

Hw3 Solutions

5.4

The delay of the carry out signal for each carry-skip segment depends on either the carry generated at the beginning of the segment, or the carry-in of the segment. The delays in both cases are denoted as $T_{\text{carry}}^{\text{local}}(i)$ and $T_{\text{carry}}^{\text{global}}(i)$ respectively. $T_{\text{carry}}(i) = \text{MAX}(T_{\text{carry}}^{\text{local}}(i), T_{\text{carry}}^{\text{global}}(i))$. We should also notice that the worst carry out within each segment happens at the second last bit, which is $T_{\text{carry}}(i-1)$. The following table captures the delay property of the 2-level carry-skip adder.

Bit Position i	1	3	9	17	25	30	32
$T_{\text{carry}}^{\text{local}}(i)$	t_r	$2t_r+t_b$	$3t_r+t_b$	$4t_r+t_b$	$5t_r+t_b$	$4t_r+t_b$	$2t_r+t_b$
$T_{\text{carry}}^{\text{global}}(i)$	t_r	$2t_r+t_b$	$3t_r+t_b$	$4t_r+t_b$	$5t_r+t_b$	$6t_r+t_b$	$7t_r+t_b$
$T_{\text{carry}}(i)$	t_r	$2t_r+t_b$	$3t_r+t_b$	$4t_r+t_b$	$5t_r+t_b$	$6t_r+t_b$	$7t_r+t_b$
$T_{\text{carry}}(i-1)$	-	$2t_r$	$6t_r+t_b$	$7t_r+t_b$	$7t_r+t_b$	$7t_r+t_b$	$7t_r+t_b$

So $\text{Max}(T_{\text{carry}}(i)) = 7t_r+t_b$.

5.6

X=		0	0	0	1	0	1	1	0	1	1	0	0	1	0	1	1	0	1	0	0	1	1	1	1
Y=		0	0	1	0	0	1	1	1	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1	1
Step1	S ⁰	0	0	1	1	0	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	1	0	0	0
	C ⁰	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
	S ¹	1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	1
	C ¹	0	0	1	1	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	-
Step2	S ⁰	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1	1	0	0	1	0
	C ⁰	0		0		0		1		0		0		0		1		0		0		1		1	
	S ¹	0	1	0	0	1	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	0	1		
	C ¹	0		1		0		1		1		0		1		1		1		1		1			
Step3	S ⁰	0	0	1	1	1	1	0	1	1	1	0	0	0	0	1	0	1	1	1	1	0	1	1	0
	C ⁰	0				0				0				1				0				1			
	S ¹	0	1	0	0	1	1	1	0	1	1	0	1	0	0	1	1	0	0	0	0				
	C ¹	0				0				0				1				1							
Step4	S ⁰	0	0	1	1	1	1	0	1	1	1	0	1	0	0	1	0	0	0	0	0	0	1	1	0
	C ⁰	0								0								1							
	S ¹	0	0	1	1	1	1	1	0	1	1	0	1	0	0	1	1								
	C ¹	0								0															
Step5	S ⁰	0	0	1	1	1	1	0	1	1	1	0	1	0	0	1	0	0	0	0	0	0	1	1	0
	C ⁰	0																1							
	S ¹	0	0	1	1	1	1	0	1	1	1	0	1	0	0	1	1								
	C ¹	0																							
Step6	S	0	0	1	1	1	1	0	1	1	1	0	1	0	0	1	1	0	0	0	0	0	1	1	0

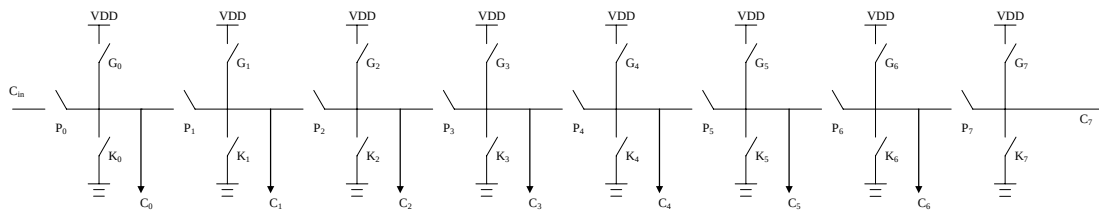
5.12

(a) (7, 7, 7, 6) counter.

(b) (9, 9, 9, 6) counter.

(c) The counter should be able to reduce m columns to a $2m$ bits binary number. Then $2^{2m}-1 \geq k \times (2^m-1) \rightarrow m \geq \lceil \log_2^{(k-1)} \rceil$.

5.20



In this case, we introduce K_i because for each C_i , both '0' and '1' are needed. K_i is used to restore '0'. However, in Fig 5.17, only logic '1' is needed to turn on the switches.

