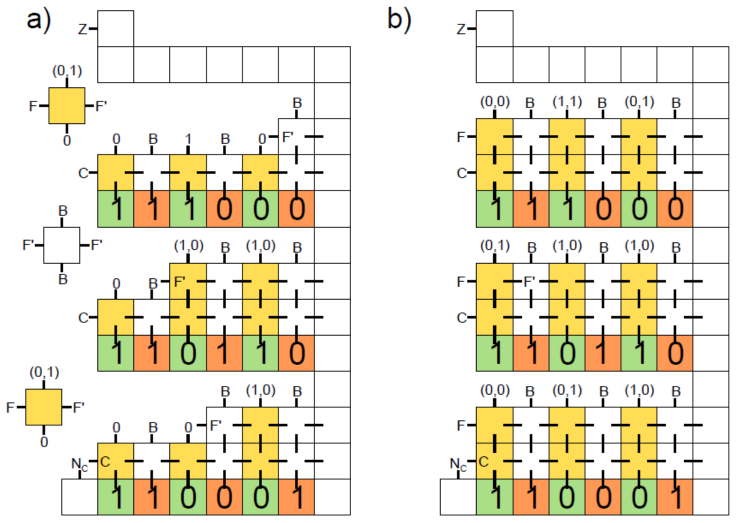


Figure 18 Step two, increment

Step Two: Increment. Then, start from the lowest bit of the second layer in each column. The result is now pretend the previous carry is 0, so by simply add 1 to the lowest bit, we then get the predicted result with a 1 coming in, as well as the corresponding output carry.

Step Three: Carry Propagation and Output. Start from the end of the bottom column, a set of propagation tiles will send the carry up. When a column received the carry from previous column, it will send the corrected output carry form current column to next column and update the result in the current column.

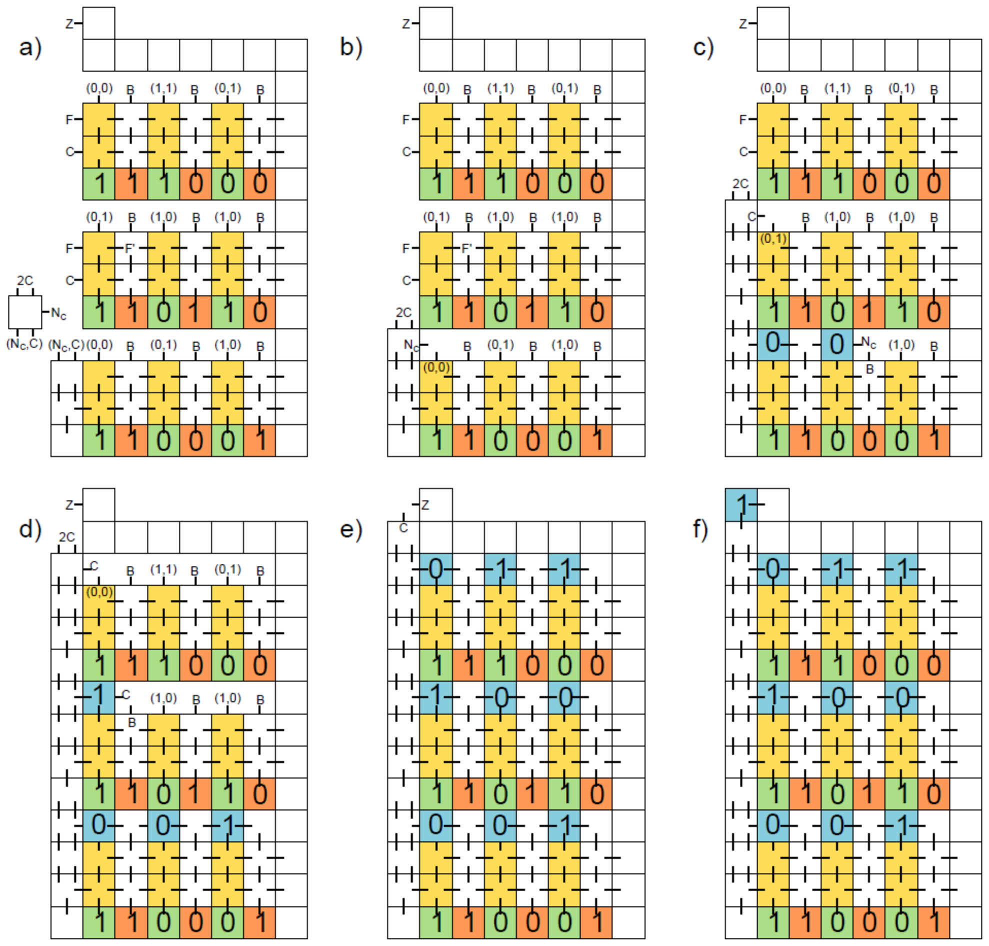


Figure 20 Step three, propagation

Time Complexity-Parallel Time

The First step, perform addition without the carry, each column happened independently, which will take $O\sqrt{n}$ parallel steps to finish. The second step, similar with the first step, each column will grow independently, so the run time is also $O\sqrt{n}$. Third step, the propagation will grow up and presumably waiting for each column finished, however, because each column will finish in parallel, so it doesn't have to take more than $O\sqrt{n}$ .time to reach the last bit. At the same time, the correction tiles will update the final results once the end of each column was reached by the propagation tiles, and the running time is also $O\sqrt{n}$. So, the total parallel runtime for this TAC is $O\sqrt{n}$.

$O(\log n)$Average Case,$O\sqrt{n}$ Worst Case Addition

This TAC combines the previous two TACs and reaches a result that consolidated both advantages. The idea is to preform $O(\log n)$average case addition under the frame of $O\sqrt{n}$ worst case addition, thus, the combined TAC will perform the $O(\log n)$average case addition, once any of the longest run of the carry decency is longer than $O\sqrt{n}$, the structure of $O\sqrt{n}$ worst case addition will ensure that the carry won't have to wait all the bits pairs but send the previous carry directly to next column.

Construction

In order to perform the $O(\mathrm{Log}\, n)$average case addition, any logistically neighboring bits pair need to stay close, which means we cannot simply use the arrangement used by $O\sqrt{n}$ worst case addition, because the distance between the highest bits pair in previous row and the lowest bits pair in current row is $O\sqrt{n}$. So, based on the structure of $O\sqrt{n}$ worst case addition, we fold the higher half of the bits pairs in each column back, so that any neighboring bits pair have a constant distance. Figure 22 a) shows an abstract diagram of $O\sqrt{n}$ worst case addition and Figure 22 b) shows the abstract diagram of $O(log\ n)$ worst case addition, and Figure 22 c) shows the abstract diagram of the combined addition.

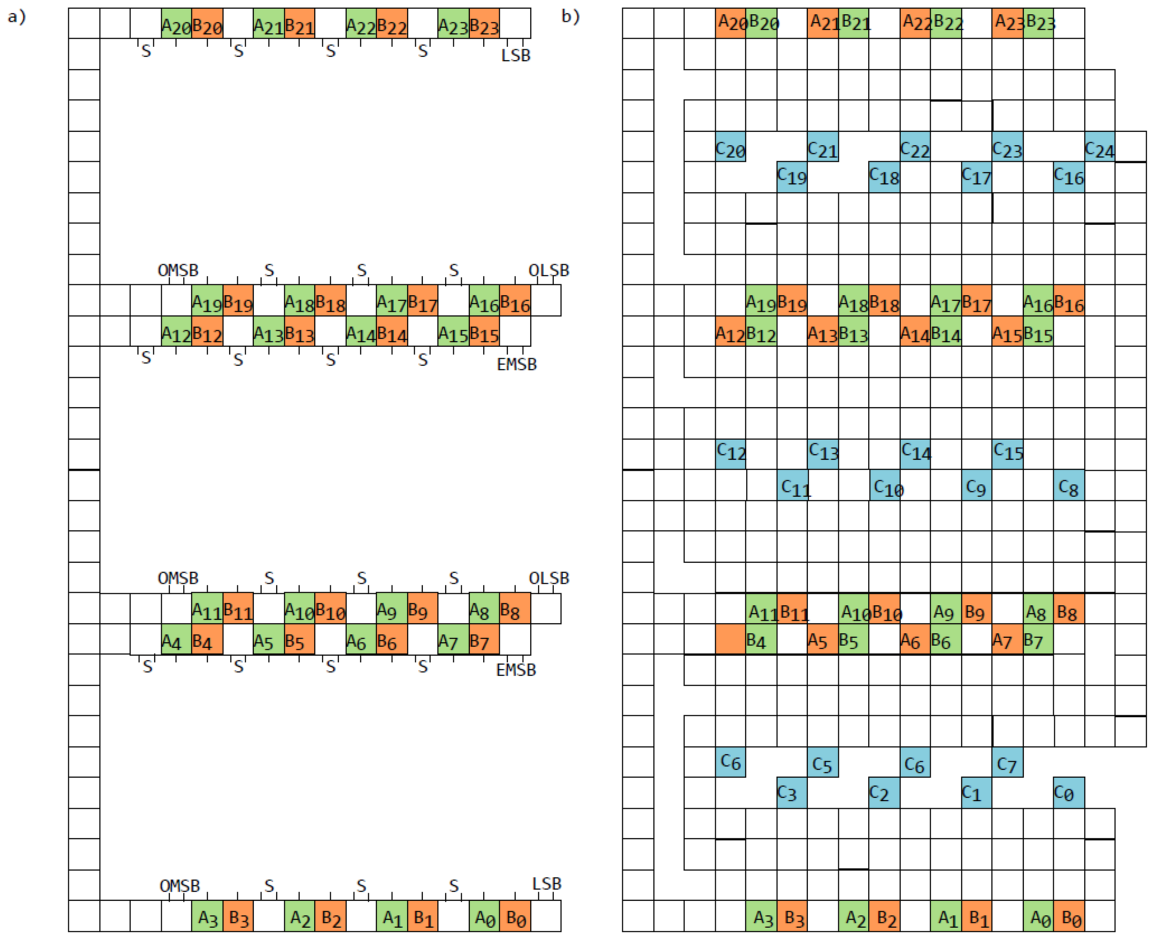
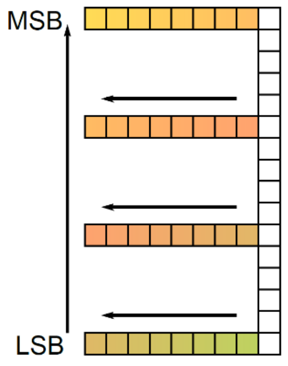
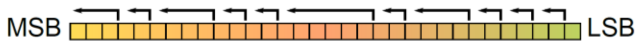


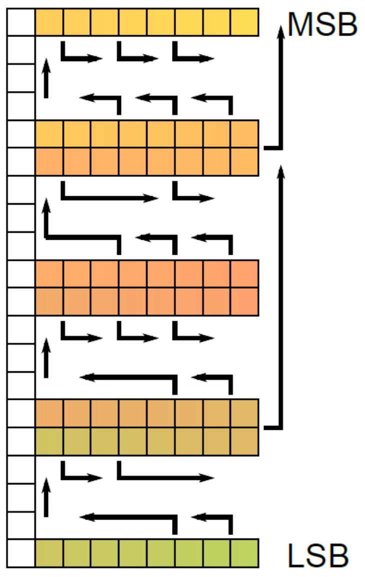
Figure 21 Left shows input template for the combined addition , right shows output template for the combined addition



a) $O\sqrt{n}$ worst case addition abstract diagram



b) $O(log\, n)$average case addition abstract diagram



c) Combined TAC abstract diagram

Figure 23 Arrows shows the computing directions

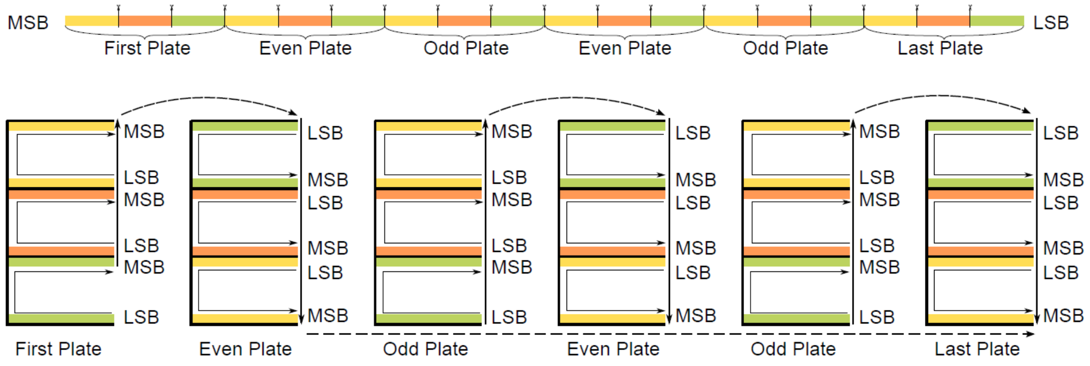
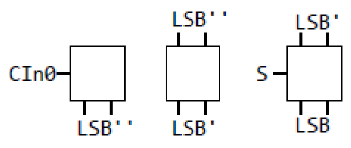
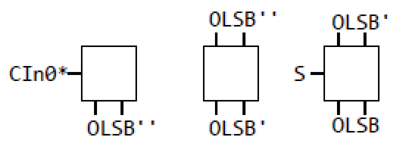


