A Design of Time-Optimum and Register-Number-Minimum
Systolic Convolver

Hiroshi Umeo

Osaka Electro-Communication University
Neyagawashi, Hatsudo, 18-8, Osaka, 572, Japan

Abstract

We present an optimum bit-parallel/word-sequential systolic convolver. Our design is the best one among the previous many convolvers in the sense that its optimality in time and space performances is simultaneously attained without augmenting any global control, broadcasting, preloading, and/or multi sequential or parallel I/O ports, which were allowed in most of the previous designs. As an application of our convolver we give a systolic polynomial divider which can compute the polynomial division in exactly \( n + O(1) \) steps on \( \lfloor \min(n-m, m)/2 \rfloor + O(1) \) systolic cells, for the division of any degree \( n \) polynomial by any degree \( m \) polynomial \( (n \geq m) \).

1. Introduction

Convolution is probably one of the most important problems in the fields of signal and image processing. A large number of convolution algorithms have been proposed, since this type of computation is commonly used for many problems, such as pattern matching, digital filtering, discrete Fourier transforms, polynomial multiplication and division, and so on. Recently much attention has been paid to the study of systolic convolution algorithms\[4-5, 7, 9-12, 14, 18-19, 21-24, 27-29, 33-36\] and several practical constructions and implementations on VLSI are made\[12, 18, 27, 35\].

In this paper we will present a time-optimum and register-number-minimum systolic convolver. Our algorithm is based on Atrebuin's binary parallel multiplier\[2\], Knuth's revised version\[20\] and Cole's real-time iterative palindrome recognizer\[8\], which have been known as tricky cellular algorithms.

The systolic array that we assume is the most basic model which consists of a locally-connected semi-infinite array of identical systolic cells with a single I/O port being positioned at one end of the array. The data broadcasting, preloading, and/or multi sequential or parallel I/O ports are not allowed in our model.

\[ (*) \] The author stays as an Alexander-von-Humboldt researcher at Institut für Theoret. Informatik, Techn. Universität Braunschweig, Gaussstrasse 11, D-3300, Braunschweig, West Germany.

It is shown that there exists a systolic array \( M \) which can compute the convolution \( c_i (i = 0, 1, 2, \ldots, n + m - 1) \) of any two finite sequences \( a_j (j = 0, 1, 2, \ldots, n - 1) \) and \( b_k (k = 0, 1, 2, \ldots, m) \) in optimum real-time, exactly in \( n + m + O(1) \) steps, using \( \lceil \min(m,n)/2 \rceil + O(1) \) systolic cells. The number of data registers in each systolic cell is minimum. A new data routing scheme developed for our convolver helps us to understand the correctness of the tricky algorithms above. As an application of our convolver, we give a systolic real-time pattern matcher and show that not only data preloading but also broadcasting are not necessarily essential operations in the design of systolic optimum-time pattern matcher. In addition we give a fastest systolic polynomial divider which can compute the polynomial division in exactly \( n + O(1) \) steps on \( \lfloor \min(n-m, m)/2 \rfloor + O(1) \) systolic cells, for the division of any degree \( n \) polynomial by any degree \( m \) polynomial \( (n \geq m) \). The divider is time-optimum and register-number-minimum. Our design is superior to the previous ones\[21, 33, 36\] in both the time and space(register-number) complexities.

The organization of this paper is as follows:
In section 2 an optimum real-time and register-number-minimum systolic convolver is presented. Its validity is also given. In section 3 several variations and applications of our convolver are presented. Lastly we give a conclusion in section 4.

