Modular Algorithm in Tile Self-assembly Model

Xiwen Fang and Xuejia Lai*

Department of Computer Science and Engineering, Shanghai Jiao Tong University,
Shanghai 200240, China
lai-xj@cs.sjtu.edu.cn

Abstract. In this paper we propose a system computing $A \bmod B$ for given n_A-bit binary integer A and n_B-bit binary integer B, which is the first system directly solving the modulus problem in tile assembly model. The worst-case assembly time of our system is $\Theta(n_A(n_A - n_B))$ and the best-case assembly time is $\Theta(n_A)$.

Although the pre-existing division system which computes A/B can also be used to compute $A \bmod B$, the assembly time of this system is not ideal in some cases. Compared with the pre-existing division system, we achieved improved time complexity in our system. Our advantage is more significant if n_A is much greater than n_B.

Keywords: tile assembly model, DNA computing, modulus problem, assembly time.

1 Introduction

1.1 Background and Related Work

The tile assembly model theory [1,2] provides a useful framework to study the self-assembly of DNA. Researchers have demonstrated DNA implementations of several tile systems: Barish et al. [3] have demonstrated DNA implementations of copying and counting; Rothemund et al. [4] have demonstrated DNA implementation of xor tile system. Several systems solving satisfiability problem are also proposed [5,6].

The efficiency of a tile assembly system involves two factors: the tileset-size and the assembly time. In [7], a division system which computes A/B for given n_A-bit binary integer A and n_B-bit binary integer B was proposed. The assembly time is always $\Theta(n_A(n_A - n_B))$. The system can also be used to compute $A \bmod B$ by computing the remainder of A/B. However, the time complexity of this system is not ideal in some cases:

Assume we have $A = 110000001_2$, $B = 11_2$, we can get the result of $A \bmod B$ by computing $A - B * 10000000_2$. Such expression can be computed using tile assembly system which requires linear time. Our goal in this paper is to construct a system which directly solves the modulus problem, thus the assembly time can be improved.

* Corresponding author.

The remaining of this paper is organized as follow: in section 1.2 we briefly introduce the concept of tile assembly model to assist the reader. In section 1.3 we introduce the corresponding algorithms. Several subsystems are discussed in section 2. In section 3 we present a system solving modulus problem and compare our system with existing system [7] in terms of time complexity. Our contributions are summarized in section 4.

1.2 Tile-Assembly Model

To assist the reader, in this section we briefly introduce the concept of tile assembly model. We refer to Σ as a finite alphabet of symbols called binding domains. We assume $null \in \Sigma$. Each tile has a binding domain on its north, east, south and west side. We represent the set of directions as $D = \{N, E, S, W\}$.

A tile over a set of binding domains is a 4-tuple. For a tile t, for $\langle\delta_N$, δ_E, δ_S, $\delta_W\rangle \in \Sigma^4$, we will refer to $bd_d(t)$ as the binding domain of tile t on d 's side.

A **strength function** $g : \Sigma \times \Sigma \to N$ denotes the strength of the binding domains. g is commutative and $\forall \delta \in \Sigma, g(\delta, null) = 0$. Let T be a set of tiles containing *empty*. A tile system S is a triple $< T, g, \epsilon >$. $\epsilon \in \mathbb{N}$ is the temperature. A tile can attach to a configuration only in empty positions if and only if the total strength of the appropriate binding domains on the tiles in neighbouring positions meets or exceeds the temperature ϵ.

Given a set of tiles Γ, a **seed configuration** $S' : \mathbb{Z}^2 \to \Gamma$ and $S =< T, g, \epsilon >$, configurations could be produced by S on S'. If no more attachment is possible, then we have the **final configuration**.

The reader may refer to [1,2,8] for more discussion of the concept of tile assembly model.

1.3 Preliminary Algorithms

To illustrate the basic idea of the modular tile system, we introduce Algorithm 1 which computes A mod B without proof. According to the algorithm, a subsystem which is able to compare two numbers and a subsystem which performs subtraction according to the compare result are required.