Figure 22.By using the 3D space, stack several layer of combined adding in 2D, to achieve the worst case $O(\sqrt[3]{n})$ and $O(log\, n)$ in average case adding

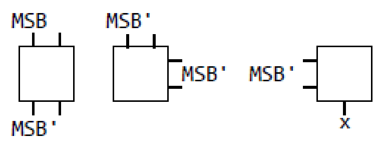
The fowling two pages shows the tile set for the combined addition TAC



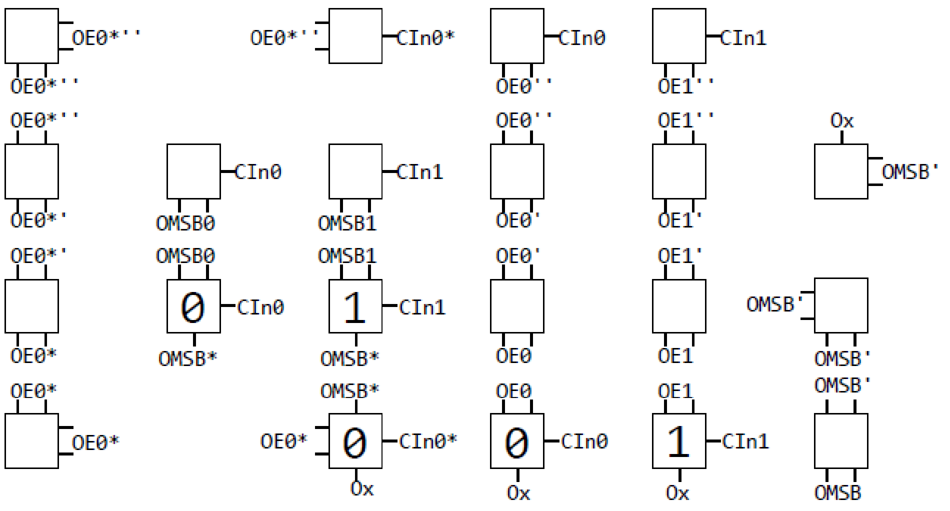
a) LSB Carry In



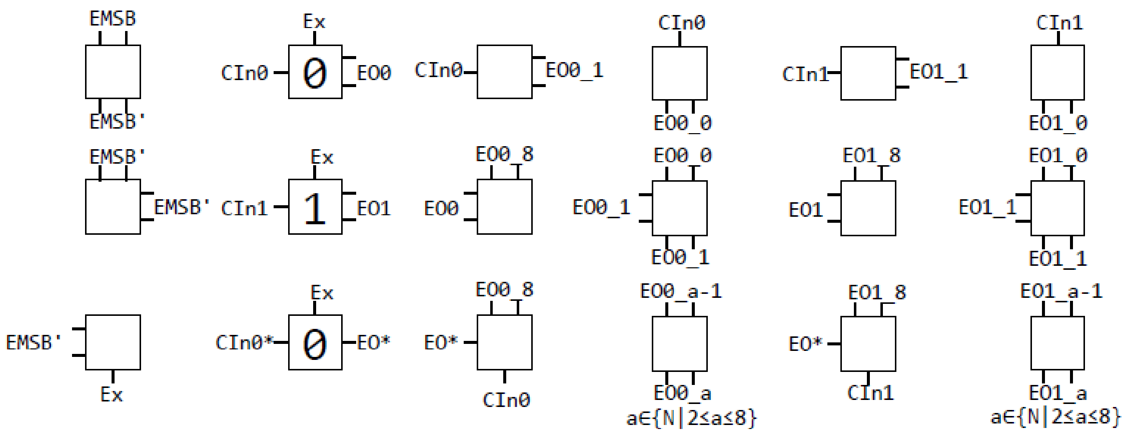
b) OLSB Carry In



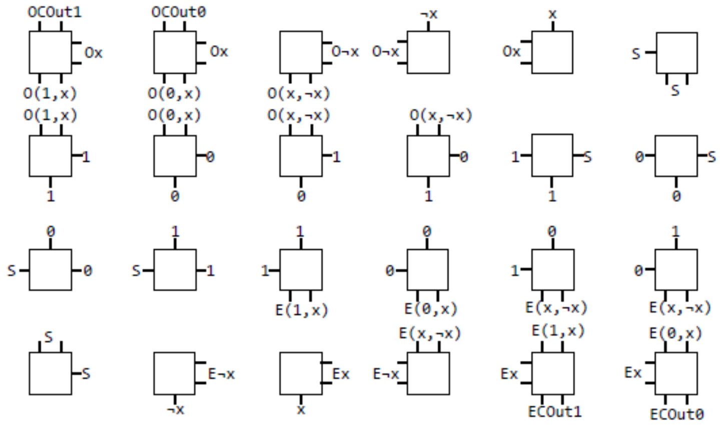
c) MSB Slot



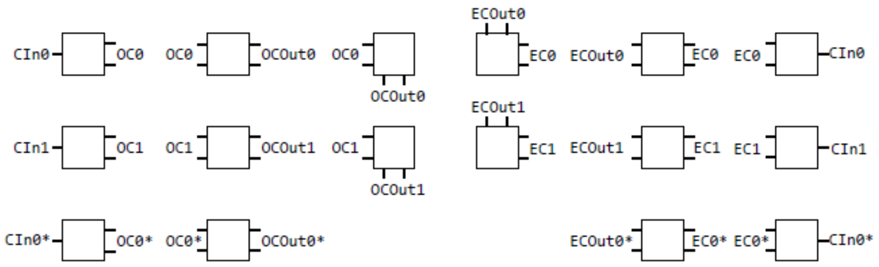
d)OMSB Slot



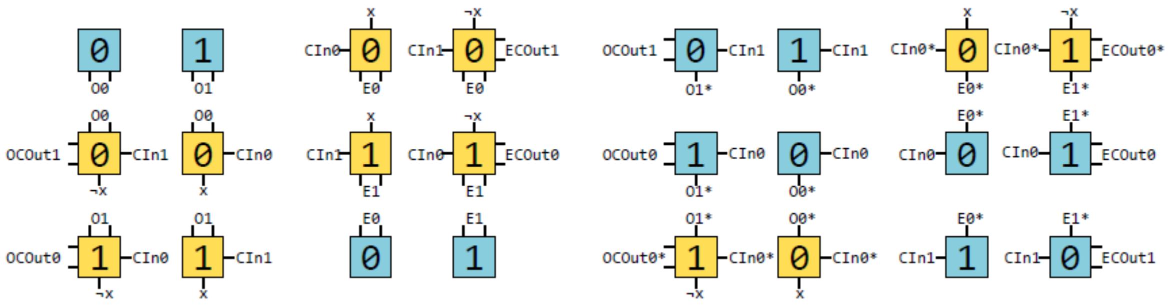
e) EMSB Slot



a) Addition

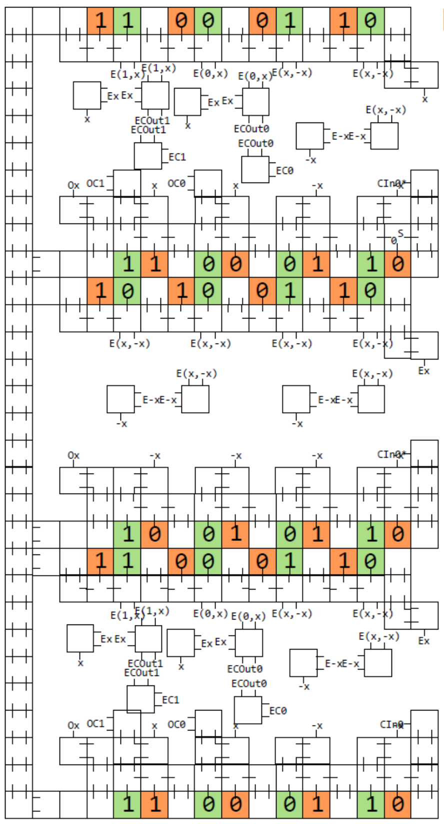


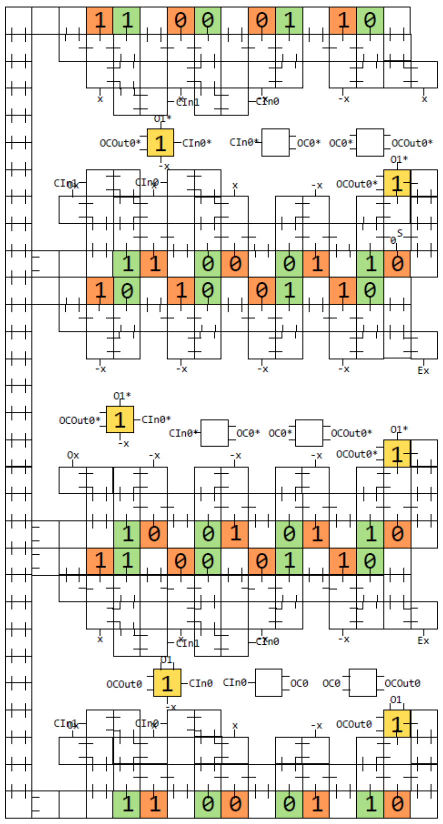
b) Carry Transfer

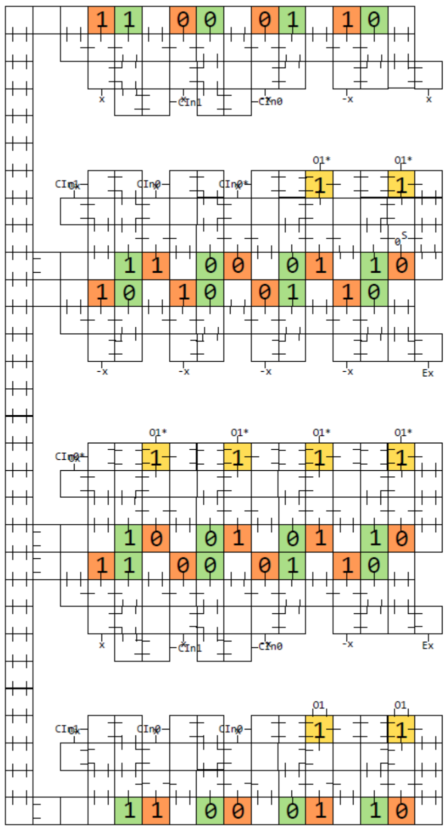


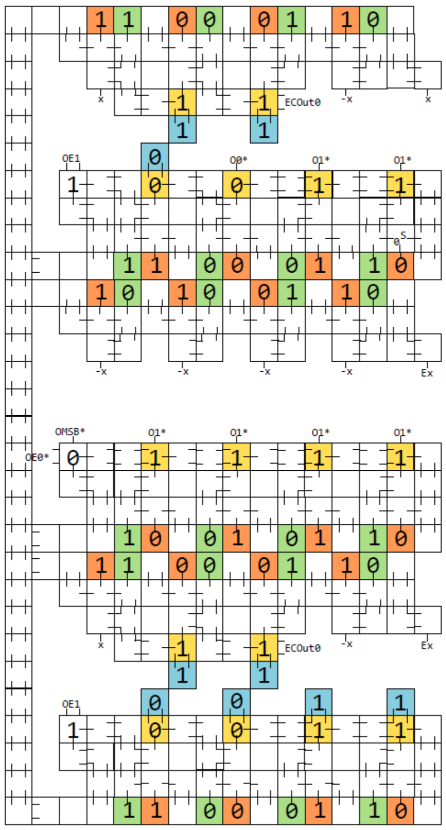
c) Print Answer

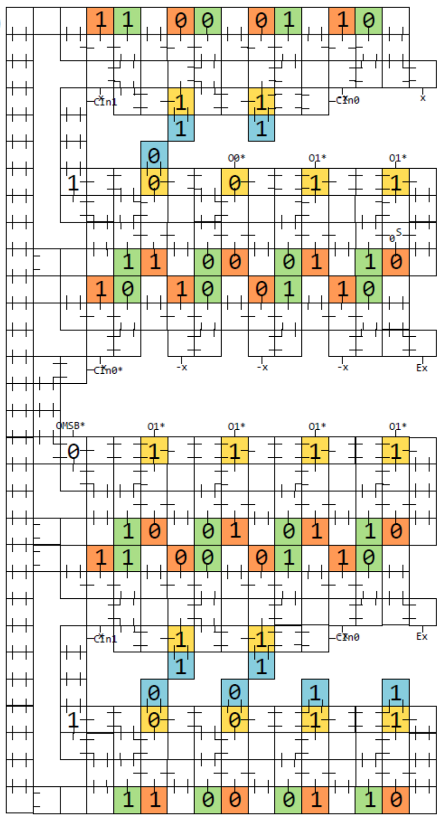
The fowling two and half pages shows an example of combined addition TAC