2. Time-optimum and register-number-minimum systolic convolver

2.1 Basic definitions

We first consider the convolution of two infinite sequences defined as follows:
Convolution: Given two infinite sequences \( A \) and \( B \) such that
\[ A = < a_0, a_1, \ldots, a_i, \ldots > \ldots (1) \]
\[ B = < b_0, b_1, \ldots, b_i, \ldots > \ldots (2) \]
Compute an infinite sequence \( C \),
\[ C = < c_0, c_1, \ldots, c_i, \ldots > \ldots (3) \]
defined by \( c_i = \sum_{k=0}^{i} a_k b_{i-k} \), \( i = 0, 1, 2, \ldots \ldots (4) \)
A systolic convolver $M$ that we design consists of a buffer and a semi-infinite array of identical processors, called systolic cells, $C_i$, $i = 1, 2, \ldots$, that operate synchronously at discrete time steps driven by a common clock. See Fig. 1. The input $(a_i, b_i)$, $i = 0, 1, 2, \ldots$, is fed serially to the leftmost cell via the buffer, and from which the serial output $c_j$, $j = 0, 1, 2, \ldots$, is also obtained. The buffer receives a pair of $(a_i, b_i)$, $i = 0, 1, 2, \ldots$, at time $t = i$ from the host computer at the rate of one pair / one step. The buffer also outputs $c_i$ to the host computer at time $t = i + k$ at the rate of one symbol / one step, where $k$ is some fixed integer, for any nonnegative integer $j$. We establish the following main theorem. The design of the convolver and its correctness will be given below (Propositions 1 to 4).

![Fig. 1 An illustration of real-time systolic convolver.](image)

[Theorem 1] The systolic convolver $M$ presented below can compute the convolution $c_i(i = 0, 1, 2, \ldots)$ of two infinite sequences $a_i$, $b_i$ ($i = 0, 1, 2, \ldots$) in real-time, that is, for any $i$, $i = 0, 1, 2, \ldots$, $M$ outputs $c_i$ at time $t = i + k$, where $k$ is some fixed integer. The number of data registers in each systolic cell is minimum. Further, $M$ outputs the convolution $c_i(i = 0, 1, 2, \ldots, 2n - 1)$ of $a_i$, $b_i$ ($i = 0, 1, 2, \ldots, n - 1$) in exactly $2n + O(1)$ steps, using $[n/2] + O(1)$ systolic cells.

2.2 Design

First we design a systolic convolver with each cell having four pairs of data registers, since it helps us to understand the correctness of our convolver. Afterward, we will show that three-pair is the necessary and sufficient data-register number for the real-time convolution computation.

**Basic cell definition:**

Each systolic cell has four pairs of data registers $R_i$ $(1 \leq j \leq 4)$ and $R$. Each $R_i$ consists of two subregisters $A_i$ and $B_i$, storing $(a_i, b_i)$ for some $k$. The buffer has three registers $A$, $B$, and $C$ storing $(a_i, b_i, c_i)$ similarly as $R_i$, $R$, and $R$ of the systolic cell. See Fig. 1. Each register is used for the following purposes:

- $R_1$, $R_2$: Data holding register.
- $R_3$: A pipeline register which transmits data to the right neighbour cell at unit speed.
- $R_4$: Stores data temporary.
- $R_5$: Stores the partial sum of convolution and transmits it to the left neighbour cell at unit speed.

In the description below we use the following symbolic conventions.

$R_i^t$: Denotes the content of $R_i$ in $C_i$ at step $t$. Similar notations are used for other registers.

$R = \emptyset$, $R \neq \emptyset$, $R = \phi$: Each denotes that the register $R$ is empty, is not empty, and is set empty, respectively.

- **Parallel Data Routing Scheme:**
  - **Initialization:** At time $t = 0$ we assume that:
    
    $R_3^t(buff) = (a_0, b_0)$,
    
    $R_3^t(buff) = \emptyset$,
    
    $c_1(t = 1): R_3^t(a_1, b_1) \neq \emptyset \quad (1 \leq t \leq 4)$

  - **At time $t + 1$:**
    
    $R_3^{t+1}(buff) = (a_2, b_2)$,
    
    $R_3^{t+1}(buff) = R_3^t(buff)$

    $c_1(t = 1 + 1):
    
    \begin{cases}
    \text{if } (R_3^t(a_1, b_1) \neq \emptyset) \text{ then } \{ c_i \text{ does nothing } \} \\
    \text{else if } (R_3^t(a_1, b_1) = \emptyset) \text{ then } (R_3^t(a_1) - R_3^{t-1}(a_1)) \quad (A) \\
    \text{else if } (R_3^t(a_1, b_1) \neq \emptyset) \text{ then } (R_3^t(a_1) - R_3^{t-1}(a_1)) \quad (B) \\
    \text{else if } (R_3^t(a_1, b_1) \neq \emptyset) \text{ then } (R_3^t(a_1) - R_3^{t-1}(a_1)) \quad (C) \\
    \end{cases}$