Given n_A-bit binary integer A and n_B-bit binary integer B. We denote by A_i and B_i the i_{th} digit of A and B. $A = \sum_{i=0}^{n_A-1} 2^i A_i$, $B = \sum_{i=0}^{n_B-1} 2^i B_i$.

Let $A \geq B$. In order to analysis the assembly time of our modular system in later sections, we define the best-case of Algorithm 1 as follow: for the first time we compute $A - B * 2^{n_A - n_B}$, we get the difference and the difference is less than B, i.e. for the best-case the loop body will be executed only once. A typical example is $A = B * 2^m + C$, $C < B$. In this case we have $A - B * 2^{n_A - n_B} < B$, thus the loop body will be executed only once.

We define the worst-case as follow: each time we perform the subtraction, the length of A is decreased only by 1. Let $Q = A/B$, the loop body will be executed repeatedly n_Q times. A typical example is $A = 2^{n_A} - 1, B = 2^{n_B - 1}$. The loop body will be executed repeatedly $n_A - n_B + 1$ times.

```
   **Input**: $A$ , $B$
   **Output**: $S = A \bmod B$
1  **while** $A \geq B$ **do**
2      **if** $A \geq B * 2^{n_A - n_B}$ **then**
3          $A = A - B * 2^{n_A - n_B}$.
4      **end**
5      **else**
6          $A = A - B * 2^{n_A - n_B - 1}$
7      **end**
8  **end**
9  $S = A$
10 return $S$
```

Algorithm 1. Modular arithmetic

2 Subsystems of Modular System

2.1 Connector System

Fig. 1 shows the tile set of the connector system that will encode the modulus B and connect B to the configuration encoding A. This system will also be used to connect the extended-subtractor system and the shift-comparer system, which will be discussed in later sections. The tile set consists of two parts: the 9-computational-tile set(Fig. 1(a)) and the $3(n_B - 1) + 1$-inputting-tile set(Fig. 1(b)). The inputting tile set includes $< (1, 1), con_{n_B} - 1, 1, pre >$ which encodes the most significant bit(MSB) of B and connects it to the MSB of A. For the remaining bits of B, according to the inputs on the west side and south side, the encoding rule is shown in Fig. 1(b).

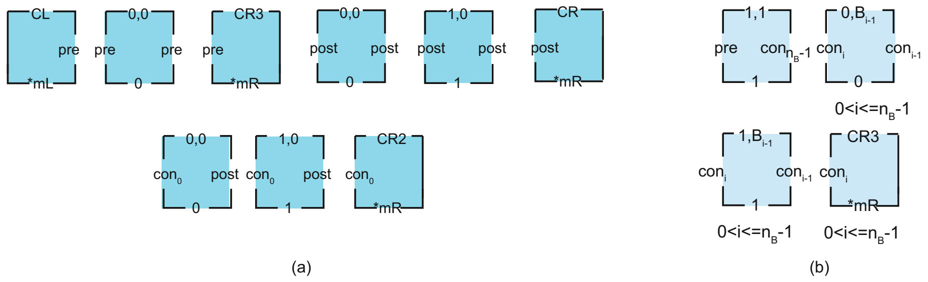

Fig. 1. The tiles have two input sides(south and west) and two output sides (north and east). (a) the computational tile set (b) the inputting tile set.

Let $\Sigma_{con} = \{||, pre, post, *mL, *mR, CL, CR, CR2, CR3,\ 0, 1,\ (0,0),\ (0,1),\ (1,0),\ (1,1)\} \cup \{con_i | 0 \leq i \leq n_B - 1\}$, the strength function g_{con} is defined as follow:

$$\begin{cases} \forall \delta \in \Sigma_{con}, g_{con}(\sigma, null) = 0 \\ g_{con}(CL, CL) = 2; g_{con}(*mL, *mL) = 2; g_{con}(||, ||) = 2 \\ \forall \sigma \in \Sigma_{con} \backslash \{null, *mL, CL\}, g_{con}(\sigma, \sigma) = 1 \\ \forall \sigma, \sigma' \in \Sigma_{con} \backslash \{null\}, \text{if} \sigma \neq \sigma', \text{then} g_{con}(\sigma, \sigma') = 0 \end{cases} \quad (1)$$

