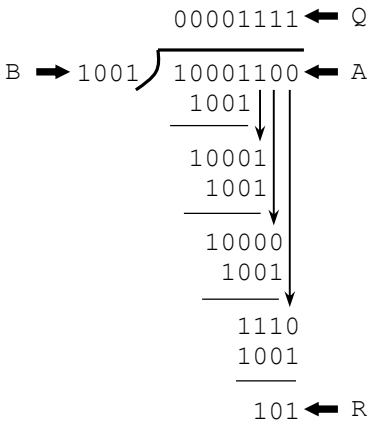
Divider Implementation

ALGORITHM

- The division of two unsigned integer numbers A/B (where A is the dividend and B the divisor), results in a quotient Q and a remainder R. These quantities are related by $A = B \times Q + R$.
  For the implementation, we follow the hand-division method. We grab bits of A one by one and comparing it with the divisor. If the result is greater or equal than B, then we subtract B from it. On each iteration, we get one bit of Q. Fig. 1 shows the algorithm as well as an example: A = 10001100; B = 1001




```
ALGORITHM

R = 0
for i = n-1 downto 0
    left shift R (input = a_i)
    if R ≥ B
       q_i = 1, R ← R-B
    else
       q_i = 0
    end
end
```

Figure 1. Division Algorithm

For hardware implementation, we consider restoring dividers (i.e., those that keep the actual residue value at every step).

SUBTRACTION OF UNSIGNED NUMBERS REPRESENTED WITH n BITS: $T = R - B$

- This point deserves special attention as the divider hardware relies on a result obtained here.
- We usually determine the sign of the subtraction by sign-extending R and B so that they are in 2's complement representation with $n+1$ bits. Then, we do: $T = R + not(B) + 1$, where $T = t_n t_{n-1} t_{n-2} \dots t_0$, and t_n determines the sign of the subtraction result.
  However, when R and B are unsigned, we can compute $not(B)$ without sign-extending B. We then analyze $c_n = cout$:
  - If $c_n = 1 \rightarrow R \geq B$ (and $R - B$ is equal to $t_{n-1} t_{n-2} \dots t_0$, i.e. it is an unsigned number with n bits)
  - If $c_n = 0 \rightarrow R < B$ (here $R - B$ is NOT equal to $t_{n-1} t_{n-2} \dots t_0$)

NOTE ABOUT THE 2'S COMPLEMENT OF ZERO

- Let A be a number in 2's complement with n bits: $A = a_{n-1} a_{n-2} \dots a_0$, where $A = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$ is the signed decimal value of A.
- The 2's complement of A is given by: $P = not(A) + 1$. $P = p_{n-1} p_{n-2} \dots p_0$
  If P and A are thought as n-bit unsigned numbers, i.e.: $A = \sum_{i=0}^{n-1} a_i 2^i$, $P = \sum_{i=0}^{n-1} p_i 2^i$ then: $P = 2^n - A$.
- What if $A = 0$? Here $P = 2^n$ requires $n+1$ bits. Why P is not zero? This is actually consistent with 2's complement arithmetic, as in the operation $Q - A$:
  $Q - A = Q + not(P) + 1$, we let cin hold the value of 1, so that if $A = 0$, then $not(A) = 11 \dots 11$ and $cin = 1$. This way, $not(A) + 1$ is properly represented. Fig. 2 shows this operation. Note that with $cin = 1$, all carries (from c_0 to c_N) are one. The result of the operation is then Q. There is no overflow as $overflow = c_n \oplus c_{n-1} = 0$. Thus, the case $A = 0$ works very well for 2's complement operations, if we include let cin carry the value of 1.