    $c_1(t = 1 + 1):
    
    \begin{cases}
    \text{if } (R_3^t(a_1, b_1) = \emptyset) \text{ then } (R_3^{t-1}(a_1) - R_3^{t-1}(a_1)) \quad (D) \\
    \end{cases}$

![Fig. 2 Data routing scheme for real-time systolic convolver.](image)
The operation of the systolic cell consists of data routing which involves \( R_j(i \leq j \leq 4) \) and convolution computation on \( R_3 \). Both of them are performed simultaneously, however, we will explain them separately for ease of description and understanding.

In the equation (7) we regard \( R_3^{-1}(i-1) \) as \( B \) in the case \( i = 1 \). \( C_3 \) does nothing at time \( t \) while \( R_3^{-1}(i-1) \) is empty. Let \( t \) be the time that \( R_3(i-1) \) has its first data. Then, within the next three steps, that is, \( t+1, t+2, \) and \( t+3, \) \( C_3 \) performs the operation (A), (B), and (C) in this order. After that \( C_3 \) follows (D) at every step. Each data-pair is transmitted through the pipeline register \( R_3 \) in the right direction at unit speed until it will find an empty \( R_3 \) and/or \( R_4 \). When they are found, \( R_3 \) has a priority for storing the data. The next pair will be loaded in \( R_3 \) in the same cell. Once a pair is stored in \( R_3 \) or \( R_4 \), it will stay within that cell forever. The content of \( R_4 \) is also copied by \( R_3 \) in the same cell at the next step and is stored temporarily in it. Fig. 2 shows our data routing scheme given above.

#### Computation rule for convolution:

- **Initialisation:** \( E_c^0(\text{buffer}) = E_c^0(i) = \phi, \text{for all } i \geq 1 \).

At time \( t \geq 1 \):
- **Buffer:** \( E_c^0(\text{buffer}) = E_c^0(i) \).
- **Cell \( C_i(i \geq 1):** Each cell \( C_i \) obeys the rule given below according to the number of pairs of data loaded in its \( E_c^0(i \geq 1 \leq 4) \) at time \( t = 1 \). We will refer them as rule 1, 2, 3, and 4, respectively.

#### Rules:

- **No-pair:** \( E_c^0(i) = A_c^{-1}(i) A_1^{-1}(i) \).
- **One-pair:** \( E_c^0(i) = A_c^{-1}(i) A_1^{-1}(i) A_2^{-1}(i) \).
- **Two-pair:** \( E_c^0(i) = A_c^{-1}(i) A_1^{-1}(i) A_2^{-1}(i) A_3^{-1}(i) \).
- **Three-pair:** \( E_c^0(i) = A_c^{-1}(i) A_1^{-1}(i) A_2^{-1}(i) A_3^{-1}(i) A_4^{-1}(i) \).
- **Four-pair:** \( E_c^0(i) = A_c^{-1}(i) A_1^{-1}(i) A_2^{-1}(i) A_3^{-1}(i) A_4^{-1}(i) \).

Fig. 3 shows our computation rule for convolution. Exactly the following equations hold for \( R_3 \) in the buffer and \( C_3(i \geq 1) \). Consult Fig. 4 for the help of understanding these equations. The mark \( \phi \) in \( E_c^0(i) \) denotes that its content is partial.

\[
\begin{align*}
R_4^3(\text{buffer}) &= \phi \quad (t < 3) \\
&= c_{t-3} \quad (t \geq 3) \\
R_4^1(i) &= \phi \quad (t < 2) \\
&= c_{t-2} \quad (t \geq 2) \\
R_4^2(i) &= \phi \quad (t \leq 1, i \geq 2) \\
&= d_{t+1} \quad (t \geq 1, i \geq 2)
\end{align*}
\]

#### 2.3 Validity

We will show informally the correctness of our convolver. Let \( c_{i} \) be the term such that:

\[
c_i = a_1 b_0 + a_1 b_{i-1} + a_2 b_{i-2} + \cdots + a_{k-1} b_{i-k+1} + a_k b_{i-k}.
\]