Let $A_i, B_i \in \{0, 1\}$, $n \geq n_A$. Let $\Gamma = \{\alpha_0 = \langle 0, ||, null, || \rangle$, $\alpha_1 = \langle 1, ||, null, || \rangle$, $\gamma_h = \langle *mL, ||, null, null \rangle$, $\gamma_t = \langle *mR, null, null, || \rangle\}$. Let the seed configuration $S : \mathbb{Z}^2 \to \Gamma$ be such that

$$\begin{cases} S(1, 0) = \gamma_t; S(-n, 0) = \gamma_h \\ \forall i = 0, 1, \ldots, n-1 : S(-i, 0) = \alpha_{A_i} \\ \text{For all other } (x, y) \in \mathbb{Z}^2, S(x, y) = empty. \end{cases} \quad (2)$$

Lemma 1. *Let $\epsilon_{con} = 2$, and T_{con} be a set of tiles over Σ_{con}(Fig. 1). Given a seed configuration encoding A which is defined in*(2)*, $S_{con} =< T_{con}, g_{con}, \epsilon_{con} >$ connects B with A:*

- *If $n_A > n_B$, pads B with $n_A - n_B$ 0-bits in the least significant bit place and $n - n_A$ 0-bits in the most significant bit place, outputs the identifiers CL and CR;*
- *If $n_A = n_B$, pads B with $n - n_B$ 0-bits in the most significant bit place, outputs the identifiers CL and $CR2$;*
- *If $n_A < n_B$, pads B with $n - n_A$ 0-bits in the most significant bit place, outputs the identifiers CL and $CR3$*

Let $A = 1101010_2$,$B = 1011_2$, $n = 8$. Fig. 2(a) shows the tile system encoding $B = 1011_2$. Fig. 2(b) shows the seed configuration which encodes A and pads it with one 0-tile. S_{con} connects the modulus B with A. Fig. 2(c) shows the example execution of S_{con} on the seed configuration. In this case, we have $n_A > n_B$, thus the final configuration outputs (01101010_2,$0\mathbf{1011}000_2$) and the identifiers CL and CR.

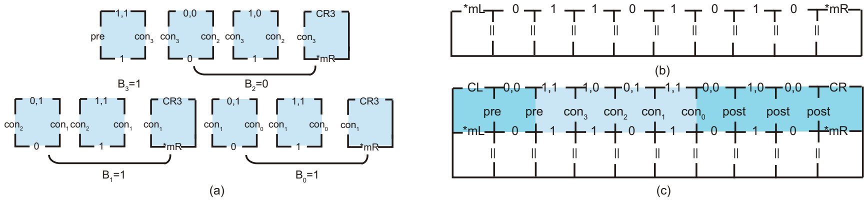

Fig. 2. (a) Inputting tile set encoding $B = 1011_2$ (b) seed configuration encoding $A = 1101010_2$ (c) S_{con} connects B with A

2.2 Shift-Comparer System

In this section we present a system with two jobs: to compare A with B and to perform right-shift on B. Fig. 3(a) shows the concepts behind the tiles in T_C, which have two input sides (west and south) and two output sides(north and east). The concepts includes variables a, b, c, r, r'. We have $a, b, c \in \{0, 1\}$ and $r, r' \in \{>, =, <\}$. The input sides includes variables a, b, c, r, thus there are 24 tiles(Fig. 3 (c)). The rule of comparing A with B is shown in Fig. 3 (b). We compare A_i with $B_i (i = n-1, n-2, ..., 0)$. The assumption $A = B$ holds until we find $A_i \neq B_i$: if $A_i \geq B_i$, then we have $A \geq B$; else if $A_i < B_i$, we have $A < B$(Fig. 3(b)). Fig. 3(d) shows the corresponding boundary tiles.