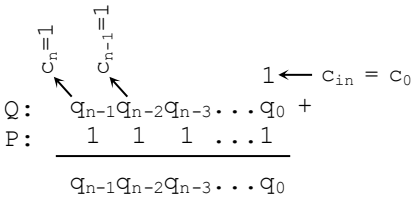


Figure 2. Q-A when A=0

COMPUTING $R - B$ WITH n bits

- $R = r_{n-1} r_{n-2} \dots r_0$, $B = b_{n-1} b_{n-2} \dots b_0$. With R, B unsigned, we have $0 \leq R, B \leq 2^n - 1$
- To do $R - B$, we sign-extend R and B to $n+1$ bits turning them into two numbers in 2's complement representation. The sign-extension actually amounts to zero-extending. Then: $R = 0 r_{n-1} r_{n-2} \dots r_0$, $B = 0 b_{n-1} b_{n-2} \dots b_0$. $r_n = b_n = 0$. In 2's complement, we have that: $0 \leq R, B \leq 2^n - 1$. It follows that: $-(2^n - 1) \leq R - B \leq 2^n - 1$. Thus $R - B$ can be represented in 2's complement with $n+1$ bits (as expected).
- Let $K = not(B) + 1$, $K = k_n k_{n-1} k_{n-2} \dots k_0$. In unsigned representation, $K = 2^{n+1} - B$.

Fig. 3 shows the operation $R - B$ by using: $R + K$, where $K = not(B) + 1$. Recall that we let 1 be held by cin. Note that if $B = 0 \rightarrow K = 2^{n+1}$ (here K is represented by the second operator as well as $cin = 1$)

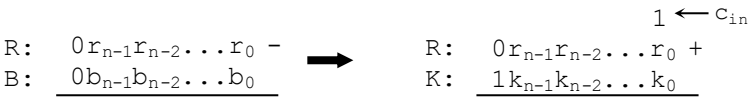


Figure 3. Operation $R - B \equiv R + K = R + not(B) + !$

Now, we determine the value of k_{n-1}:

- Case $B \neq 0$: $1 \leq B \leq 2^n - 1 \rightarrow 2^{n+1} - (2^n - 1) \leq K \leq 2^{n+1} - 1 \therefore 2^n + 1 \leq K \leq 2^{n+1} - 1$. Thus, $k_n = 1$
- Case $B = 0$: $K = 2^{n+1}$. K requires $N + 2$ bits, with $k_{n+1} = 1$, and $k_n = 0$:

	K	$k_n k_{n-1} k_{n-2} \dots k_0$	k_n
$B \neq 0$ (or $B > 0$)	2^n	**1**00...0	$k_n = 1$
	$2^n + 1$	**1**00...1	
	...	...	
	$2^{n+1} - 1$	**1**11...1	
$B = 0$	2^{n+1}	**10**00...0	$k_n = 0$

Now, we consider R, B, and K to represent unsigned integers.

$$R - B \equiv R + K = \sum_{i=0}^{n} r_i 2^i + \sum_{i=0}^{n} k_i 2^i = \sum_{i=0}^{n-1} r_i 2^i + k_n 2^n + \sum_{i=0}^{n-1} k_i 2^i$$

$$R + K = R + 2^{n+1} - B = \sum_{i=0}^{n-1} r_i 2^i + 2^{n+1} - \sum_{i=0}^{n-1} b_i 2^i$$

- $\underline{R - B < 0}$:
  Since $R \geq 0 \rightarrow B > 0 \rightarrow k_n = 1$
  $\Rightarrow R + 2^{n+1} - B = \sum_{i=0}^{n-1} r_i 2^i + 2^{n+1} - \sum_{i=0}^{n-1} b_i 2^i < 2^{n+1}$
  $\Rightarrow R + K = \sum_{i=0}^{n-1} r_i 2^i + k_n 2^n + \sum_{i=0}^{n-1} k_i 2^i < 2^{n+1} \rightarrow \sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i < 2^n$
  - The $(n+1)$-bit sum (considering the operation as unsigned) of R and K is lower than 2^{n+1}. Then, there is no overflow in the $(n+1)$- bit unsigned sum. Thus $c_{n+1} = 0$.
  - The n-bit sum (considering the operations as unsigned) of R and $k_{n-1} k_{n-2} \dots k_0$ is lower than 2^n. Thus, there is no overflow of the n-bit unsigned sum. Thus $c_n = 0$.