Defined in (4). The index \( i \) can be represented as \( i = 4k + \xi \), where \( k \) is any nonnegative integer and \( \xi \) is any integer in \( \{0, 1, 2, 3\} \). Then,

\[
c_i = c_{4k+\xi} = w_1 + w_2 + \cdots + w_k + w_{k+1} \quad \text{(14)}.
\]

Fig. 3 Computation rules for convolution.

where \( w_1, w_2, \ldots, w_{k+1} \) are defined as follows:

\[
w_j = a_{j-2j} b_{2j-2} + a_{j-1-2j} b_{2j-1} + a_{j-2-2j} b_{2j-2} + w_{2j-1} \quad \text{for} \quad 1 \leq j \leq k \text{ and} \]

\[
\begin{align*}
&\left\{ \begin{array}{ll}
a_{2k} b_{2k} + a_{2k+1} b_{2k+1} & \text{when } \xi = 0 \\
 a_{2k} b_{2k+2} + a_{2k+1} b_{2k+3} + a_{2k+2} b_{2k+4} + a_{2k+3} b_{2k+5} + a_{2k+4} b_{2k+6} & \text{when } \xi = 1 \\
 a_{2k} b_{2k+3} + a_{2k+1} b_{2k+4} + a_{2k+2} b_{2k+5} + a_{2k+3} b_{2k+6} + a_{2k+4} b_{2k+7} & \text{when } \xi = 2 \\
 a_{2k} b_{2k+4} + a_{2k+1} b_{2k+5} + a_{2k+2} b_{2k+6} + a_{2k+3} b_{2k+7} + a_{2k+4} b_{2k+8} + a_{2k+5} b_{2k+9} & \text{when } \xi = 3
\end{array} \right.
\end{align*}
\]

The computation of the value \( c_{4k+\xi} \) is made on \( M \) as follows: The systolic cell \( C_4 \) first computes \( w_{k+1} \) using rule 1, 2, 3, or 4 depending on whether \( \xi = 0, 1, 2, \text{ or } 3 \), respectively. At the next step this partial sum is transmitted to the left neighbour cell \( C_3 \).\( w_{k+1} \) is computed on it, using rule 4. And at the next step, applying rule 4, \( C_3 \) computes \( w_{k+2} \) using rule 1, 2, 3, or 4, and so on. And \( k \) steps later, \( w_{k+1} \) is obtained on \( C_1 \). On the next step this value is
of each systolic cell. From the definition of I/O rates the time optimality in [Theorem 1] is easily seen. Next we show that three pairs of data registers in each cell \( L \), excluding \( R_i \), are minimum for the real-time systolic computation of convolution.

[Proposition 2] The three pairs of data registers \( R_1 \), \( R_2 \), and \( R_3 \) in each cell are necessary and sufficient for any real-time systolic convolver.

(Proof) Let \( c_1 = a_{i,0} + a_{i-1, b_1} + \ldots + a_{1, b_1} \). We refer the set of cells \( C_1, C_2, \ldots, C_{i/4} \) of \( M \) as region 1. See Fig. 5. The following observations are obtained.

![Fig. 5](image)

Fig. 5 A region of systolic cells which involve the computation of \( c_1 = a_{i,0} + a_{i-1, b_1} + \ldots + a_{1, b_1} \).

1. It is the subterm \( a_{i/2} b_{i/2} \) that can be computed first by on \( M \) within \( i/2 + i/2 \) subterms which constitute \( c_1 \), since it is the first subterm that both of the input data are ready for the computation. Moreover the multiplication of \( a_{i/2} b_{i/2} \) must be made on some cell in the region 1, since the pair is followed at time \( t = i/2 \) and the result must be output at time \( t = i \) in real-time. From the same reason \( a_{i, b_1} \), \( i = 0, 1, 2, \ldots, i/2 - 1 \) must stay at somewhere in the region 1.