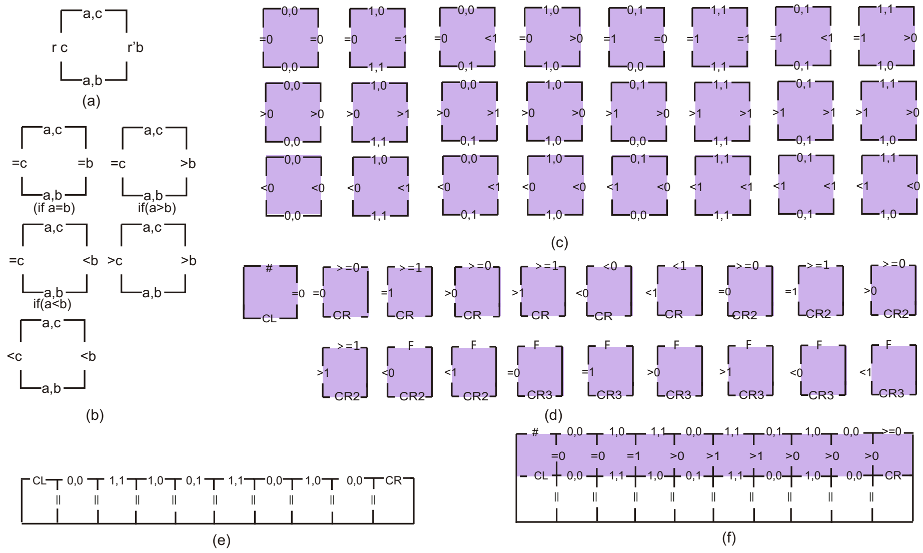

Fig. 3. (a) concepts behind the comparer system (b) rules of comparing A and B (c) computational tiles (d) boundary tiles (e) $A = 1101010_2$ and $B = 1011000_2$. (f)$A \geq B$, $B_0 = 0$. F outputs (1101010_2,101100_2) and the boundary flags (#, >= 0).

Let $\Sigma_C = \{||, (0,0), (0,1), (1,0), (1,1), =0, =1, <0, <1, >0, >1, >=0, >=1, CL, CR, CR2, CR3, F, \#\}$. Let $n \geq \max(n_A, n_B)$. Let $\Gamma = \{\alpha_0\beta_0 = \langle(0,0), ||, null, ||\rangle, \alpha_0\beta_1 = \langle(0,1), ||, null, ||\rangle, \alpha_1\beta_0 = \langle(1,0), ||, null, ||\rangle, \alpha_1\beta_1 = \langle(1,1), ||, null, ||\rangle, \gamma_h = \langle CL, ||, null, null\rangle, \gamma_{t1} = \langle CR, null, null, ||\rangle\}, \gamma_{t2} = \langle CR2, null, null, ||\rangle, \gamma_{t3} = \langle CR3, null, null, ||\rangle\}$. The strength function g_C is defined as follow:

$$\begin{cases} \forall \sigma \in \Sigma_C, g_C(\sigma, null) = 0 \\ g_C(CL, CL) = 2; g_C(>=0, >=0) = 2 \\ g_C(>=1, >=1) = 2; g_C(<,<) = 2; g_C(||,||) = 2 \\ \forall \sigma \in \Sigma_C' \backslash \{null, CL, >=0, >=1, <, ||\}, g_C'(\sigma, \sigma) = 1 \\ \forall \sigma, \sigma' \in \Sigma_C' \backslash \{null\}, \sigma \neq \sigma', g_C(\sigma, \sigma') = 0 \end{cases} \tag{3}$$

Let $n \geq \max(n_A, n_B)$. Let the seed configuration $S : \mathbb{Z}^2 \to \Gamma$ be such that

$$\begin{cases} S(1,0) = \gamma_t, \gamma_t \in \{\gamma_{t1}, \gamma_{t2}, \gamma_{t3}\}; S(-n,0) = \gamma_h \\ \forall i = 0, 1, \ldots, n-1 : S(-i, 0) = \alpha_{A_i} \beta_{B_i} \\ \text{For all other } (x,y) \in \mathbb{Z}^2, S(x,y) = empty. \end{cases} \tag{4}$$

Let T_C be a set of tiles over Σ_C as described in Fig. 3(c,d). We have $|T_C| = 43$.

