Computer Arithmetic 2016–2017 | **Homework 4 Solutions**

1 Multiplication by a constant

If each product is done on its own we need 6 adders as follows:

$$\begin{aligned} 9X &= 8X + X \\ 13X &= 8X + 4X + X \\ 18X &= 16X + 2X \\ 21X &= 16X + 4X + X \end{aligned}$$

When terms are shared many possibilities exist. A design with four adders is:

$$\begin{aligned} 9X &= 8X + X \\ 13X &= 9X + 4X \\ 18X &= 2(9X) \\ 21X &= 16X + 4X + X \end{aligned}$$

which has a time delay of two adders. Another design is:

$$\begin{aligned} 9X &= 8X + X \\ 13X &= 9X + 4X \\ 18X &= 2(9X) \\ 21X &= 18X + 2X + X \end{aligned}$$

Note that if the $2X + X$ part of the the $21X$ is prepared in parallel with the $9X$ part then the total delay of the whole system to get the four products is only that of two adders.

A design with three adders only is:

$$\begin{aligned} 9X &= 8X + X \\ 13X &= 9X + 4X \\ 18X &= 2(9X) \\ 21X &= 13X + 8X \end{aligned}$$

but in this design the critical path delay is three adders.

2 Multipliers with signed digits

1. It is important to understand the logical function implemented by each part and how it leads to a mathematical relation between the inputs and outputs. For example, the multiplexer with i_1 as its select line and i_2 and its complement as the inputs yields the output $i_1\bar{i}_2 + \bar{i}_1 i_2 = i_1 \oplus i_2$. Similarly, at the bottom of the figure, we get $i_3 \oplus i_4$ then we get $(i_1 \oplus i_2) \oplus (i_3 \oplus i_4)$ and finally $s_1 = c'_{in} \oplus (i_1 \oplus i_2) \oplus (i_3 \oplus i_4)$ which is equivalent to a modulo 2 summation of all these inputs.

   On the other hand, $c'_{out} = i_3(i_1 \oplus i_2) + i_1\overline{i_1 \oplus i_2} = i_3(i_1 \oplus i_2) + i_1 i_2$ which is equal to a carry signal for the sum of those three inputs alone and is independent of i_4 and independent of c'_{in}.

   Similarly, $c''_{out} = c'_{in}((i_1 \oplus i_2) \oplus (i_3 \oplus i_4)) + i_4\overline{(i_1 \oplus i_2) \oplus (i_3 \oplus i_4)} = c'_{in}((i_1 \oplus i_2) \oplus (i_3 \oplus i_4)) + i_4(i_1 \oplus i_2 \oplus i_3)$ which is a carry signal for c'_{in}, i_4, and the value $i_1 \oplus i_2 \oplus i_3$ (the modulo 2 sum of the first three inputs).

So, in general, we are getting a $[4:2]$ compressor out of this circuit. The mathematical relation is thus (here the + sign indicates addition):

$$2(c''_{out} + c'_{out}) + s_1 + s_2 = i_1 + i_2 + i_3 + i_4 + c'_{in} + c''_{in} \quad (1)$$

2. The complement of bit i_1 is equal to $1 - i_1$. Using this fact to substitute in equation 1 yields the new relation:

$$\begin{aligned} 2(c''_{out} + (1 - c'_{out})) + (1 - s_1) + s_2 &= (1 - i_1) + i_2 + (1 - i_3) + i_4 + (1 - c'_{in}) + c''_{in} \\ 2(c''_{out} - c'_{out}) - s_1 + s_2 &= -i_1 + i_2 - i_3 + i_4 - c'_{in} + c''_{in} \end{aligned} \quad (2)$$

or simply that we have the equivalent of negatively valued bits at the locations of the inverters. This can be used to represent signed digits for example.

3. With vertical connections between the carries we get a row of $[4:2]$ compressors. It is important to note that in this case, there is no need to invert c'_{out} of one compressor and invert again c'_{in} of the following compressor. The two inverters cancel each other.

   With horizontal connections, we get a tree of compressors similar to the conventional multiplier trees.

4. If the inverted outputs are connected to inverted inputs then they cancel each other and there is no need for them anywhere inside the tree. Only the inputs and the outputs of the whole tree (but not inside it) may have some inverters.

5. According to the results of this problem, the use of signed digits in multiplication is as simple as the use of unsigned digits since the whole body of the tree is similar and only some additional inverters are needed in some locations at its boundary.