- $\underline{R - B \geq 0}$:
  $\Rightarrow R + 2^{n+1} - B = \sum_{i=0}^{n-1} r_i 2^i + 2^{n+1} - \sum_{i=0}^{n-1} b_i 2^i \geq 2^{n+1}$
  $\Rightarrow R + K = \sum_{i=0}^{n-1} r_i 2^i + k_n 2^n + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1} \rightarrow \sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1} - k_n 2^n$
  - The $(n+1)$-bit sum (considering the operation as unsigned) of R and K is greater or equal than 2^{n+1}. Then, there is overflow of the $(n+1)$-bit unsigned sum. Thus $c_{n+1} = 1$.
  - For the n-bit sum of R and $k_{n-1} k_{n-2} \dots k_0$, we have two cases:
    $B > 0 \rightarrow k_n = 1$. Then $\sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1} - 2^n \rightarrow \sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^n$
    $B = 0 \rightarrow k_n = 0$. Then $\sum_{i=0}^{n-1} r_i 2^i + \sum_{i=0}^{n-1} k_i 2^i \geq 2^{n+1}$
    In both cases, the n-bit sum (considering the operands as unsigned) of R and $k_{n-1} k_{n-2} \dots k_0$ is greater of equal than 2^n. So, there is overflow of the n-bit unsigned sum. Thus $c_n = 1$ when $R \geq B$.

- 2's complement operation $R - B$ with $n + 1$ bits: There is no overflow of the subtraction as $c_n = c_{n-1}$.
- For $R - B \geq 0$: The result $T = R - B$ is a positive number, thus $T_n = 0$. Therefore $t_{n-1} t_{n-2} \dots t_0$ contains $R - B$ in unsigned representation.

In conclusion:

- $If\ R < B \rightarrow c_n = 0$. The n bits $T_{n-1} T_{n-2} \dots T_0$ DO NOT contain the result $R - B$.
- $If\ R \geq B \rightarrow c_n = 1$. The n bits $T_{n-1} T_{n-2} \dots T_0$ DO represent $R - B$ in unsigned representation.

RESTORING ARRAY DIVIDER FOR UNSIGNED INTEGERS

- A, B: positive integers in unsigned representation. $A = a_{N-1}a_{N-2}\dots a_0$ with N bits, and $B = b_{M-1}b_{M-2}\dots b_0$ with M bits, with the condition that $N \geq M$. $Q = quotient,\ R = residue$. $A = B \times Q + R$.

In this parallel implementation, the result of every stage is called the remainder R_i.

Fig. 4 depicts the parallel algorithm with N stages. For each stage $i,\ i = 0, \dots, N-1$, we have:
R_i: output of stage i. Remainder after every stage.
Y_i: input of stage i. It holds the minuend.

For the next stage, we append the next bit of A to R_i. This becomes Y_{i+1} (the minuend):

$$Y_{i+1} = R_i \& a_{N-1-i}, i = 0, \dots, N-1$$

At each stage i, the subtraction $Y_i - B$ is performed. If $Y_i \geq B$ then $R_i = Y_i - B$. If $Y_i < B$, then $R_i = Y_i$.

Stage	Y_i	Computation of R_i	# of R_i bits
0	$Y_0 = a_{N-1}$	$R_0 = Y_0 - B, if\ Y_0 \geq B$ $R_0 = Y_0, if\ Y_0 < B$	1
1	$Y_1 = R_0 \& a_{N-2}$	$R_1 = Y_1 - B, if\ Y_1 \geq B$ $R_1 = Y_1, if\ Y_1 < B$	2
2	$Y_2 = R_1 \& a_{N-3}$	$R_2 = Y_2 - B, if\ Y_2 \geq B$ $R_2 = Y_2, if\ Y_2 < B$	3
…	…	…	…
M-1	$Y_{M-1} = R_{M-2} \& a_{M-N}$	$R_{M-1} = Y_{M-1} - B, if\ Y_{M-1} \geq B$ $R_{M-1} = Y_{M-1}, if\ Y_{M-1} < B$	M

Since B has M bits, the operation $Y_i - B$ requires M bits for both operands. To maintain consistency, we let Y_i be represented with M bits.

R_i: output of each stage. For the first M stages, R_i requires $i+1$ bits. However, for consistency and clarity's sake, since R_i might be the result of a subtraction, we let R_i use M bits.

For stages $0\ to\ M-2$:
R_i is always transferred onto the next stage. Note that we transfer R_i with $M-1$ least significant bits. There is no loss of accuracy here since R_i at most requires M-1 bits for stage M-2. We need R_i with M-1 bits since Y_{i+1} uses M bits.

Stages $M-1\ to\ N-1$:
Starting from stage $M-1$, R_i requires M bits. We also know that the remainder requires at most M bits (maximum value is $2^M - 2$). So, starting from stage M-1 we need to transfer M bits.
As Y_{i+1} now requires $M+1$ bits, we need $M+1$ units starting from stage M.