Lemma 2. *Let $\epsilon_C = 2$, g_C defined as above. Given a seed configuration encoding A and B which is defined in* (4)*, the system $S_C =< T_C, g_C, \epsilon_C >$ produces the final configuration F:*
I) $bd_N(F(1,-n))$ =‘#’. S_C' performs a right-shift on B: $\forall 0 \leq i \leq n-1$, $bd_N(F(1,-i)) = (A_i, B_{i+1})$.
II) For seed configurations with different boundary flags, S_C' outputs different information in the position (1,1):

- *$S(1,0) = \gamma_{t1}$: If $A \geq B$, $bd_N(F(1,1))$ =‘$>= B_0$’; else $bd_N(F(1,1))$ =‘$<$’.*
- *$S(1,0) = \gamma_{t2}$: If $A \geq B$, $bd_N(F(1,1))$ =‘$>= B_0$’; else $bd_N(F(1,1))$ =‘F’.*
- *$S(1,0) = \gamma_{t3}$: $bd_N(F(1,1))$ =‘F’.*

Fig. 3(e) shows a sample seed configuration S that encodes $A = 1101010_2$ and $B = 1011000_2$. Fig. 3 (f) shows an example execution of S_c on the seed configuration and the top row reads the right-shift solution of B, i.e. 101100_2; the rightmost boundary flag ‘>= 0’ indicates that $A \geq B$ and the initial LSB of B is 0.

2.3 Extended Subtractor System

In this section we propose a extended subsystem that performs subtraction and shift-subtraction. The concepts behind the system are showed in Fig. 4(a). The concepts include variables a, b, c, b', c', and s. each of which can take on as values the elements of the set $\{0, 1\}$. The tiles have two input sides (east, south) and two output sides (west, north). For the subset that performs subtraction, on the input sides there are 3 variables, thus there are 8 tile types in this system; for the subset that performs shifter-subtraction, on the input sides there are 4 variables, thus there are 16 actual tile types in this system. Therefore in this system we have totally 24 tile types(Fig. 4(b)). Fig. 4(c) shows 5 extra boundary tiles.