6.13

Apply $\text{NOT}(X)=1-X$ to Fig 6.24:

				$a_4(1-x_0)$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
			$a_4(1-x_1)$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
		$a_4(1-x_2)$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
	$a_4(1-x_3)$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
a_4x_4	$(1-a_3)x_4$	$(1-a_2)x_4$	$(1-a_1)x_4$	$(1-a_0)x_4$				
$1-a_4$				a_4				
1	$1-x_4$			x_4				

It's equal to the following table:

				$-a_4x_0$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
			$-a_4x_1$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
		$-a_4x_2$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
	$-a_4x_3$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
a_4x_4	$-a_3x_4$	$-a_2x_4$	$-a_1x_4$	$-a_0x_4$				
$1-a_4$	a_4	a_4	a_4	$2 \times a_4$				
1	$1-x_4$	x_4	x_4	x_4	$2 \times x_4$			

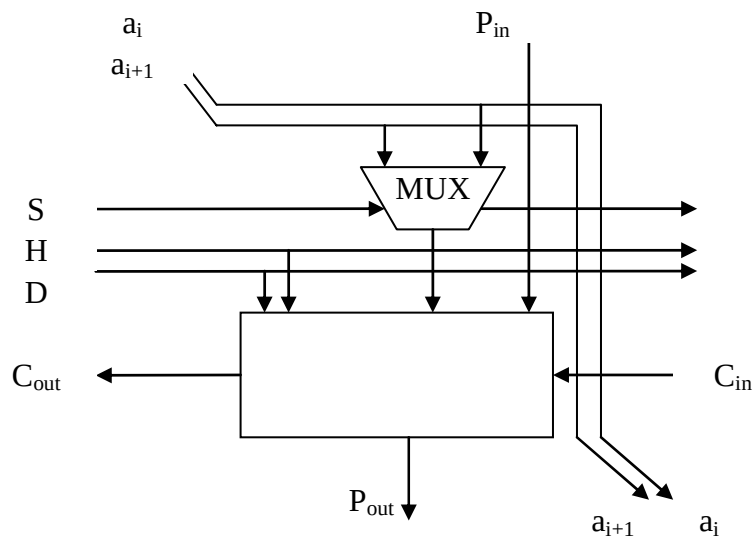
The sum of the last two rows equals 0 with a carry out. So the function of

Fig 6.24 is correct.

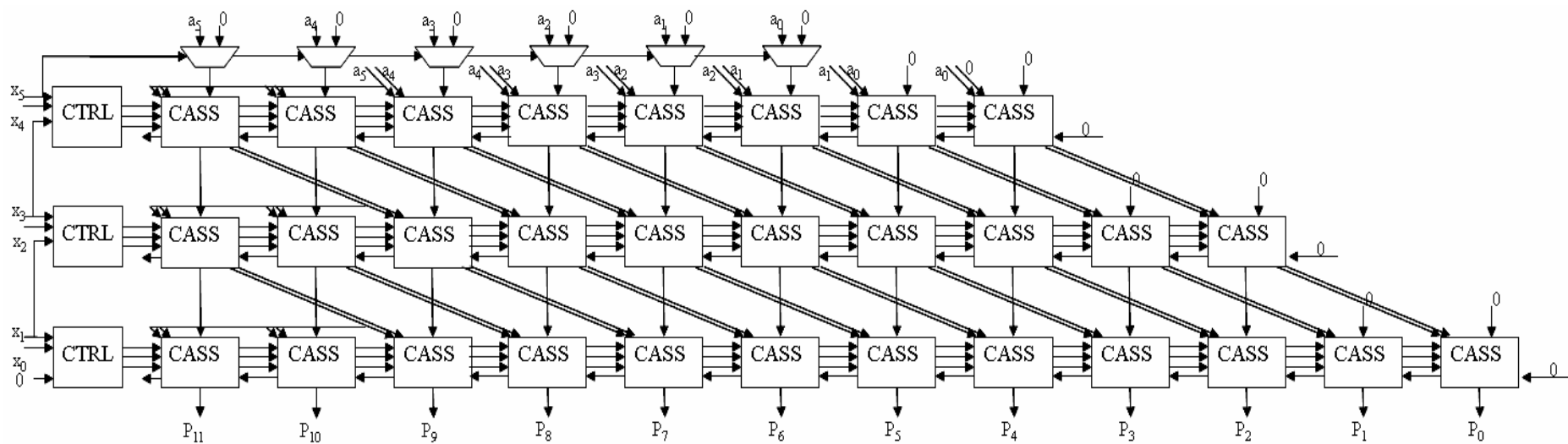
Compared with Fig 6.19, Fig 6.24 takes one more level to finish the computation, but it only needs only one kind of logic cell (full adder). Hence Fig 6.24 can be easier to be implemented. Compared with Fig 6.20, Fig 6.24 avoids sign extension and saves much on hardware cost. The penalty is two more logic levels.

6.15

The structure of CASS cells is shown as follow:



Then the diagram of the multiplier is:



6.20

Apply $\text{NOT}(X)=1-X$ to Fig 6.25:

				1	$(1-a_4x_0)$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$(1-a_4x_1)$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$(1-a_4x_2)$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$(1-a_4x_3)$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
1	a_4x_4	$(1-a_3x_4)$	$(1-a_2x_4)$	$(1-a_1x_4)$	$(1-a_0x_4)$				

It's equal to the following table:

				1	$-a_4x_0$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$-a_4x_1$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$-a_4x_2$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$-a_4x_3$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
1	a_4x_4	$-a_3x_4$	$-a_2x_4$	$-a_1x_4$	$-a_0x_4$				
		1	1	1	1				
		1	1	1	1				

The sum of all the ones in the table equals 0 with a carry out. So the function of Fig 6.25 is correct. Compared with Fig 6.24, this method needs almost the same hardware but less logic levels. So it's a better solution.