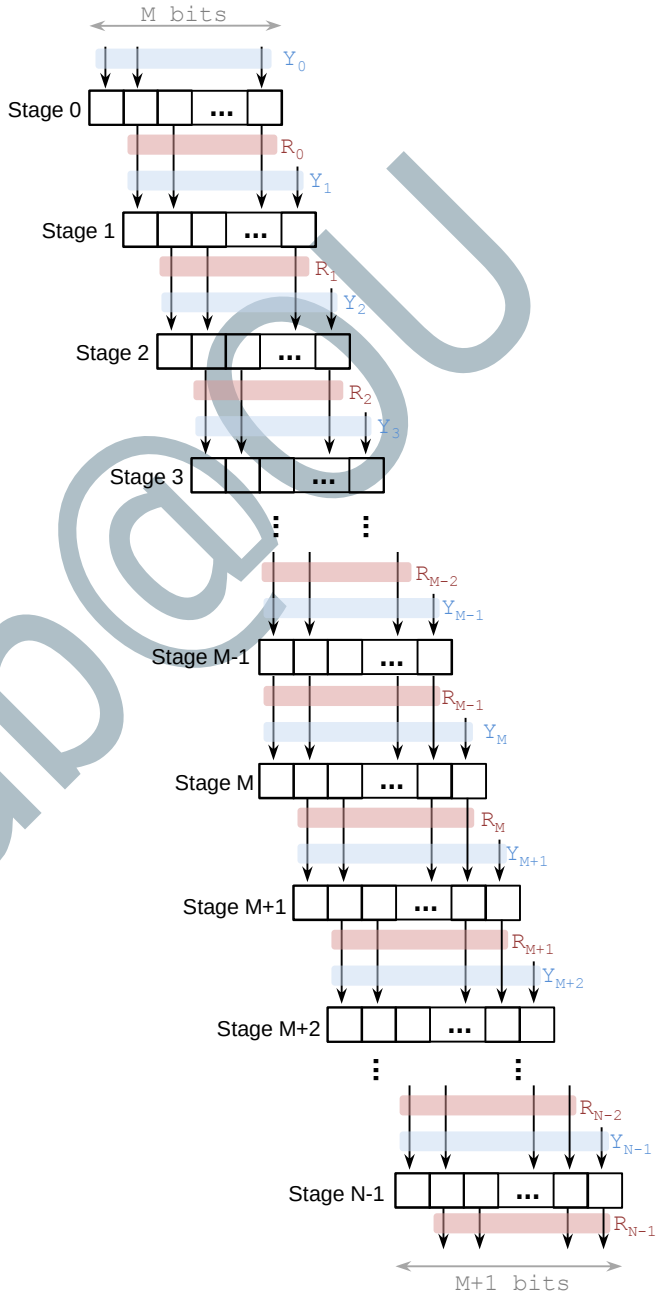


Figure 4. Parallel implementation algorithm

- To implement the operation $Y_i - B$ we use a subtractor. When $Y_i \geq B \rightarrow cout_i = 1$, and when $Y_i < B \rightarrow cout_i = 0$. This $cout_i$ becomes a bit of the quotient: $Q_i = cout_{N-1-i}$. This quotient Q requires N bits at most.
- Also, the final remainder is the result of the last stage. The maximum theoretical value of the remainder is $2^M - 2$, thus the remainder R requires M bits. $R = R_{N-1}$.
- Also, note that we should avoid a division by 0. If B=0, then, in our circuit: $Q = 2^N - 1$ and $\mathrm{R} = a_{M-1}a_{M-2}\dots a_0$.

COMBINATIONAL ARRAY DIVIDER

Fig. 5 shows the hardware of this array divider for N=8, M=4. Note that the first M=4 stages only require 4 units, while the next stages require 5 units. This is fully combinatorial implementation.

- Each level computes R_i. It first computes $Y_i - B$. When $Y_i \geq B \rightarrow cout_i = 1$, and when $Y_i < B \rightarrow cout_i = 0$. This $cout_i$ is used to determine whether the next R_i is $Y_i - B$ or Y_i.
- Each Processing Unit (PU) is used to process $Y_i - B$ one bit at a time, and to let a particular bit of either $Y_i - B$ or Y_i be transferred on to the next stage.