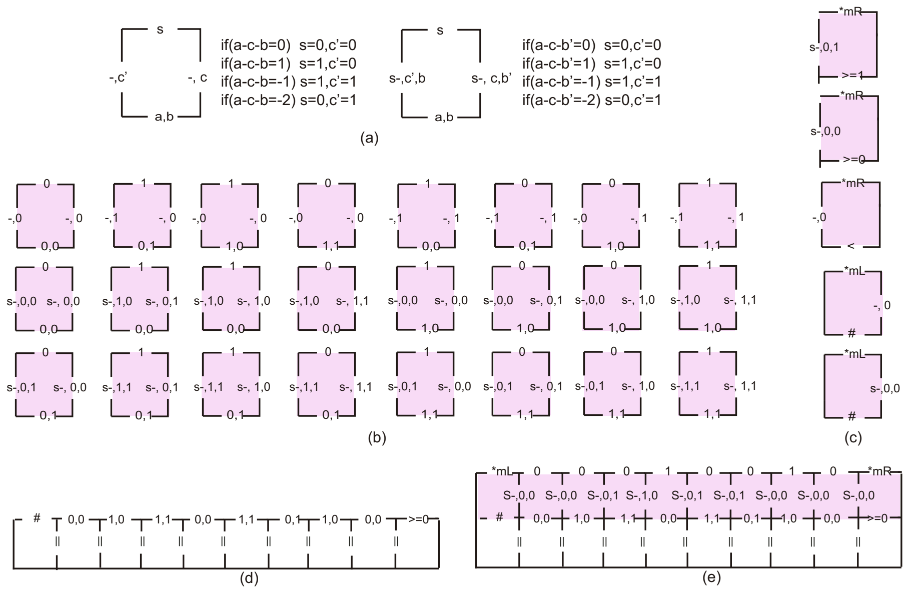

Fig. 4. (a) the concept behind the system computing $f(A, B) = A - B$ and $f(A, B) = A - 2B - t$ (b) the 24-computational-tile types (c) the 5-boundary-tile types (d)$A = 1101010_2$ and $B = 101100_2$ (e)$A - 2 * B = 10010_2$

Let $\Sigma_- = \{||, (0,0), (0,1), (1,0), (1,1), (-,0), (-,1),$ 0,1, $(s-,0,0)$, $(s-,0,1)$, $(s-,1,0)$, $(s-,1,1)$, #, $>= 0$, $>= 1$, $<$,$*mL$, $*mR\}$, the strength function g_- is defined as follow:

$$\begin{cases} \forall \sigma \in \Sigma_-, g_-(\sigma, null) = 0 \\ g_-(*mL, *mL) = 2; g_-(>= 0, >= 0) = 2 \\ g_-(>= 1, >= 1) = 2; g_-(<, <) = 2; g_-(||, ||) = 2 \\ \forall \sigma \in \Sigma_- \backslash \{null, *ml, >= 0, >= 1, <, ||\}, g_-(\sigma, \sigma) = 1 \\ \forall \sigma, \sigma' \in \Sigma_- \backslash \{null\}, \text{if} \sigma \neq \sigma', \text{then} g_-(\sigma, \sigma') = 0 \end{cases} \tag{5}$$

Let $n \geq \max(n_A, n_B)$, $A \geq B$. Let $\Gamma = \{\alpha_0\beta_0 = \langle(0,0), ||, null, ||\rangle$, $\alpha_0\beta_1 = \langle(0,1), ||, null, ||\rangle$, $\alpha_1\beta_0 = \langle(1,0), ||, null, ||\rangle$, $\alpha_1\beta_1 = \langle(1,1), ||, null, ||\rangle$, $\gamma_h = \langle \#, ||, null, null\rangle$, $\gamma_{t1} = \langle >= 0, null, null, ||\rangle$, $\gamma_{t2} = \langle >= 1, null, null, ||\rangle$, $\gamma_{t3} = \langle <, null, null, ||\rangle\}$. Let the seed configuration $S : \mathbb{Z}^2 \to \Gamma$ be such that

$$\begin{cases} S(1,0) = \gamma_t, \gamma_t \in \{\gamma_{t1}, \gamma_{t2}, \gamma_{t3}\}; S(-n, 0) = \gamma_h \\ \forall i = 0, 1, .., n-1 : S(-i, 0) = \alpha_{A_i}\beta_{B_i} \\ \text{For all other } (x,y) \in \mathbb{Z}^2, S(x,y) = empty. \end{cases} \tag{6}$$

Lemma 3. *Let $\epsilon_- = 2$, let Σ_-, g_- defined as above, and T_- be a set of tiles over Σ_- (see Fig. 4(b,c)). Given a seed configuration encoding A and B which is defined in* (6)*, the system $S_- =< T_-, g_-, \epsilon_- >$ computes different functions:*

- $S(1,0) = \gamma_{t1}$ *: S_- computes the function* $f(A, B) = A - 2B$
- $S(1,0) = \gamma_{t2}$ *: S_- computes the function* $f(A, B) = A - 2B - 1$
- $S(1,0) = \gamma_{t3}$ *: S_- computes the function* $f(A, B) = A - B$

Fig. 4(d) shows a sample seed configuration encoding $A = 1101010_2$ and $B = 101100_2$. There is an identifier '$>= 0'$ which indicates $A-2B$ will be calculated. Fig. 4(e) shows an example execution: the top row displays the solution $A-2B = 10010_2$.