2. The \( i/4 \) pairs of data fed from time \( t = i/4 \) to \( i \) cannot go out of the region 1, even if they are ready for the computation. Therefore \( M \) must hold \( 31/4(= i/2 + i/4) \) pairs of input data on the region 1 consisting of \( i/4 \).
systolic cells. So three pairs of data registers are necessary for our computation.

By a slight observation we show that the register \( R_4 \) of each systolic cell can be removed from \( M \). The \( R_4 \) is used only for the rule 4. When we apply the rule 4, the following register \( R \) is used instead of \( R_4^{-1}(1) \).

\[
R = \begin{cases} 
R_4^{-1}(i+1), & \text{when } R_2^{-1}(i+1) = 0, \\
R_2^{-1}(i+1), & \text{when } R_2^{-1}(i+1) \neq 0 \text{ and } R_3^{-1}(i+1) = \phi, \\
R_3^{-1}(i+1), & \text{otherwise.}
\end{cases}
\]

It is easily seen from (7) that the content of \( R_4^{-1}(1) \) is the same as that of \( R \) above.

3. Other convolvers and their applications

We will develop several variations and applications of our systolic real-time convolver designed in the preceding section. A slight modification enables us to use \( M \) for the computation of the convolution of one finite and one infinite sequences, which is an important problem in the fields of digital filtering.

[Theorem 2] The systolic convolver \( M \) can compute the convolution \( c_i(1 = 0, 1, 2, \ldots) \) of an infinite sequence \( a_j(j = 0, 1, 2, \ldots) \) and a finite sequence \( b_k(k = 0, 1, 2, \ldots, k - 1) \) in real-time, using only \( [k/2] + O(1) \) systolic cells.

(Proof sketch) We use * as an end mark of the finite sequence \( b_k \), \( k = 0, 1, 2, \ldots, k - 1 \). The input to \( M \) is \((a_j, *)\) for any \( j \geq k \). The number of systolic cells necessary for the computation is \([k/2] + O(1)\), since the product of * and \( a_j \) is zero.

If multiplication and addition operations are interpreted as character-comparison and boolean AND, respectively, then the convolution considered above becomes the pattern matching problem[13], [15], [22]. In [22] and [29], were presented systolic pattern matching algorithms with global data broadcasting and/or preloading. In the next theorem it is shown that the both broadcasting and preloading are not necessarily essential operations in the design of real-time systolic pattern matcher.

[Theorem 3] For any infinite long "pattern" and any "text" of length \( k \), the systolic convolver \( M \) can detect and output the positions of all occurrences of the text in the pattern in real-time, using \([k/2] + O(1)\) systolic cells.

The following theorem for open convolutions of two finite sequences is easily obtained from Theorems 1 and 2.

[Theorem 4] The systolic convolver \( M \) can compute the convolution \( c_i(1 = 0, 1, 2, \ldots, n + m - 1) \) of any two finite sequences \( a_j(j = 0, 1, 2, \ldots, n - 1) \) and \( b_k(k = 0, 1, 2, \ldots, m - 1) \) in exactly \( n + m + O(1) \) steps, using \([\min(m,n)/2] + O(1)\) systolic cells.

Lastly we apply our convolver to the design of systolic real-time polynomial divider. Our design is superior to the previous ones[21], [33], and [36] in both the time and space complexities. In Table 1 we give a summary of the designs. Let's begin with the definitions. Let \( A(x) \) and \( B(x) \) be any polynomials of degree \( n \) and \( m(n \geq m) \), respectively, such that:

\[
A(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n \quad (15),
\]

\[
B(x) = b_0 x^m + b_1 x^{m-1} + \ldots + b_{m-1} x + b_m \quad (16).
\]

Then there exist unique polynomials \( C(x) \) and \( D(x) \) that satisfy the following division property:

\[
A(x) = B(x) \ C(x) + D(x), \quad \text{where} \quad \ldots \quad (17)
\]