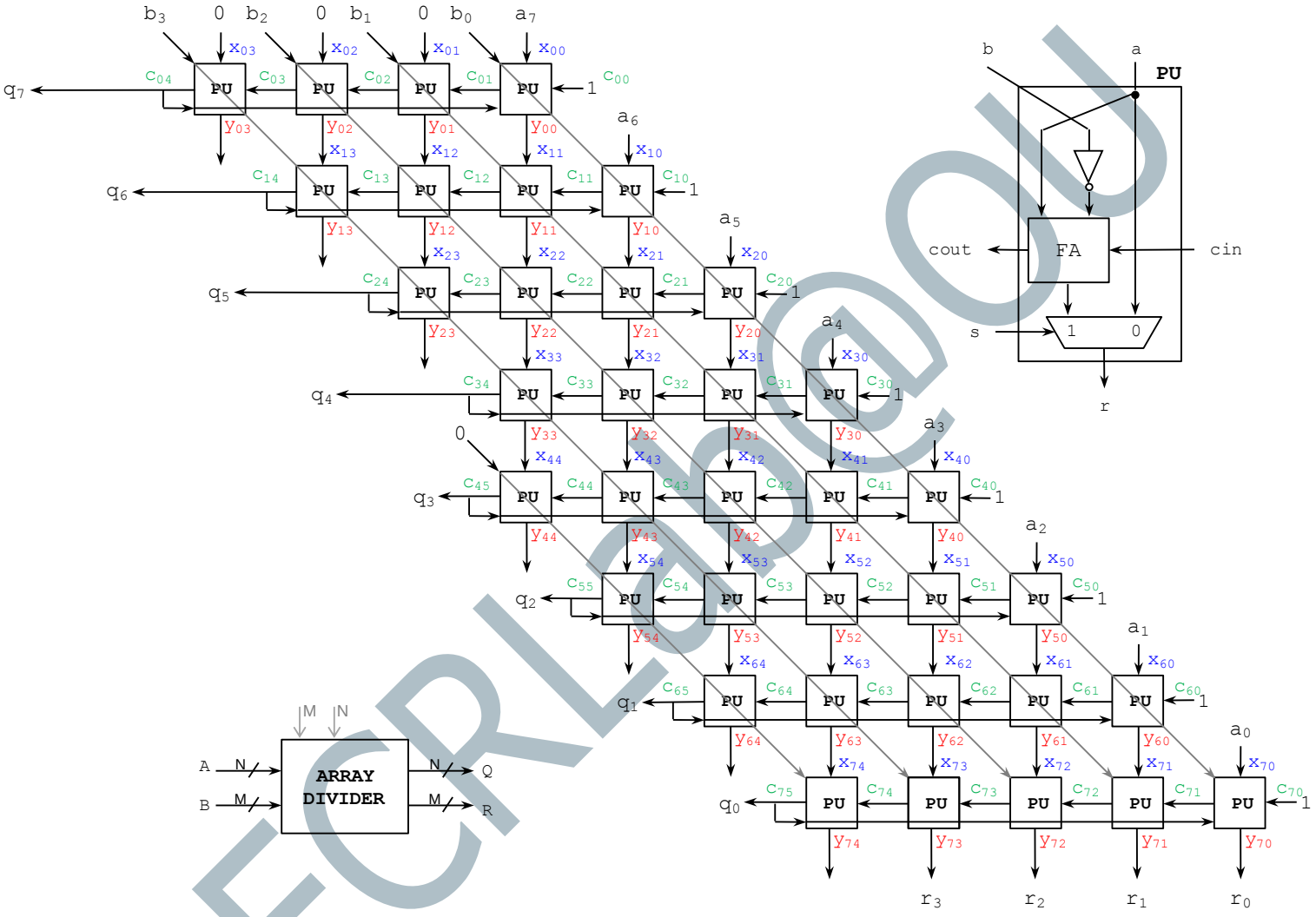


Figure 5. Fully Combinatorial Array Divider architecture for N=8, M=4

FULLY PIPELINED ARRAY DIVIDER

Fig. 6 shows the hardware core of the fully pipelined array divider with its inputs, outputs, and parameters.