3 Modular System

3.1 Definition and Sample Execution

Let the seed configuration be defined as it is in (2). Let $\Sigma_M = \Sigma_{con} \cup \Sigma_C \cup \Sigma_-$. Let the strength function g_M be such that g_M agrees with g_{con}, g_C, g_- on their respective domains.

Theorem 4. *Let $\epsilon_M = 2$. Let Σ_M and g_M defined as above. Let T_{con} be the connector system defined in section 2.1 which contains an inputting set encoding B. Let $T_M = T_{con} \cup T_C \cup T_-$. Given a seed configuration encoding A which is defined in* (2)*, the system $S_M =< T_M, g_M, \epsilon_M >$ computes the function $f(A, B) = A \bmod B$.*

Due to limitations of space, we are not able to give the proof in detail. A brief explanation of Theorem 4 is given as follow:

According to Lemma 1, Lemma 2 and Lemma 3, S_C could be executed on the final configuration of S_{con}, S_{con} could be executed on the seed configuration of S_M and the final configuration of S_-, and S_- could be executed on the final configuration of S_C. Thus at the very beginning, S_{con} will be executed. Then S_C will be executed according to the output of S_{con}, which is (A, $B * 2^{n_A - n_B}$).

According to Algorithm 1, the intuition behind the modular system computing $A \bmod B$ is that comparing A with B:

- If $A < B$, then A itself is the solution;
- Else if $A \geq B * 2^{n_A - n_B}$, S_C outputs the binding domain '$>= B_0$' which indicates the comparison result; later S_- will calculate $A = A - B * 2^{n_A - n_B}$;
- Else if $A < B * 2^{n_A - n_B}$, S_C outputs the binding domain '$<$', and S_- will calculate $A = A - B * 2^{n_A - n_B - 1}$.

Then for A with the updated value, $A \bmod B$ will be computed again.The process will be repeated until we have $A < B$. The system outputs A as the final solution.

Fig. 5 shows an example execution of S_M with $A = 1101010_2$ and $B = 1011_2$. The seed configuration encoding $A = 1101010_2$ is shown in Fig. 5(a). The modulus $B = 1011_2$, $n_B = 4$, thus $B_3 = 1$, $B_2 = 0$, $B_1 = 1$ and $B_0 = 1$. Fig. 5(b) shows the corresponding inputting tile set encoding $B = 1011_2$. Fig. 5(c) shows the final configuration F_M. On the north side of row 8, S_C outputs $(111_2, 10_2)$ and the final identifier 'F', which indicates 111_2 is the solution, i.e. $1101010_2 \bmod 1011_2 = 111_2$.

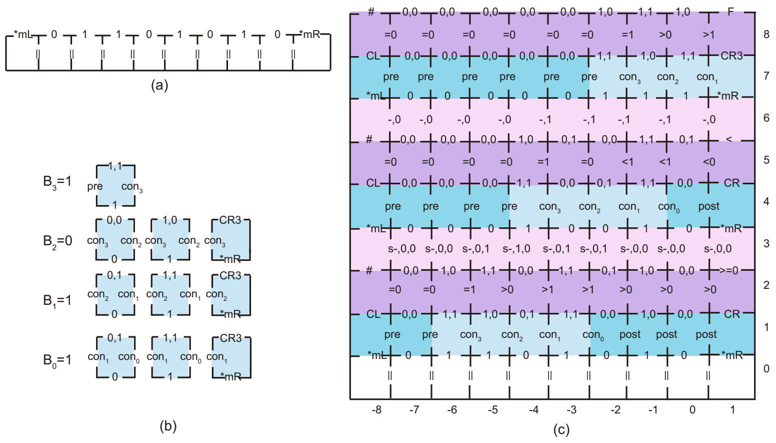

Fig. 5. (a) a sample seed configuration S that encodes $A = 1101010_2$, $n_A = 7$, $n = 8$. (b) the inputting tile set encoding the modulus $B = 1011_2$ (c) the modular system performs the computation 1101010_2 mod 1011_2 and produces a final configuration F_M on S. Along the top row F_M encodes the solution, i.e. 00000111_2.