<table>
<thead>
<tr>
<th>Design</th>
<th>Number of Systolic Cells</th>
<th>Time Complexity (cycles)</th>
<th>I/O Rates</th>
<th>I/O Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kung[21] (1985)</td>
<td>n</td>
<td>2n + m</td>
<td>1-data/2-step</td>
<td>Sequential I/O at opposite ends</td>
</tr>
<tr>
<td>Wada, Mizuno and Kawaguchi (1985)</td>
<td>n + 1</td>
<td>2n + 2</td>
<td>1-data/1-step</td>
<td>Sequential I/O at opposite ends</td>
</tr>
<tr>
<td>Lek and Hwang[36] (1985)</td>
<td>n - m + 1</td>
<td>2n - m + 2</td>
<td>1-data/1-step</td>
<td>Parallel I/O</td>
</tr>
<tr>
<td>this paper</td>
<td>([\min(n - m, m)/2]) (n) (optimum)</td>
<td>1-data/1-step</td>
<td>Sequential I/O at one end</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Performance comparison of systolic polynomial divider in the case where degree \( n \) polynomial is divided by degree \( m \) polynomial.
\[ a_i = z_1 + d_{i-n} - n - 1 \quad 0 \leq i \leq n \]  

\[ d_j = b_{ij} + c_{ij} - \sum_{j=0}^{m} b_{ij} - n - 1 \quad 0 \leq i \leq m - 1 \]  

Fig. 6 Systolic cell \( C_i \) for the polynomial divider.

We want to design a real-time systolic polynomial divider \( A \) which outputs both \( c_i (0 \leq i \leq n - m) \) and \( d_j (0 \leq j \leq m - 1) \) in exactly \( n + 0(1) \) steps, when coefficients of \( A(x) \) and \( B(x) \) are given as inputs of \( A \). The operation of \( A \) is based on \( M \) which computes the open convolution of \( (b_0, b_1, \ldots, b_n) \) and \( (c_0, c_1, \ldots, c_n) \) in [Theorem 4]. Therefore, the number of systolic cells needed for the division is \( \min(n, m + 1, m + 1/2) + 0(1) \) [\min(n, m, m/2) + 0(1)]. All of the systolic cells of \( A \), except \( C_i \), are the same as these of \( M \). So in the below, we will only give the description of \( C_i \).

The cell \( C_i \) consists of several registers shown in Fig. 6. The \( R_0 \) and \( S \) registers are added to the previous design. The \( R_0 \) is the data register storing \( a_i (0 \leq i \leq n - 1) \), and \( b_0 \), respectively. The \( S \) register assumes either \( \eta_i \) or \( \nu \) state, which notifies \( C_i \) that the current computation is for the quotient or the remainder, respectively. From time \( t = 0 \) to \( n - m + 1 \), \( S \) assumes \( \eta \) and after that, from \( t = m - n + 2 \) to the end, \( S \) assumes \( \nu \). The change of the state in \( S \) is caused by the signal fed by the host computer at time \( t = n - m + 1 \). \( C_i \) performs the operation given in Table 2 depending on whether \( S \) is \( \eta \) or \( \nu \), respectively.

In Fig. 7 we show an example of the systolic polynomial division on \( A \) in the case where \( n = 7 \) and \( m = 3 \). We give the following theorem without proof.

[Theorem 5] The division of any degree \( n \) polynomial by any degree \( m \) polynomial \( (n \geq m) \), there exists a systolic array \( A \) which can compute the polynomial division in exactly \( n + 0(1) \) steps on \( \min(n, m, m/2) + 0(1) \) systolic cells. The divider \( A \) is time-optimum and register-number-minimum.

4. Conclusion

In this work we have presented a best systolic convolver which can compute the convolution \( c_i (i = 0, \ldots, n - m - 1) \) of any two finite sequences \( a_i (j = 0, \ldots, m - 1) \) and \( b_j \) \( (k = 0, 1, \ldots, n - 1) \) in optimum real-time, exactly in \( n + m + 0(1) \) steps, using \( \min(n, m, m/2) + 0(1) \) systolic cells. The number of data registers in each systolic cell is minimum. Several variations and applications of our convolver, such as real-time pattern matcher and time-optimum polynomial divider, are also developed. Our convolver design is the best one as far as the design is within the systolic architectures.

Acknowledgement

The author is pleased to acknowledge the considerable assistance of Prof. Dr. R. Vullman. This research is supported in part by an Alexander-von Humboldt Foundation Fellowship and by the Grant-in-Aid (No.61750346) for Scientific Research of the Ministry of Education, Culture, and Science of Japan.