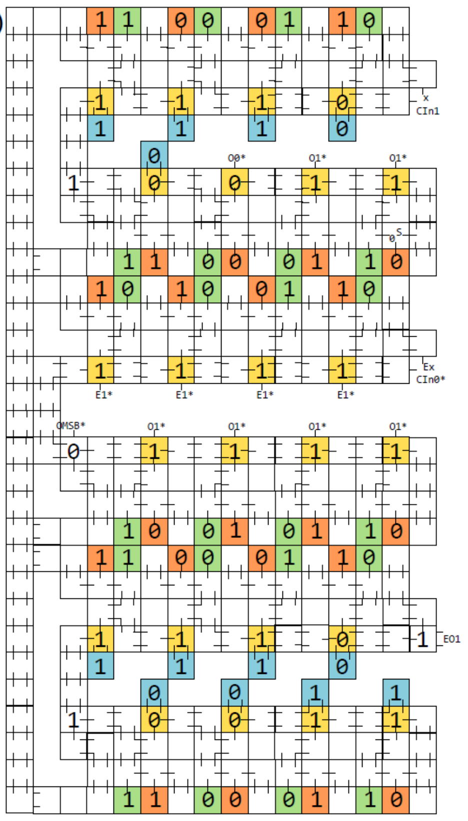
a)


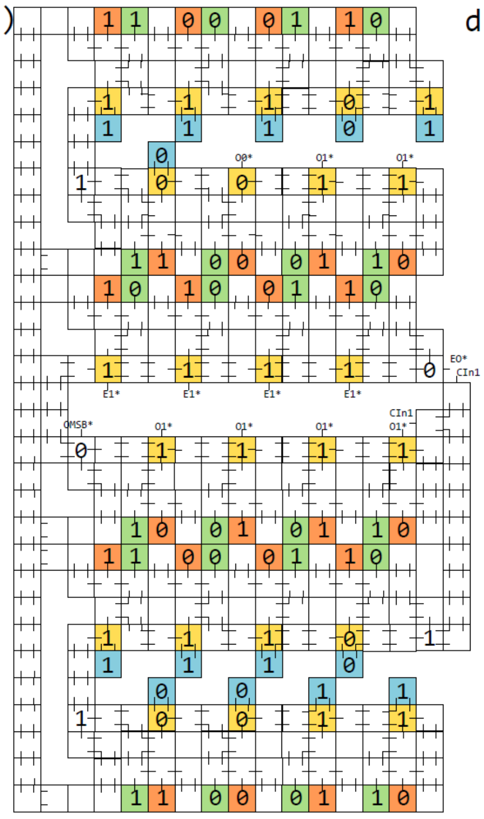
b)


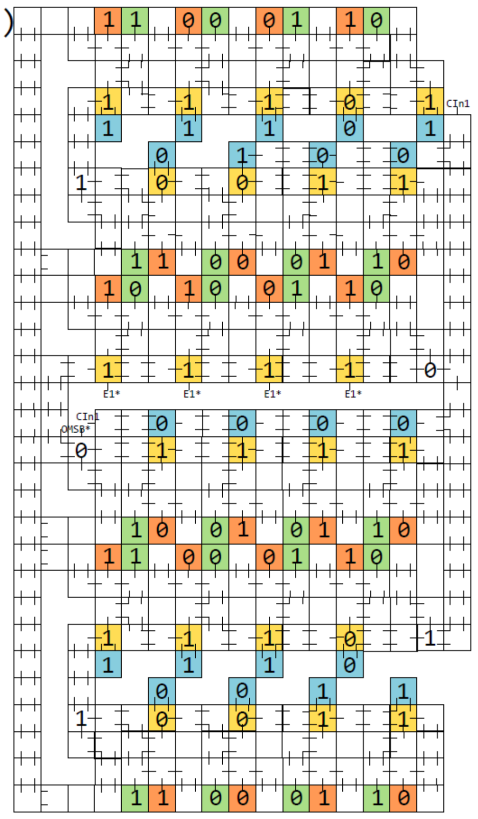
c)


d)


a)


b)


c)


d)


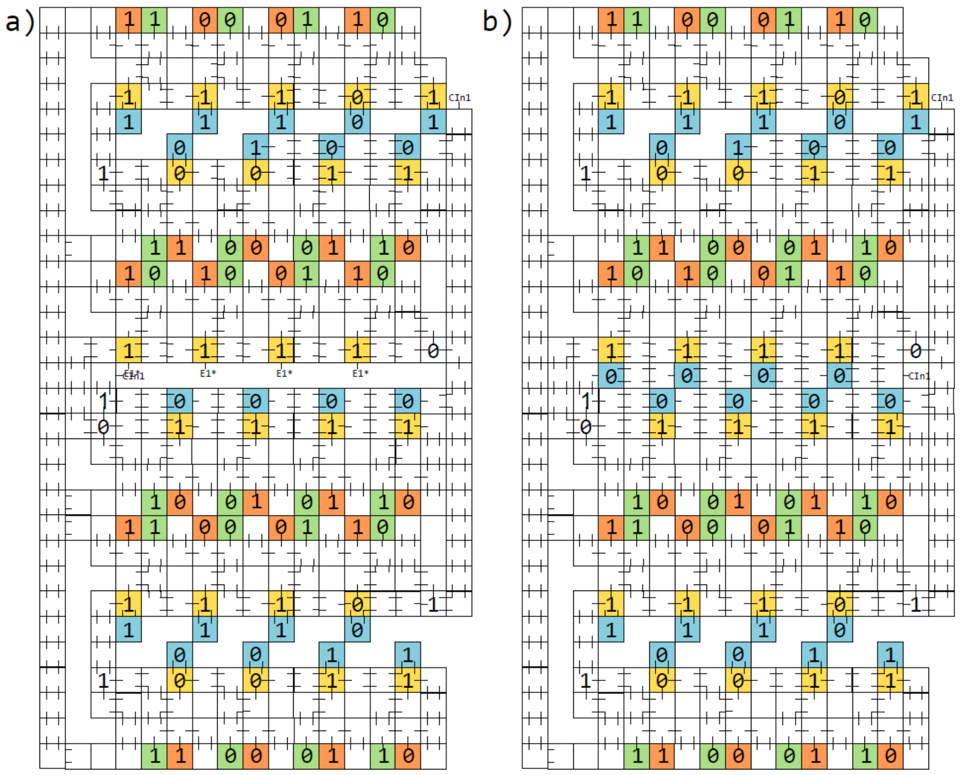


Figure b) Shows the final output

Time Complexity

By using the same principle of the $O(\log n)$average case addition to analyze the average case and using the principle of $O\sqrt{n}$ addition for the worst case, the time complexity is $O(\log n)$ in average case and $O\sqrt{n}$ in worst case.

Sub liner Multiplication

The two n-bits number multiplication is basically apply n times n-bits number addition, it is possible to recursively using the faster addition algorithm we just discussed and design a multiplication algorithm that could achieves a sub liner runtime. So, here we introduce a multiplication algorithm using ATAM model that achieves a $O(n^{\frac{5}{6}})$ runtime for multiply two n-bits numbers. Here is the basic idea, first, we put two n-bits numbers as input into an n by 2n square in the seed, the TAC will deploy the numbers to a cube. The cube has $\sqrt[3]{n}$ layers, each layer has $\sqrt[3]{n}$ columns, and each column contains $\sqrt[3]{n}$ numbers. After deploy the two n-bits numbers into this cube, the TAC will then perform the summation. First, all the numbers in each column will add together, and then summing of all columns in the same layer, finally, summing all three layers' results together. According to the result that the addition of two numbers could be expected as $O\sqrt{n}$ runtime in worst case, the running time of this $\sqrt[3]{n}$ by $\sqrt[3]{n}$ by $\sqrt[3]{n}$ cube could be expected as $O(n^{\frac{5}{6}})$.

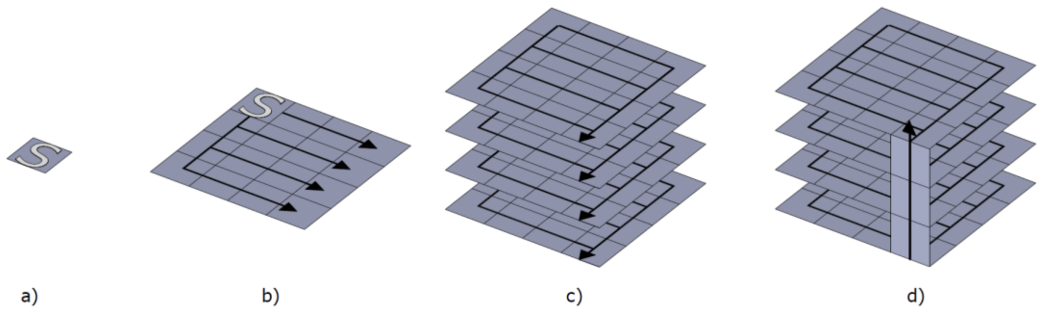


Figure 24 abstract figure for two numbers multiplication. The arrows shows the direction of computations

Construction

To describe this construction, I will use an example of multiplying two 64-bit numbers.

Vector Labeled Tile Types. For easily perform the description of this multiplication TAC, we use *Vector Label* to describe the binding process between tiles. Without showing the detailed glue types, we describe the TAC rules by showing the group of tiles that contains a certain element in the vector could attach to a certain assembly. The amount of tile types present by a Vector Labeled TAC is at most $O(l^d)$, which would be considered as a constant, where l is the number of label types and d is the vectors dimension.

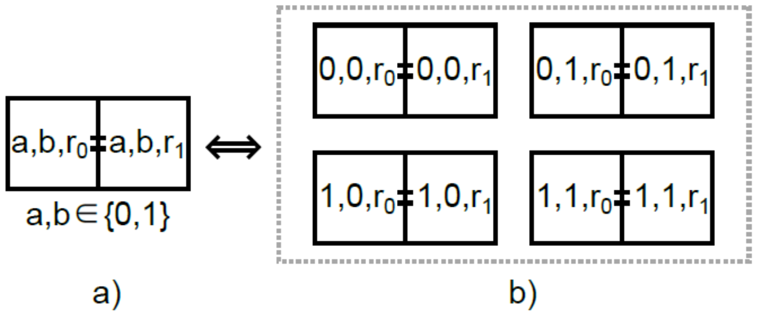
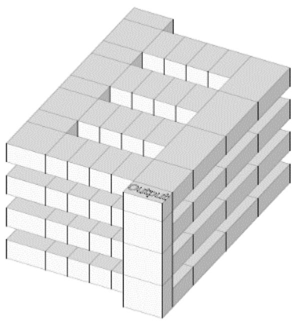


Figure 25 An example of vector labels

Input and Output Template. The result of two n bit numbers multiplication will have at most 2n-bits, so we put extra n-bits space in the left side of the seed. The right side of the seed shows the vector labeled input, A and B. Each bit of A and B shows as the first and second element in the vector in the same tile. The output of the algorithm is on the southernmost top surface, marked as Output.