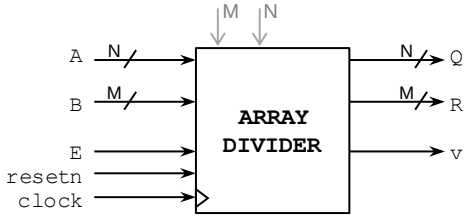


Figure 6. Fully pipelined IP core for the array divider

Fig. 7 shows the internal architecture of this pipelined array divider for N=8, M=4. Note that the first M=4 stages only require 4 units, while the next stages require 5 units. Note that the enable input 'E' is only an input to the shift register on the left, which is used to generate the valid output v. This way, valid outputs are readily signaled. If E='1', the output result is computed in N cycles (and v='1' after N cycles).

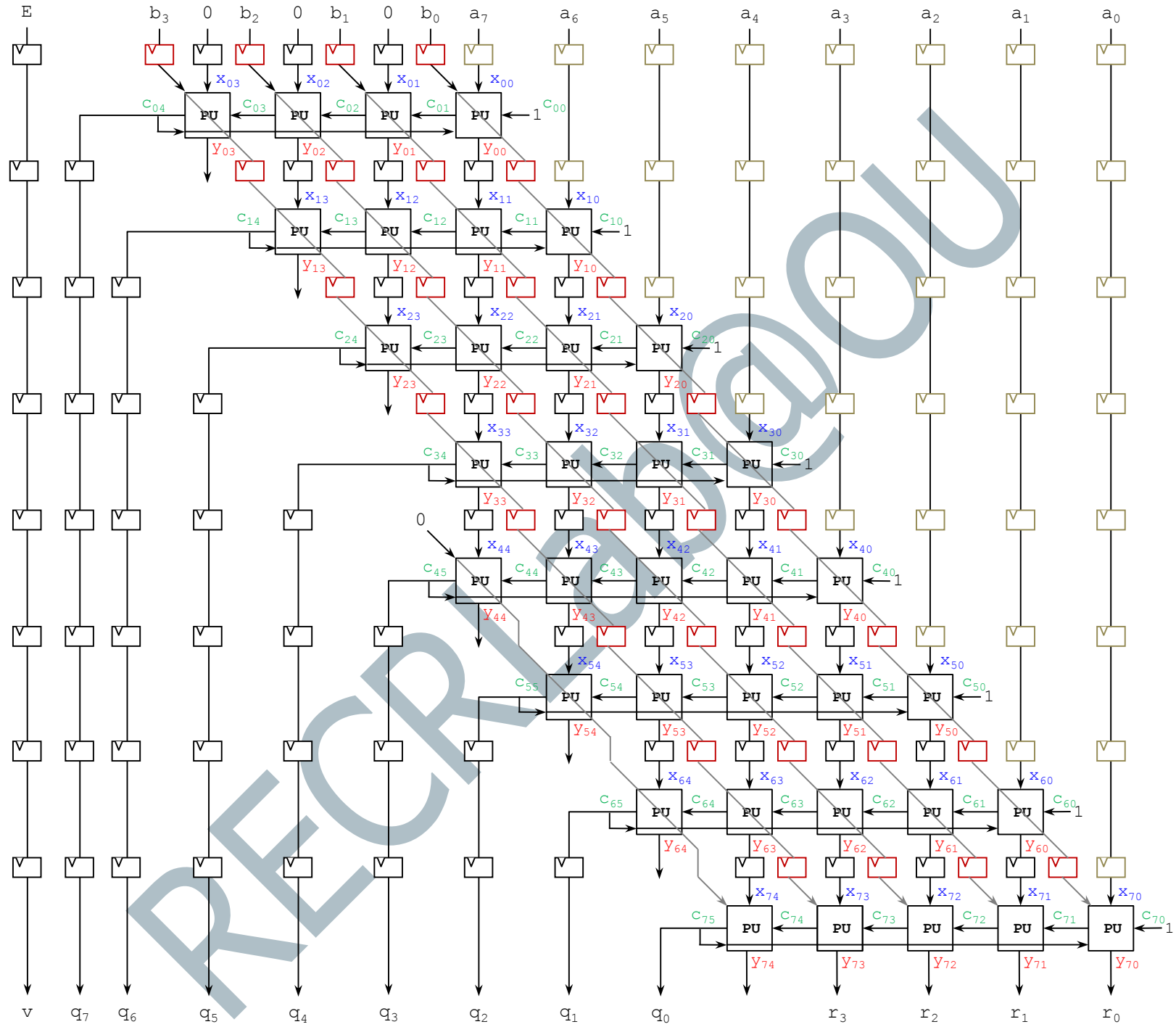


Figure 7. Fully Pipelined Array Divider architecture for N=8, M=4

ITERATIVE RESTORING DIVIDER

Fig. 8 shows the iterative hardware architecture as well as the state machine. Here, R_i is always held at register **R**. The subtractor computes $Y_i - B$. This requires $M + 1$ bits in the worst case.