References


\[ z = 0: b_0^0(1) = b_0^1(1) = \Phi, (0 \leq j \leq 4), \text{ and } s^0 = q \]
\[ z = 1: b_0^0(1) = c_0^0, b_0^1(1) = b_0^0(1) = \Phi, (1 \leq j \leq 4), \text{ and } s^0 = q \]
\[ z \geq 2: \text{Given below.} \]

\[
\begin{align*}
\text{while } (s^{t-1} = "s") \text{ do } & (A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1)) ; \\
\text{if}(b_0^{t-1}(1) = \Phi) \text{ then } & (A_0^{t-1}(1) = \Phi, b_0^{t-1}(buffer) ; b_0^{t-1}(1) = \Phi) \\
\text{else if}(b_0^{t-1}(1) = \Phi) \text{ then } & (A_0^{t-1}(1) = XX, b_0^{t-1}(buffer) ; b_0^{t-1}(1) = XX) \\
\text{else if}(b_0^{t-1}(1) = \Phi) \text{ then } & (A_0^{t-1}(1) = XXX, b_0^{t-1}(buffer) ; b_0^{t-1}(1) = XXX) \\
\text{else if}(b_0^{t-1}(1) = \Phi) \text{ then } & (A_0^{t-1}(1) = XXXX, b_0^{t-1}(buffer) ; b_0^{t-1}(1) = XXXX) \\
& \text{....(0)} \\
& (A_0^{t-1}(1) = A_3^{t-1}(1), b_0^{t-1}(buffer) ; b_0^{t-1}(1) = XXXX(0)) \\
& (A_0^{t-1}(1) = A_3^{t-1}(1), b_0^{t-1}(buffer) ; b_0^{t-1}(1) = XXXX(0)) \\
& (A_0^{t-1}(1) = A_3^{t-1}(1), b_0^{t-1}(buffer) ; b_0^{t-1}(1) = XXXX(0)) \\
& \text{....}(8) \\
\end{align*}
\]

where \( \Phi, XX, XXX, XXXX, \) and \( XXXX \) are defined as follows:
\[ \Phi: A_0^{t-1}(1)/b_0^{t-1}(1) \]
\[ XX: (A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1)) \]
\[ XXX: (A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1)) \]
\[ XXXX: (A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1)) \]

\[
\begin{align*}
\text{while } (s^{t-1} = "c") \text{ do } & (A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1)) ; \\
\text{if}(b_0^{t-1}(1) = \Phi) \text{ then } & (A_0^{t-1}(1) = "c", b_0^{t-1}(buffer) ; b_0^{t-1}(1) = \Phi) \\
\text{else if}(b_0^{t-1}(1) = \Phi) \text{ then } & (A_0^{t-1}(1) = "c", b_0^{t-1}(buffer) ; b_0^{t-1}(1) = SS) \\
\text{else if}(b_0^{t-1}(1) = \Phi) \text{ then } & (A_0^{t-1}(1) = "c", b_0^{t-1}(buffer) ; b_0^{t-1}(1) = $$SS$$) \\
& \text{....}(8) \\
& (A_0^{t-1}(1) = "c", b_0^{t-1}(buffer) ; b_0^{t-1}(1) = $$SS$$) \\
& (A_0^{t-1}(1) = "c", b_0^{t-1}(buffer) ; b_0^{t-1}(1) = $$SS$$) \\
& \text{....}(8) \\
& (A_0^{t-1}(1) = "c", b_0^{t-1}(buffer) ; b_0^{t-1}(1) = $$SS$$) \\
& \text{....}(8) \\
& \text{....}(8) \\
\end{align*}
\]

where \( \Phi, SS, $$SS$$, and $$SS$$ are defined as follows:
\[ \Phi: A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1) \]
\[ SS: A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1) \]
\[ $$SS$$: A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1) \]
\[ $$SS$$: A_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = A_0^{t-1}(1), b_0^{t-1}(1) = b_0^{t-1}(1) \]

Table 2 Operation of C_i of the real-time systolic polynomial divider.
Fig. 7 Snapshots of the real-time systolic polynomial divider (t = 0 to 10)