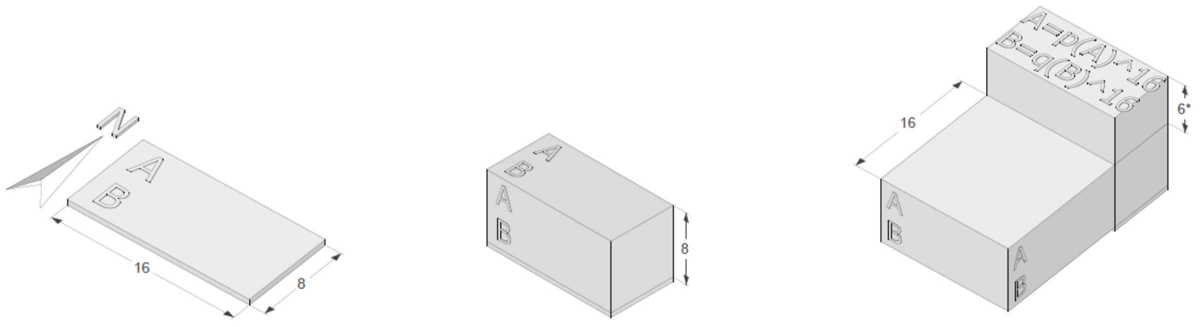
Step One: Deployment. The multiplication involves n numbers summation, for any number k_i within this n numbers $k_i = A\, b_i 2^i, i \in n$. According to orders of i from 0 to n, we

– –	– –	– –	– –	– –	– –	– –	– –	a56 b_{56}	a55 b_{55}^*	a40 b_{40}	a39 b_{39}^*	a24 b_{24}	a23 b_{23}^*	a8 b_8	a7 b_7^*
– –	– –	– –	– –	– –	– –	– –	– –	a57 b_{57}	a54 b_{54}	a41 b_{41}	a38 b_{38}	a25 b_{25}	a22 b_{22}	a9 b_9	a6 b_6
– –	– –	– –	– –	– –	– –	– –	– –	a58 b_{58}	a53 b_{53}	a42 b_{42}	a37 b_{37}	a26 b_{26}	a21 b_{21}	a10 b_{10}	a5 b_5
– –	– –	– –	– –	– –	– –	– –	– –	a59 b_{59}^*	a52 b_{52}	a43 b_{43}^*	a36 b_{36}	a27 b_{27}^*	a20 b_{20}	a11 b_{11}^*	a4 b_4
– –	– –	– –	– –	– –	– –	– –	– –	a60 b_{60}	a51 b_{51}^*	a44 b_{44}	a35 b_{35}^*	a28 b_{28}	a19 b_{19}^*	a12 b_{12}	a3 b_3^*
– –	– –	– –	– –	– –	– –	– –	– –	a61 b_{61}	a50 b_{50}	a45 b_{45}	a34 b_{34}	a29 b_{29}	a18 b_{18}	a13 b_{13}	a2 b_2
– –	– –	– –	– –	– –	– –	– –	– –	a62 b_{62}	a49 b_{49}	a46 b_{46}	a33 b_{33}	a30 b_{30}	a17 b_{17}	a14 b_{14}	a1 b_1
– –	– –	– –	– –	– –	– –	– –	– –	a63 b_{63}^*	a48 b_{48}	a47 b_{47}^*	a32 b_{32}	a31 b_{31}^*	a16 b_{16}	a15 b_{15}^*	a0 b_0



a) Input template b) Output template

Figure 26



a) Seed of multiplication.

b) The first copy block copy the seed input to both top and south

c) The copy to the south become the first block of the first column in the first layer. The copy to the up performs the shifting

Figure 27

divide every $O(\sqrt[3]{n})$ adjacent numbers into the same column, then we have every $O(n^{\frac{2}{3}})$.adjacent numbers in the same layer. So when we deploy the numbers, for any number in the same column, we simply copy both A and B from its previous neighbor and attach one more 0 after the lowest bit of A, and remove the lowest bit of B. For the first number in each column, we copy both A and B from its neighbor in the previous column, which is the first number of previous column, and attach or remove $\sqrt[3]{n}$ (in this case 4) bits relatively. This procedure is also similar with copy numbers to next layer, add or remove $n^{\frac{2}{3}}$.(in this case 16)bits relatively.

The orientation and seed is shown as figure 27 a). First step is to deploy the numbers in the seed to up and south. Figure 28 shows that for any input, it is possible to create a tile set to send the input information into two directions, top and side, within an n by n area in 2D. With same principle, a 2D shape can be copied to both up and side in 3D, and in our example to both up and south shown as figure 27 b)

Denote *p* is the function of attach one 0 after the input, *q* is the function that remove the last bit of input. The copy on the Up will then apply function of attach 16 bits of 0 after the lowest bit of A and remove 16 bits from B from the lowest bit (Figure 27 c)). Here $\mathrm{p(x)} = \mathrm{x} * 2$,

$q(x) = [\frac{x}{2}]$. We add or remove those bits by shifting the numbers. Figure 29 shows that for any input, it is possible to create a tile set to shift the input.

The copy on the south side (Figure 27 b) will then be copy to south and east (Figure 27 c). The south will be the head of next column and the east will be the first number in the first column. Then, similar to move the numbers to the next layer, the numbers moved to the next column will apply $p(A)^4$, and $q(B)^4$, shown as figure 30 c).

At the time we deploy the numbers in each column, we designed the TAC to perform the addition. The addition algorithm here is based on $O\sqrt{n}$ worst case addition. An example of two numbers multiplication, deploy as well as summation, shown as Figure35. The five elements in

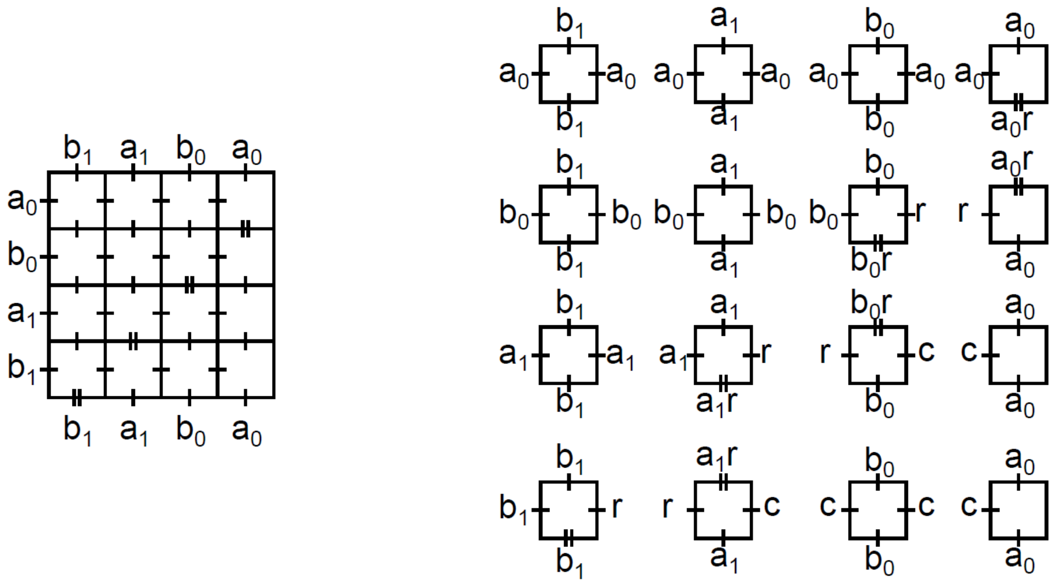


Figure 29 Tile set for copy the seed to two directions

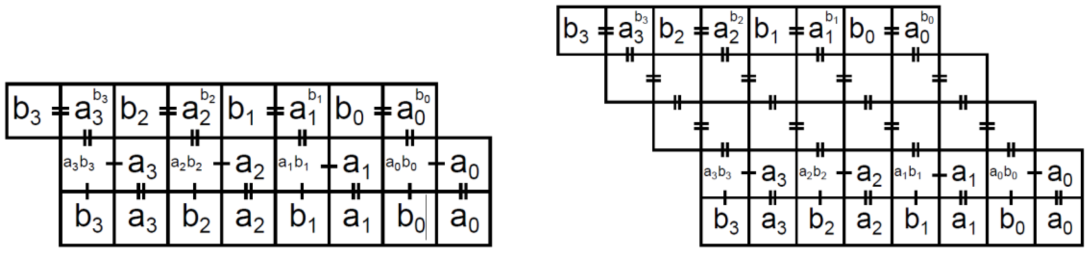
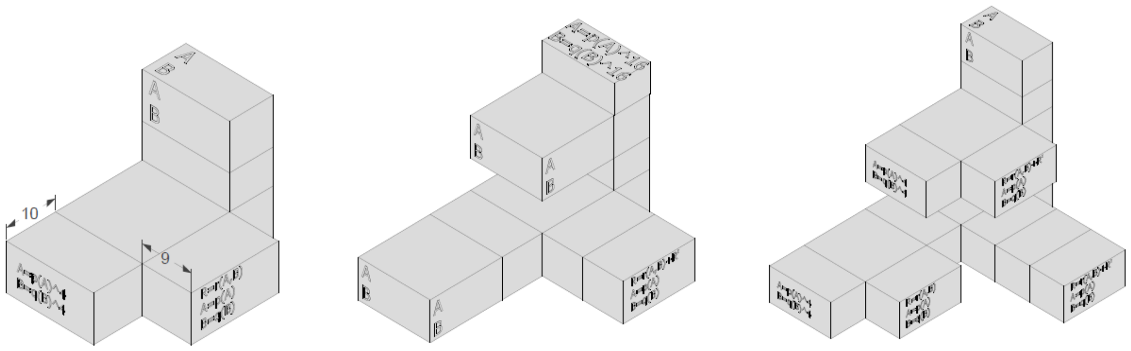


Figure 28 These two examples shows the way to shift the input to left

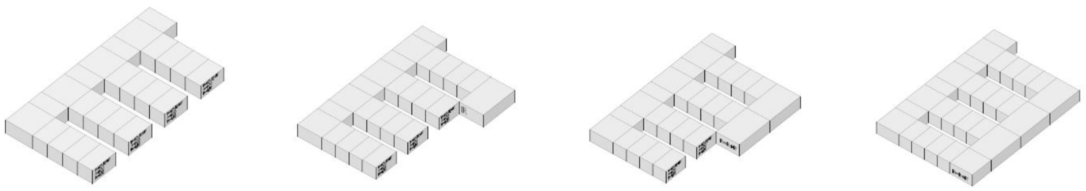
the vector Label are the current bit of A, the current bit of B, predicted result without previous carry, predicted result with previous carry equals to 1, and the current bit result. Same as the seed of Multiplication, the numbers are arranged in zigzag way.

After we got the result of each column, shown as Figure 36 a), we copy the result of first column and rotate to south and send it to its neighbor Figure 36 b). Note that we don't need A and B anymore, so we only copy the result without A and B. Using the reversed design of the rotate and copy box Figure 30, we merge the two input from two sides into one single output, and then using the same principle to sum all results in each columns together, figure 36 d). Same with the addition of each columns in same layer, the final results will be get by perform the summation of the result of each layer, shown as figure 37.