3.2 Discussion

Lemma 5. *(Modular Assembly Time Lemma). Given the seed configuration defined in* (2) *which is of length n. For all A, B, if n and n_A have the same order of magnitude, the worst-case assembly time of S_M computing $A \bmod B$ is $\Theta(n_A(n_A - n_B))$ and the best-case assembly time is $\Theta(n_A)$.*

Although the division system proposed in [7] which computes A/B can also be used to compute $A \bmod B$, our scheme enjoys advantage in aspect of assembly time: the overall assembly time of our scheme is less than that of scheme[7]. In [7], to compute the remainder, the assembly time is always identical to that of the worst-case in our scheme. Assume we have $A = 110000001_2$, $B = 11_2$, $B' = 110000000_2$, i.e. B padded with seven 0-bits. To compute the remainder of A/B, in [7] it costs seven cycles to reduce the length of B' until it is identical to n_B; while in our scheme, such operation is not required. The fact that $A - B'$ is exactly the final solution of $A \bmod B$ could be verified directly after $A - B'$ is computed. Thus to compute $A \bmod B$, $A > B$, in our scheme, the best-case assembly time is $\Theta(n_A)$ while in [7] the assembly time is always $\Theta(n_A * (n_A - n_B))$, which is identical to our worst-case assembly time. Our advantage is more significant if n_A is much greater than n_B.

Lemma 6. *(Modular Tile-set Size Lemma). To compute $A \bmod B$, T_M requires 81 computational tile types and $3n_B - 2$ inputting tile types.*

4 Conclusion

In this paper we present a modular system which computes $A \bmod B$ for given A and B. The system requires 81 computational tiles and $3n_B - 2$ inputting tile types. Compared with pre-existing division system computing A/B whose assembly time is always $\Theta(n_A(n_A - n_B))$, the worst-case assembly time of our system is $\Theta(n_A(n_A - n_B))$ and the best-case assembly time is $\Theta(n_A)$, which is an improvement on time complexity.

As the tile assembly model is a highly distributed parallelism model of computation, taking advantage of parallelism, more complicated system involving modulus problem could be constructed. We leave this as future work.

Acknowledgements. This work was supported by the National Natural Science Foundation of China (No. 61272440, No. 61073149), Research Fund for the Doctoral Program of Higher Education of China (20090073110027), State Key Laboratory of ASIC & System (11KF0020), Key Lab of Information Network Security, Ministry of Public Security (C11603) and Scholarship Award for Excellent Doctoral Student granted by Ministry of Education.

References

1. Rothemund, P., Winfree, E.: The program-size complexity of self-assembled squares. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, pp. 459–468. ACM (2000)
2. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Institute of Technology (1998)
3. Barish, R., Rothemund, P., Winfree, E.: Two computational primitives for algorithmic self-assembly: Copying and counting. Nano Letters 5(12), 2586–2592 (2005)
4. Rothemund, P., Papadakis, N., Winfree, E.: Algorithmic self-assembly of dna sierpinski triangles. PLoS Biology 2(12), e424 (2004)
5. Brun, Y.: Solving np-complete problems in the tile assembly model. Theoretical Computer Science 395(1), 31–46 (2008)
6. Brun, Y.: Improving efficiency of 3-SAT-solving tile systems. In: Sakakibara, Y., Mi, Y. (eds.) DNA 16 2010. LNCS, vol. 6518, pp. 1–12. Springer, Heidelberg (2011)
7. Zhang, X., Wang, Y., Chen, Z., Xu, J., Cui, G.: Arithmetic computation using self-assembly of dna tiles: subtraction and division. Progress in Natural Science 19(3), 377–388 (2009)
8. Winfree, E., Liu, F., Wenzler, L., Seeman, N., et al.: Design and self-assembly of two-dimensional dna crystals. Nature 394(6693), 539–544 (1998)