- If $Y_i \geq B$ then $R_i = Y_i - B$. Yi here is the minuend. $Y_i - B$ is loaded onto register **R**. Note that only M bits are needed.
- If $Y_i < B$, then $R_i = Y_i$. Here only Y_i is loaded onto register R. This is done by just shifting a_{N-1} into register R

Note that R requires M bits since it holds the remainder at every stage. Also, since we always shift $cout_i$ onto register **A**, the quotient Q is held at A in the last iteration.

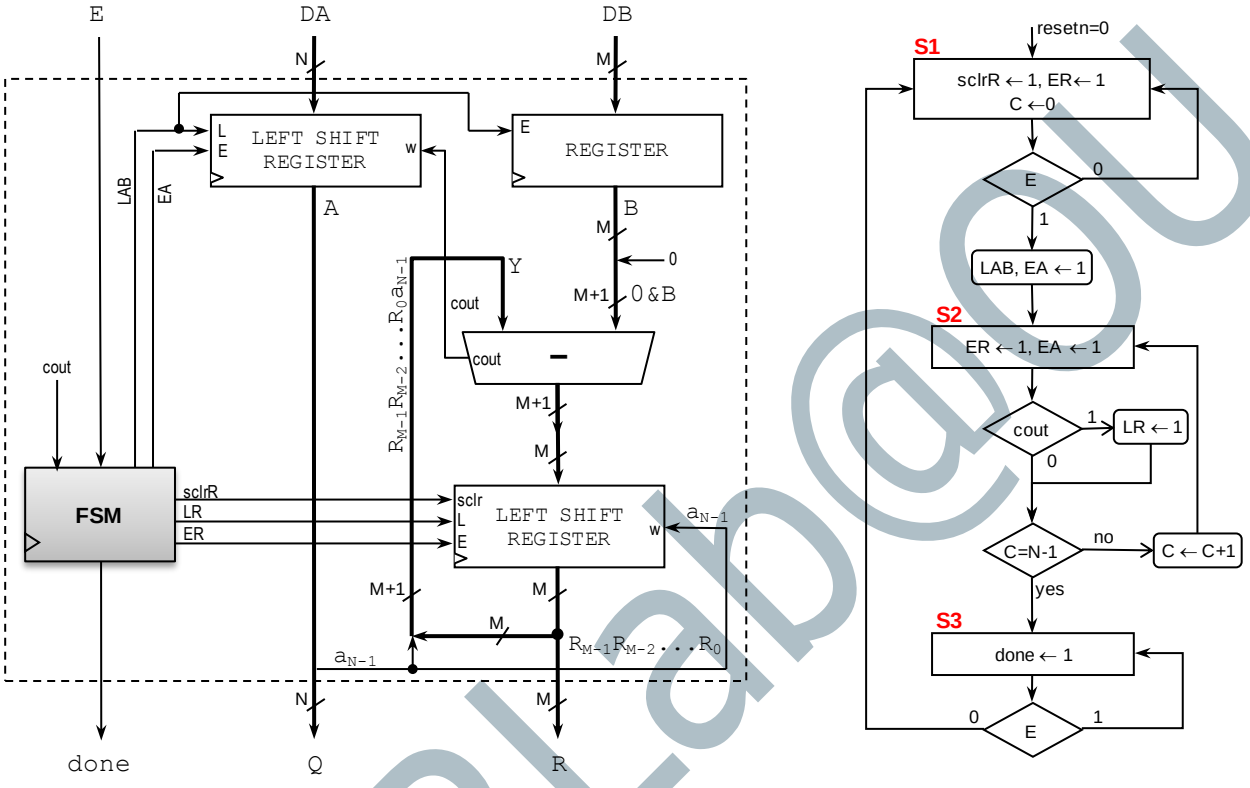


Figure 8. Iterative Divider