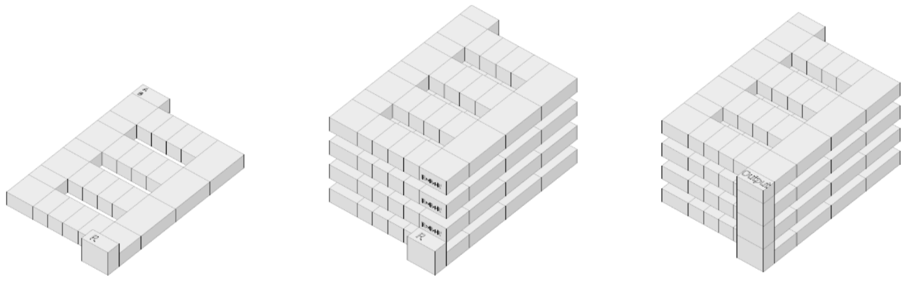
a) b) c)

Figure 32



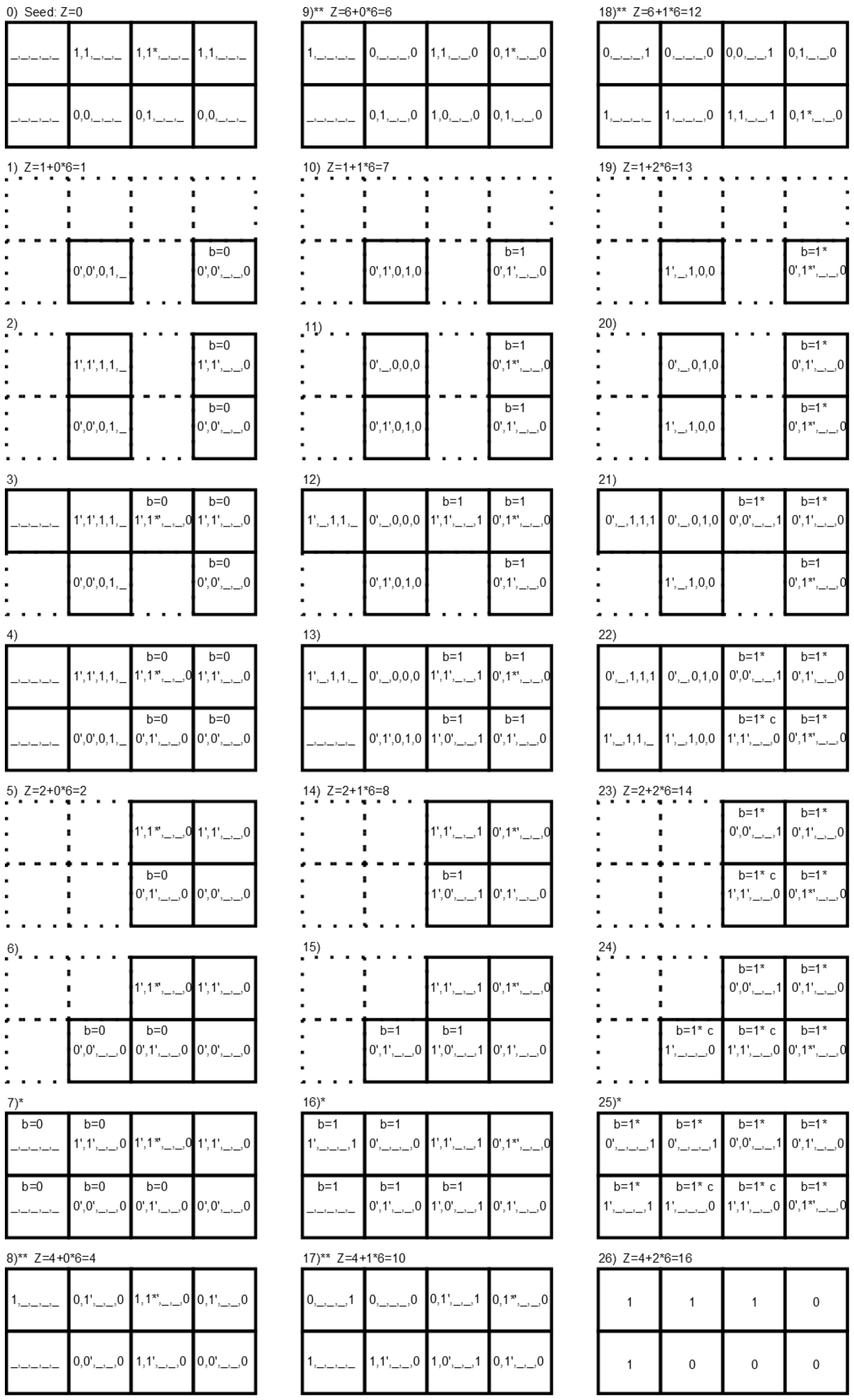
(a) (b) (c) (d)

Figure 31



(a) The first layer of multiplication (b) (c)

Figure 30



* The three newly attached tiles in step 7) 16) and 25) are attached in counterclockwise order.
** As described in the section of shifting tiles, the information in the tiles could be shifted simultaneously to its neighbor in 4 steps, and took 2 layers of tiles space beyond the origional tiles.

Figure 33

CHAPTER IV

PATTERNS REPLICATION

The automated replication process in DNA Self-Assembly is a very interesting topic. It is the simulation of self-replication in the nature world, and also be interested by manufactures. In this chapter, I will introduce the problems of patterns self-replication by using the Signal Tiles Self-Assembly Model (STAM). By solving this problem, we provide theoretical support of using signal tiles to replicate a patterned rectangle with a fast replication speed. The solution we come up is a template-directed fast growing replicated system, in particular, the algorithm we performed is by using a rectangular template with arbitrarily 0 and 1 patterns on it, and the system will create copies of template with an exponential growth. When describe this algorithm, I will mostly focus on the high-level sketch of the processes of how the system is work by showing the stages of the signal tiles perform.

Exponential Replication of 2D Patterns in 2D Space

First, let me introduce the replication of 2D patterns in the 2D space. The reason I introduce this problem first is because the problem of replicate 2D patterns in 3D space is relatively straightforward, intuitively we can simply create copies upon the 2D input template and get job done, also, the algorithms of 3D patterns replication in 3D space can be extended from the algorithm we described in 2D space.

Definitions

Patterns. Let *l* be a set of labels. A pattern is the mapping of the coordinates in a shape to the set *l*.

Exponential Replication. A replication system is an exponential replication system, if the system can replicate a patterned shape such that the patterned shape can then generating a final product as well as itself.

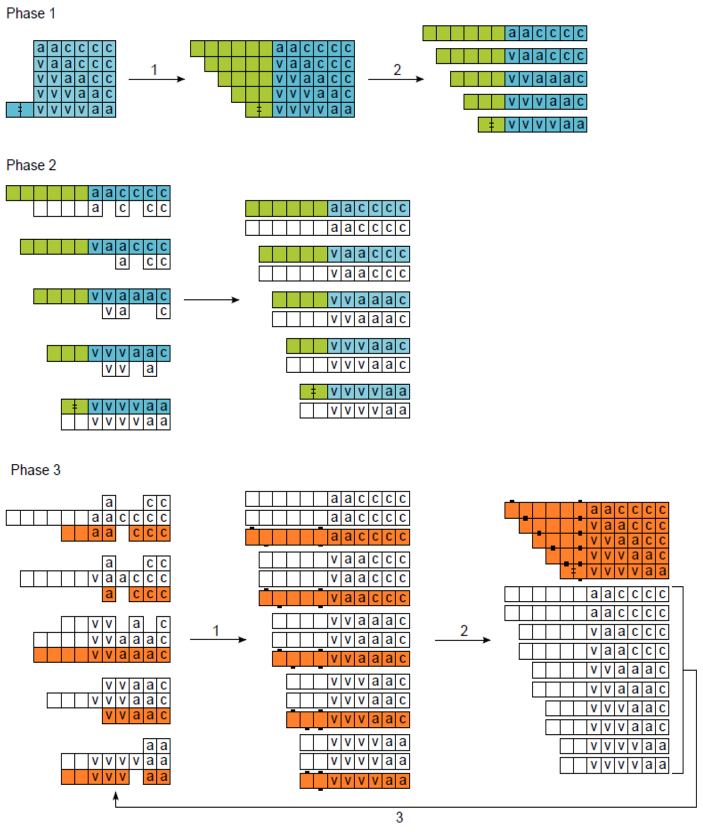


Figure 34 Over view of exponential 2D pattern replication.

Before formally describe the system, here is a brief introduce of how the mechanism for 2D patterns replication works. Figure 34 shows the three phases overview of the replication system. The input shows as the patterned rectangle shape in the left of phase 1. In phase 1, an inverted staircase (shown as the green tiles) cooperatively grows along the west edge of the input patterned shape. Then, the assembly dissembled into single tile width rows. In second phase, each rows became a template for the production of a *non-terminal replicates (ntr)*, shows as the white patterned tiles. In the third phase, those *ntr* become the template and produce the *ntr* and *terminal replicates (tr)*, shows as orange tiles. The *tr* will then attached to each other and create a copy of the original input pattered assembly and the *ntr* will keep on creating copes of *tr* and *ntr*.

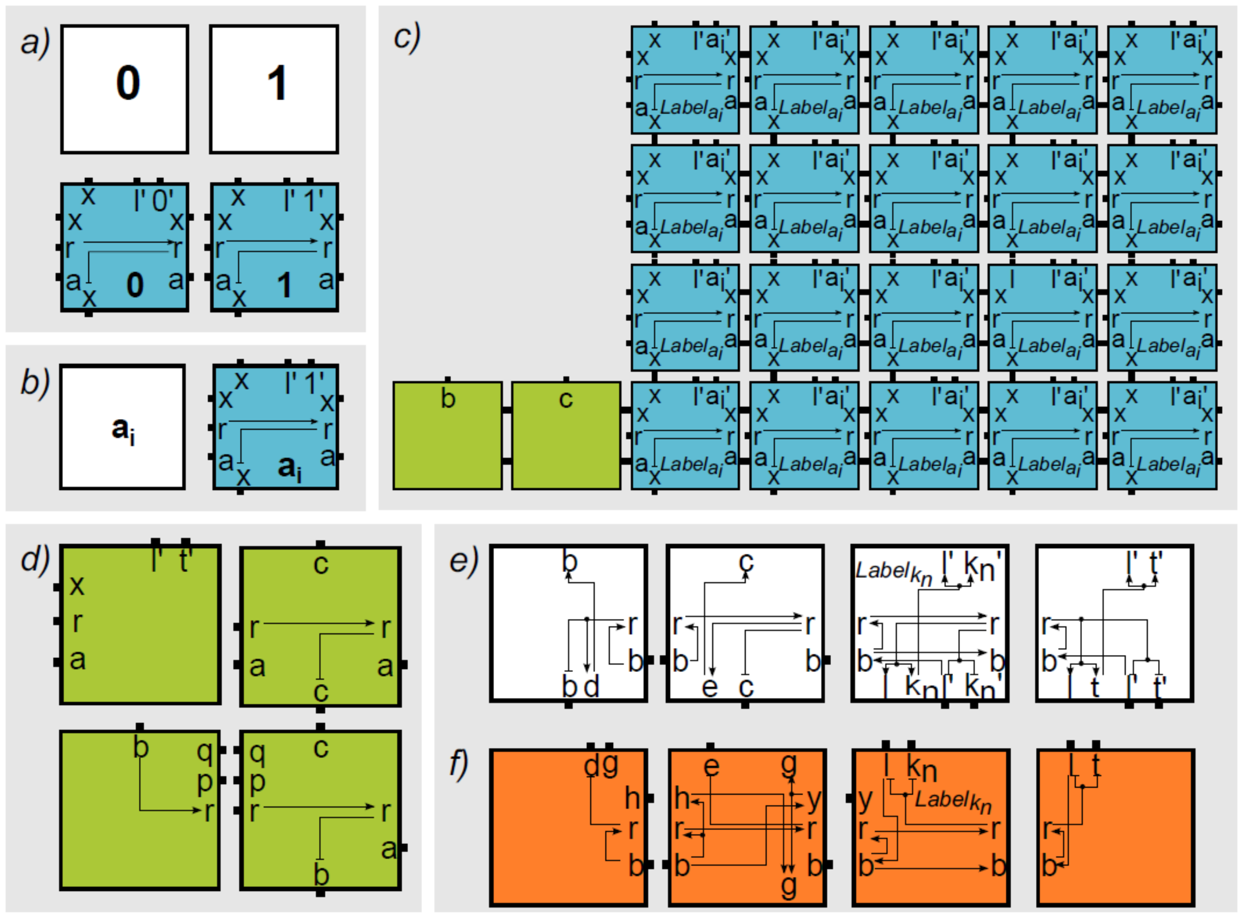


Figure 35 Tiles set for the replication system

First Step, Template Disassembly. In order to let dissembled fragment to connect together under the original order, we firstly let the template grow a stairs structure on the side of the template. The function of a stairs structure is for later reassemble of the fragment in the correct order; the fragment can reattach to another one only if the gap between two glues is match, otherwise, at most only 1 glue can overlap each other, but under the environment temperature of 2, they cannot attach. In each rows, after the leftmost tile on the end of stares attached, it will send a signal *r* to its right. The tiles received the *r* signal will pass it through, and check if the next tile can receive this *r* signal. Once the *r* signal is been received by next tile, the current tile will deactivate the glue *x* on its bottom, which is used to connect the previous row. This signal keep on sending until accepted by the tile attached on the rightmost of the row, to show that this signal has reached the end, thus, all the glue *x* on the bottom of each tile will receive a deactivate signal and then detach the row on its bottom.

First Generation of Replicates. After a row detached from the original assembly, a labeled glue on the north of each tiles in the row now been exposed. Then the tiles involved *ntr* (tiles in white) start to attach the template (yellow) rows. After the left most *ntr* tile attached the template row, it send a signal to its right. Similar with the process of template detachment, the *ntr* tiles will detached from the template after the right most tile received the signal, which also means all *ntr* tiles in the same row are connected.

Exponential Replication and Reassembly. In this step, the north side of the *ntr* tiles will perform exactly the same process during the First Generation of Replicates, and create copies of *ntr*. On the south side, similar process is performed and terminal replicate (yellow) row is created. After the *tr* rows detached from *ntr*, *tr* rows will attach to other *tr* rows that the gap between two glues on the stare structures are match. tr rows

,

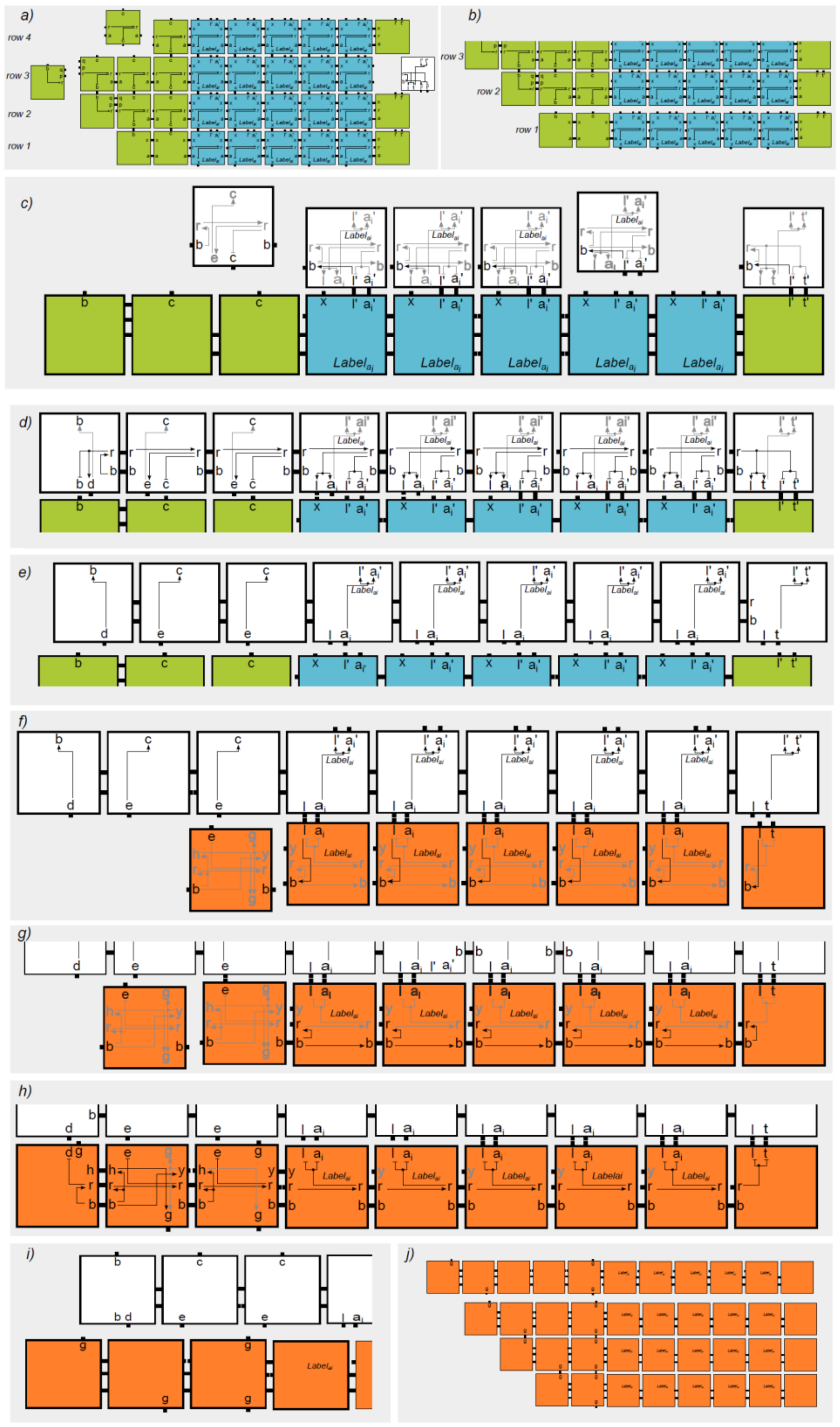


Figure 36 Detailed process of replication

REFERENCES

Erik Winfree, Algorithmic self-assembly of DNA, Ph.D. thesis, California Institute of Technology, June 1998.

David Yu Zhang Georg Seelig Dynamic DNA nanotechnology using strand-displacement reactions, Nature Chemistry 3, 103–113 (2011) doi:10.1038/nchem.957

Jennifer E. Padilla, Matthew J. Patitz, Raul Pena, Robert T. Schweller, Nadrian C. Seeman, Robert Sheline, Scott M. Summers, Xingsi Zhong: Asynchronous Signal Passing for Tile Self-assembly: Fuel Efficient Computation and Efficient Assembly of Shapes. UCNC 2013: 174-185

Zachary Abel, Nadia Benbernou, Mirela Damian, Erik Demaine, Martin Demaine,Robin Flatland, Scott Kominers, and Robert Schweller, Shape replication through self-assembly and RNase enzymes, SODA 2010: Proceedings of the Twentyrst Annual ACM-SIAM Symposium on Discrete Algorithms (Austin, Texas), Society for Industrial and Applied Mathematics, 2010.

Yuriy Brun, Arithmetic computation in the tile assembly model: Addition and mul-tiplication, Theoretical Computer Science 378 (2007), 17-31.

Nathaniel Bryans, Ehsan Chiniforooshan, David Doty, Lila Kari, and ShinnosukeSeki, The power of nondeterminism in self-assembly, SODA 2011: Proceedings ofthe 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2011,pp. 590{602.

Harish Chandran, Nikhil Gopalkri shnan, and John H. Reif, The tile complexityof linear assemblies, 36th International Colloquium on Automata, Languages and Programming, vol. 5555, 2009.

Matthew Cook, Yunhui Fu, and Robert T. Schweller, Temperature 1 self-assembly:Deterministic assembly in 3d and probabilistic assembly in 2d, Proceedings of theTwenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA2011 (Dana Randall, ed.), SIAM, 2011, pp. 570{589.

David Doty, Randomized self-assembly for exact shapes, SIAM Journal on Com-puting 39 (2010), no. 8, 3521{3552.

David Doty, Jack H. Lutz, Matthew J. Patitz, Robert Schweller, Scott M. Summers,and Damien Woods, The tile assembly model is intrinsically universal, FOCS 2012:Proceedings of the 53rd IEEE Conference on Foundations of Computer Science,2012.

David Doty, Matthew J. Patitz, Dustin Reishus, Robert T. Schweller, and Scott M.Summers, Strong fault-tolerance for self-assembly with fuzzy temperature, Proceed-ings of the 51st Annual IEEE Symposium on Foundations of Computer Science(FOCS 2010), 2010, pp. 417{426.

Bin Fu, Matthew J. Patitz, Robert Schweller, and Robert Sheline, Self-assembly with geometric tiles, ICALP 2012: Proceedings of the 39th International Collo-quium on Automata, Languages and Programming (Warwick, UK), 2012.

Ming-Yang Kao and Robert T. Schweller, Randomized self-assembly for approxi- mate shapes, International Colloqium on Automata, Languages, and Programming, Lecture Notes in Computer Science, vol. 5125, Springer, 2008, pp. 370{384.

Mark Schilling, The longest run of heads, The College Mathematics Journal 21(1990), no. 3, 196{207.

Robert Schweller and Michael Sherman, Fuel ecient computation in passive self-assembly, Proceedings of the Annual ACM-SIAM Symposium on Discrete Algo-rithms, 2013.

ISAAC D. Scherson, and Sandeep Sen: Parallel Sorting in Two-Dimensional VLSI Models of Computation. IEEE TRANSACTIONS ON COMPUTERS, VOL. 38, NO. 2, FEBRUARY 1989.

Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin, Efficient active self-assembly of shapes, Manuscript (2012).

Alexandra Keenan, Robert Schweller, Xingsi Zhong, Exponential Replication of Patterns in the Signal Tile Assembly Model, DNA Computing (DNA19).

APPENDIX A

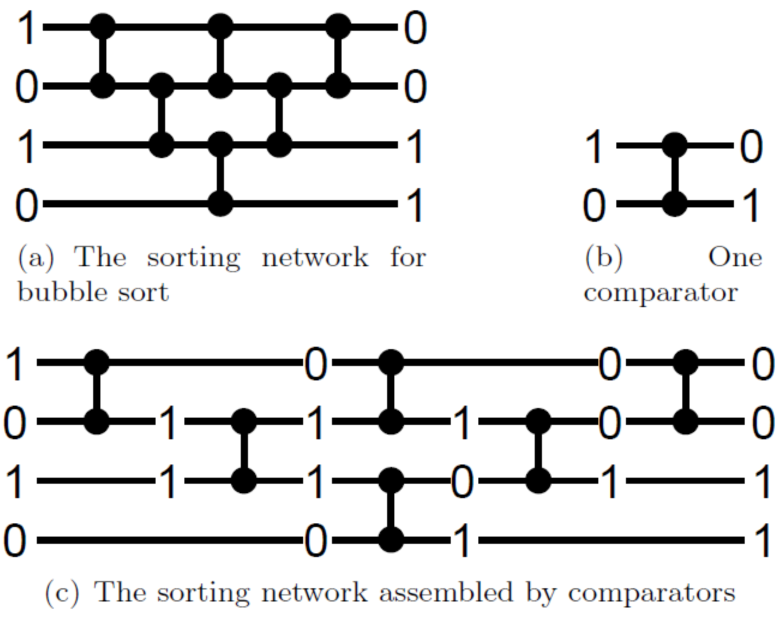
APPENDIX A

PARALLEL SORTING

The tile assembly model is particularly suited to a comparison network model of computation because many comparison operations may be performed simultaneously. The comparison network model is an method that relies on comparison elements, two-input two-output devices which the computes the minimum and the maximum of the two inputs The input of a sorting network is a bunch of numbers arranged in vertical on its left. Right next to the numbers, the sorting network use horizontal lines to track each numbers and use vertical lines to represent each comparison. And the output of a sorting network is on its right. In this paper, the algorithms we are using is bubble sorting algorithm.

Comparator. Comparator is the basic unit in a sorting network. A comparison network contains two lines and a comparator. Two lines on the left of the comparator represent the two input numbers, and the two lines on the right represent the output.

Comparison Networks And Sorting Networks. Comparison Networks is a set of comparators connected by wires. Sorting Networks is a comparison network that the output numbers of that comparison network is in sorted order.

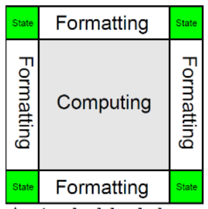

(a) The sorting network for bubble sort

(b) One comparator

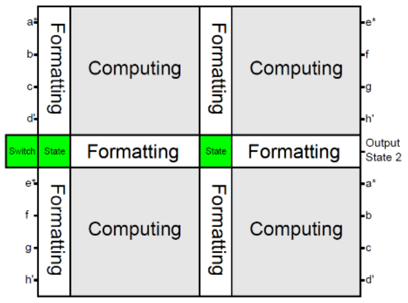
(c) The sorting network assembled by comparators

Figure 37 A sorting network

Black Box Model. The Black Box Model can be used to hide the details of the individual tiles but only focus on the functions of each black box. By using this model, we will have more time to focus on the design of complex algorithms ignoring the design of every tile upfront. The Black Box Model has a consistent format with some input and outputs sides. The general shape of a single black box unit is shown in Figure 38(a). Each black box unit has a central processing core called a computing box. Between every two computing box units there is some space in which the output from the previous computing box becomes formated to the input of the next computing box. These black box units also need a way to ensure the start and the end of a computation, the input and output stages respectively. This i/o stage information is held in the corners of each black box unit. The input stage of some black box may rely on one or more output stages allowing for cooperative growth.

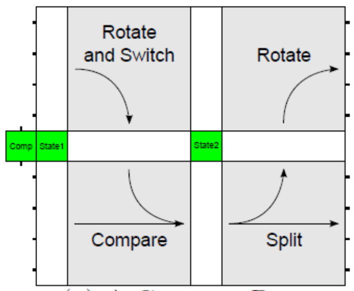


(a) A single black box unit.

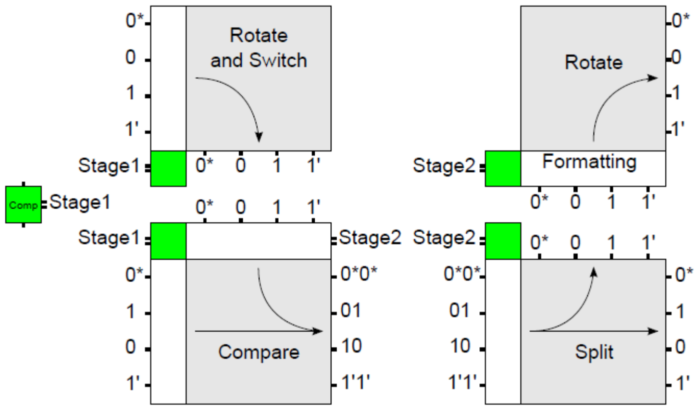


(b) A black box with multiple combined black box units.

Figure 38 Examples of black boxes.



(a) A Compare Box.



(b) Indicated by comparing 0011 and 0101, here shows the transformation and transmission of numbers.

Figure 39 The high level design of a compare box.

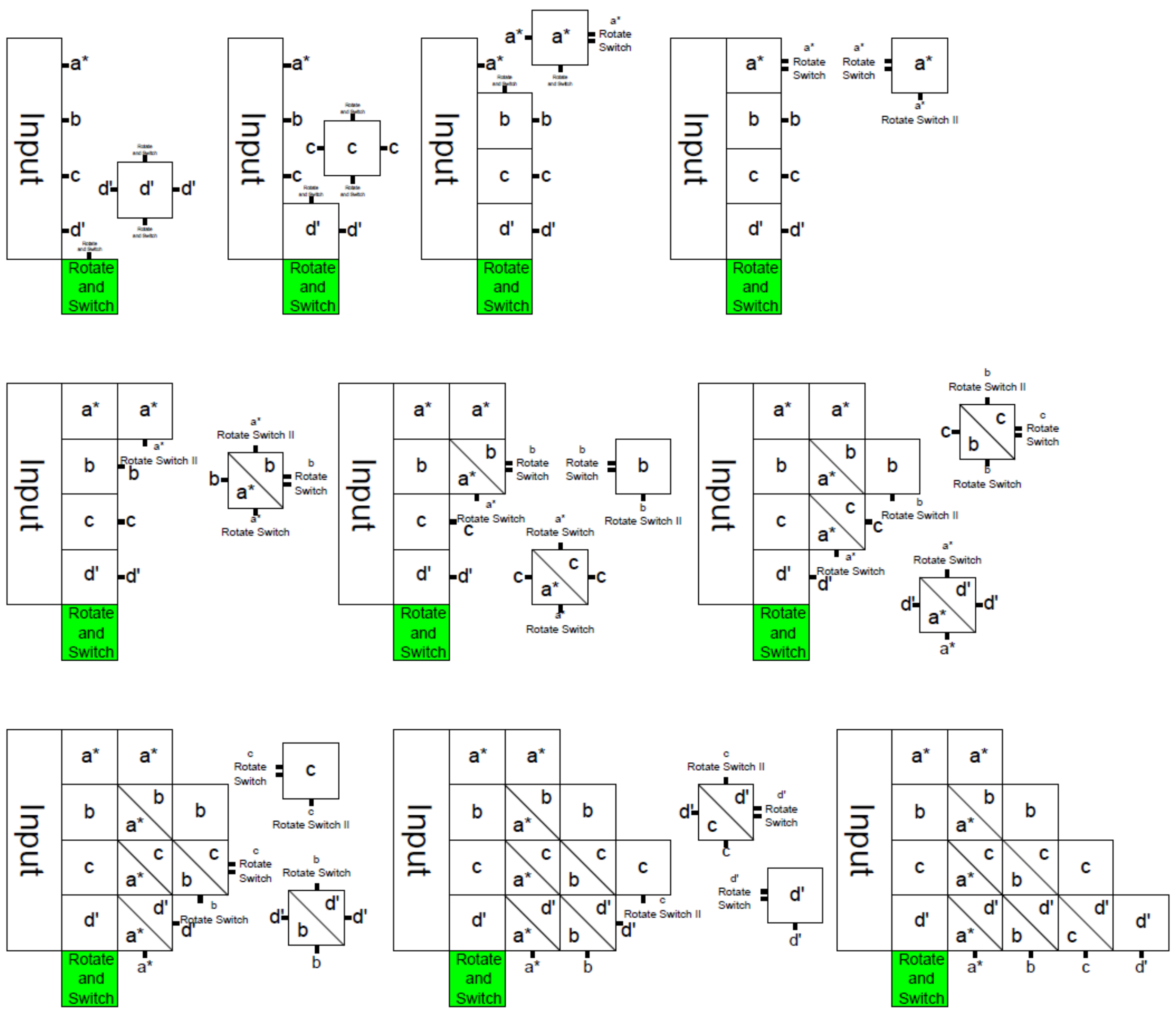


Figure 40 Design and example of Rotate And Switch Box.

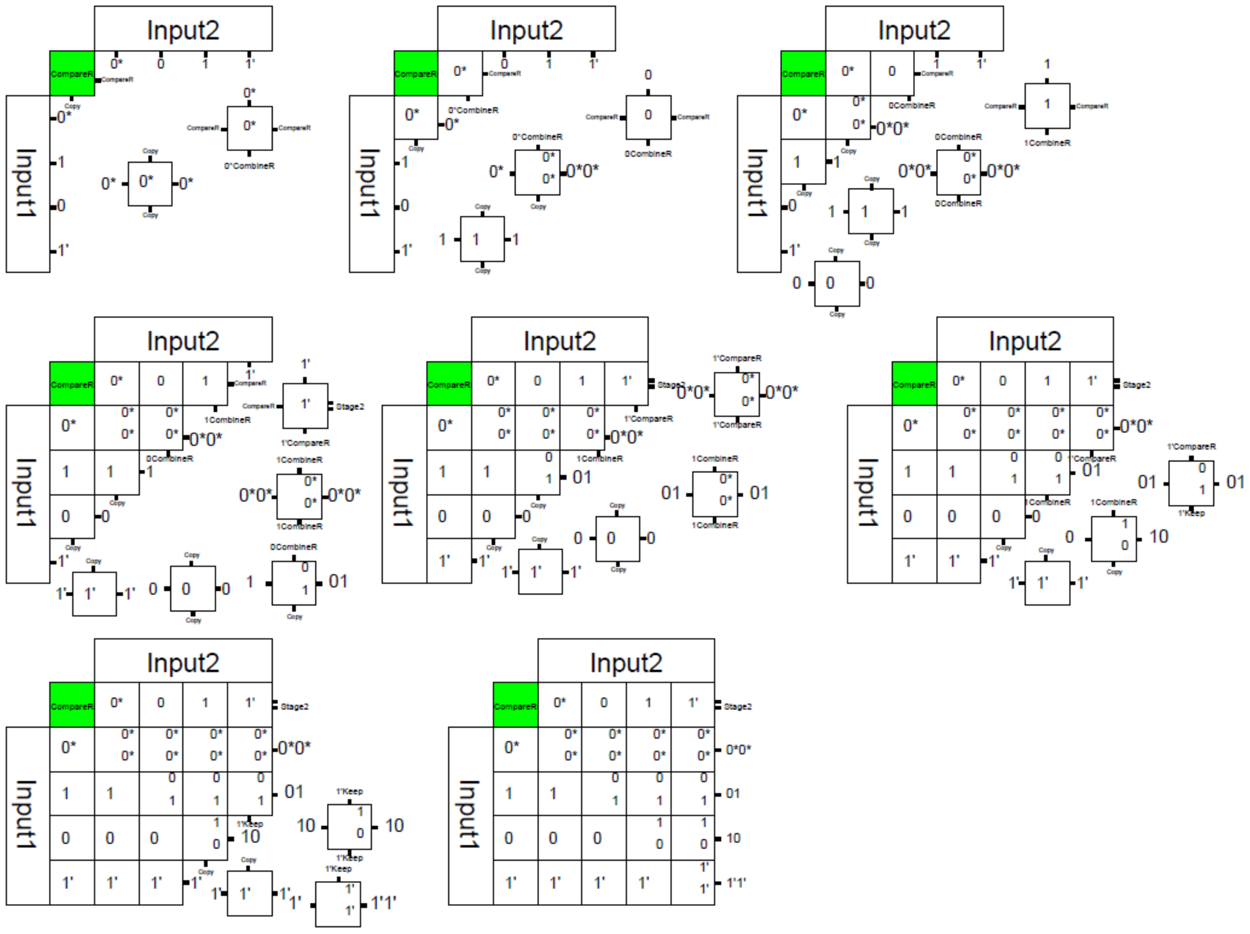
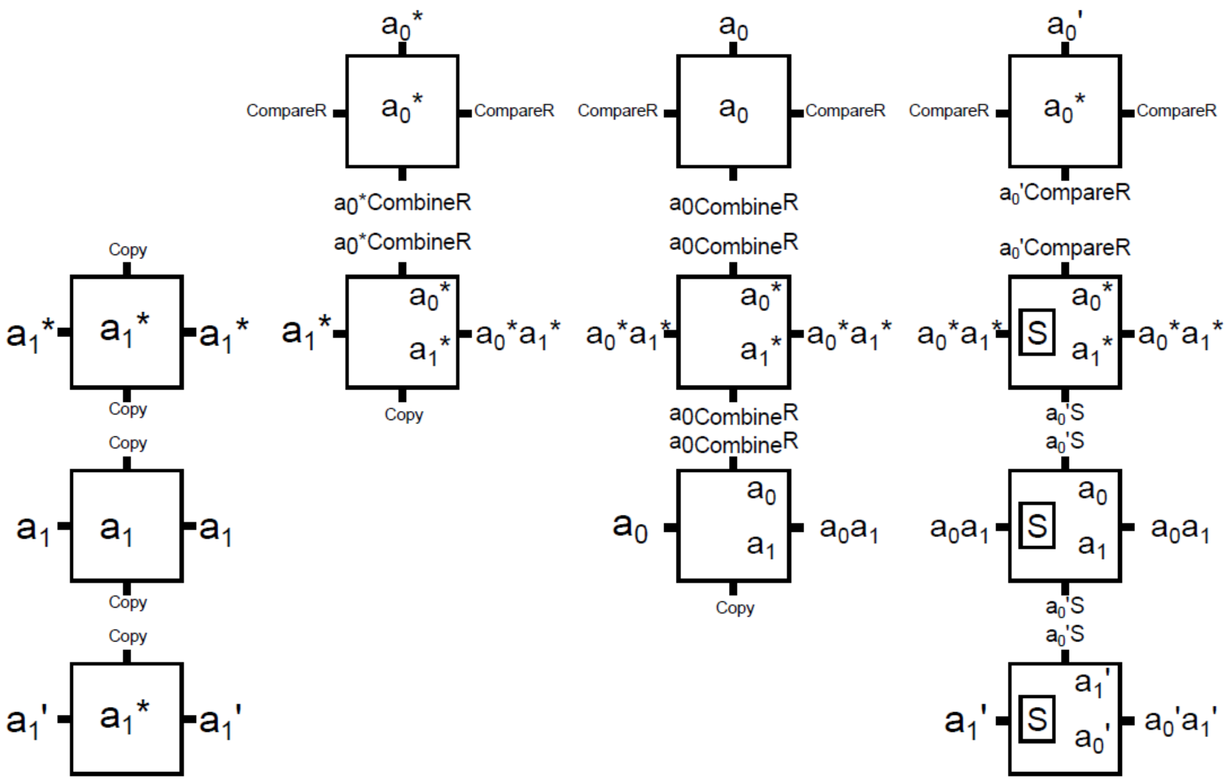


Figure 41 Design and example of Small Compare Box.



Note:

If a1*=a0* in input, then output S=CompareR; Keep the value of a1* and a0*
If a1*>a0* in input, then output S=Keep; Keep the value of a1* and a0*
If a1*<a0* in input, then output S=Switch;Switch the value of a1* and a0*
If S=CompareR in input,{
if a1*=a0* in input, then output S=CompareR; Keep the value of a1* and a0*;
if a1*>a0* in input, then output S=Keep; Keep the value of a1* and a0*;
if a1*<a0* in input, then output S=Switch; Switch the value of a1* and a0*.}
If S=Keep in input, then output S=Keep; Keep the value of a1* and a0*.
If S=Switch in input, then output S=Switch; Switch the value of a1* and a0*.
If S=CompareR in input,{
if a1*=a0* in input, then output Keep the value of a1* and a0*;
if a1*>a0* in input, then output Keep the value of a1* and a0*;
if a1*<a0* in input, then switch the value of a1* and a0*.}
If S=Keep in input, then Keep the value of a1* and a0*
If S=Switch in input, then Switch the value of a1* and a0*

Figure 42 Tile types for Small Compare Box

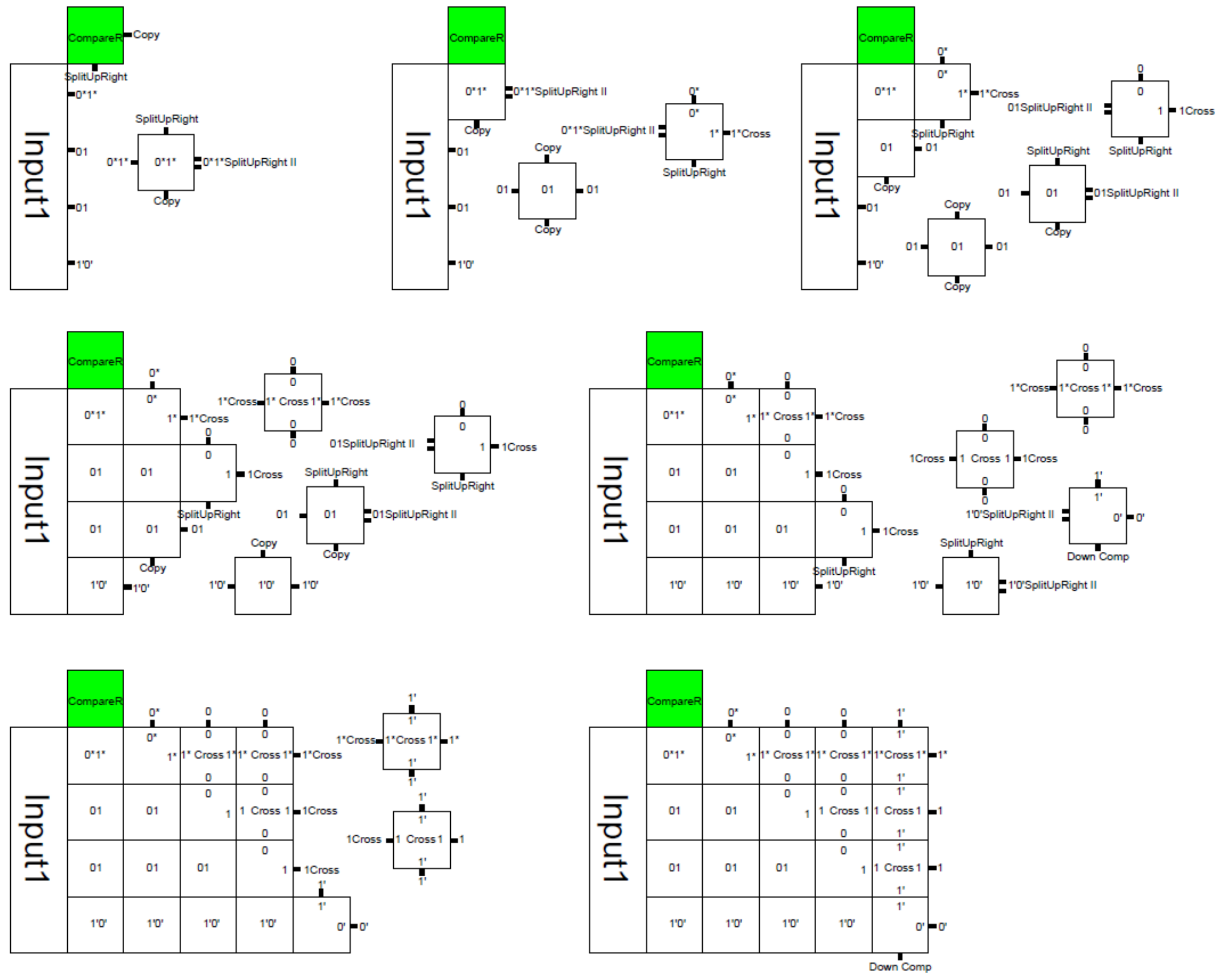


Figure 43 Design and example of Split Box.

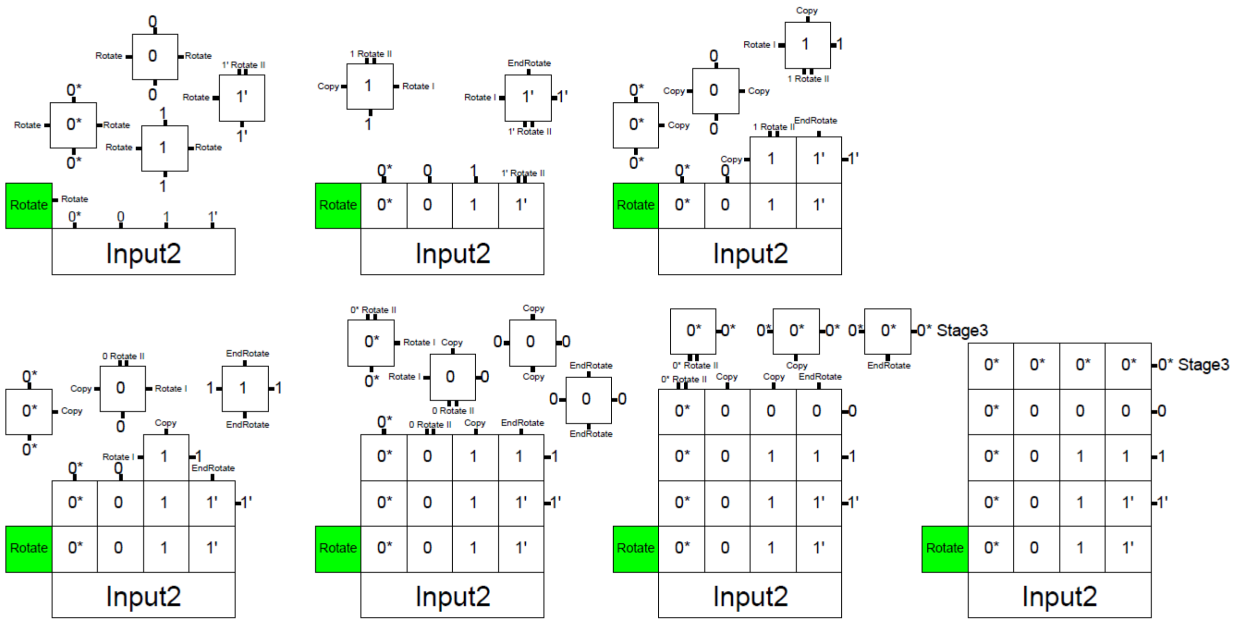
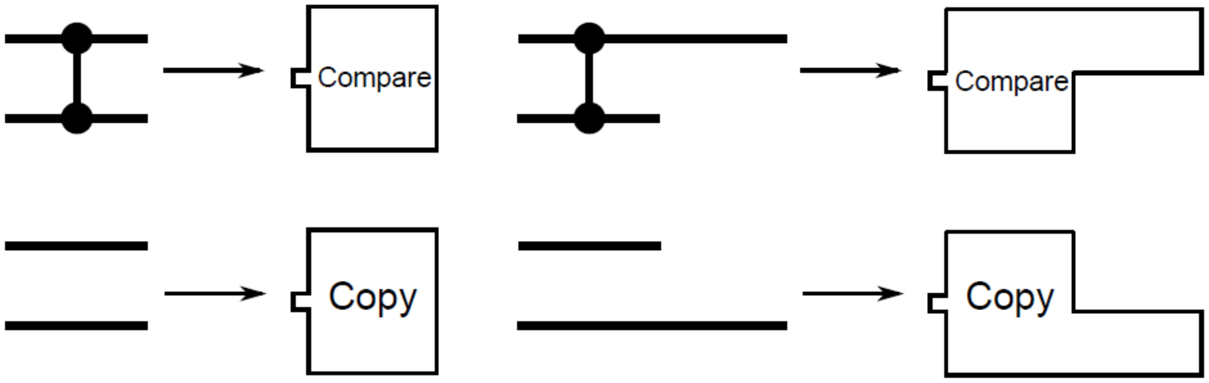


Figure 44 Design and example of Rotate Box.



Examples of transforming a sorting network to the black box model

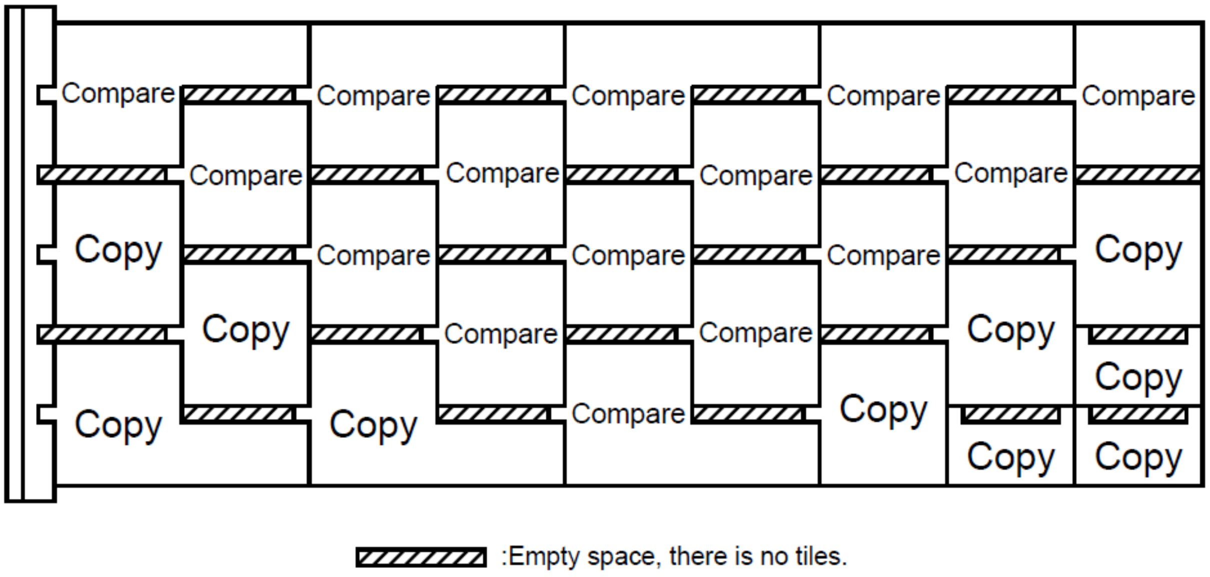


Figure 45 The seed is the one tile width tiles tape on the left most of this box. Then the structurer next to the seed is the preformatting tiles, and then the boxes preform the bubble sorting. The output is the glues on the right of this box.

BIOGRAPHICAL SKETCH

In 2013, Xingsi Zhong received his Master Of Science Degree from University of Texas Pan-American, majored in Computer Science. In 2010, he received his Bachelor Of Science Degree from Jilin University, majored in Math. His mailing address is 1607 W Schunior St Apt 515, Edinburg, Tx, 78541.