---

3 A new divider design using multiplication

1. The LSB is padded by a 0 to its right. Pad the MSB with two 0 if n is even and one 0 if n is odd.

2. The Y_h part represents normal bits which have a positive mathematical value hence we use a regular recoder.

Original bits				
y_{j+1}	y_j	c_{in}	c_{out}	value
0	0	0	0	+0
0	0	1	0	+1
0	1	0	0	+1
0	1	1	0	+2
1	0	0	1	−2
1	0	1	1	−1
1	1	0	1	−1
1	1	1	1	−0

3. In $Y_h - Y_l$, the Y_l part represents bits which have a negative mathematical value hence we may use a recoder with negative values.

Original bits				
y_{j+1}	y_j	c_{in}	c_{out}	value
0	0	0	0	−0
0	0	1	0	−1
0	1	0	0	−1
0	1	1	0	−2
1	0	0	−1	+2
1	0	1	−1	+1
1	1	0	−1	+1
1	1	1	−1	+0

4. The boundary region between Y_h and Y_l depends on whether i (the index of the MSB of Y_l) is odd or even.

i **even:** This means that the number of bits within Y_l is odd. Hence, the two "new" bits of the group at the boundary of Y_l and Y_h are one from Y_l and the other from Y_h. The carry into this group comes from a group entirely within Y_l. We can choose to recode the values as:

Original bits				
y_{i+1}	y_i	c_{in}	c_{out}	value
0	0	0	0	-0
0	0	-1	0	-1
0	-1	0	0	-1
0	-1	-1	0	-2
1	0	0	0	$+2$
1	0	-1	0	$+1$
1	-1	0	0	$+1$
1	-1	-1	0	$+0$

where the c_{out} signal is always 0 which means that the next higher up group will only have

Original bits				
y_{i+3}	y_{i+2}	c_{in}	c_{out}	value
0	0	0	0	$+0$
0	1	0	0	$+1$
1	0	0	1	-2
1	1	0	1	-1

as the possible cases. These are a subset of the cases in the regular recoder of the Y_h part.

i **odd:** This means that the number of bits within Y_l is even. Hence, the most significant two bits of Y_l are taken within a group using the modified recoder of the last step and may produce a negative carry to the higher up group. However, that higher group is the first two bits of Y_h which are positively valued. Similar to what we have just done above, we can choose to recode the values as:

Original bits				
y_{i+2}	y_{i+1}	c_{in}	c_{out}	value
0	0	0	0	-0
0	0	-1	0	-1
0	1	0	0	$+1$
0	1	-1	0	$+0$
1	0	0	0	$+2$
1	0	-1	0	$+1$
1	1	0	0	$+3$
1	1	-1	0	$+2$

where the c_{out} signal is always 0 which means that the next higher up group will only have

Original bits				
y_{i+4}	y_{i+3}	c_{in}	c_{out}	value
0	0	0	0	$+0$
0	1	0	0	$+1$
1	0	0	1	-2
1	1	0	1	-1

as the possible cases. These are a subset of the cases in the regular recoder of the Y_h part. However, with this choice of recoding the case of 1 1 0 in the boundary group produces a hard multiple of $+3$.

In order to avoid that hard multiple, we can recode the boundary group as:

Original bits				
y_{i+2}	y_{i+1}	c_{in}	c_{out}	value
0	0	0	0	-0
0	0	-1	0	-1
0	1	0	0	$+1$
0	1	-1	0	$+0$
1	0	0	1	-2
1	0	-1	0	$+1$
1	1	0	1	-1
1	1	-1	1	-2

where the c_{out} signal is not a simple function. Specifically, the case 1 0 1 must produce a zero carry to the next higher group which can use a regular recoder of the Y_h part.

Thus we end up with this table (with d indicating a don't care value)

Original bits			Y_h	Y_l	Boundary				First Y_h	
y_{j+1}	y_j	c_{in}	value	value	i even		i odd		i even	i odd
0	0	0	$+0$	-0	0	-0	0	-0	$+0$	$+0$
0	0	1	$+1$	-1	0	-1	0	-1	d	$+1$
0	1	0	$+1$	-1	0	-1	0	$+1$	$+1$	$+1$
0	1	1	$+2$	-2	0	-2	0	$+0$	d	$+2$
1	0	0	-2	$+2$	0	$+2$	1	-2	-2	-2
1	0	1	-1	$+1$	0	$+1$	0	$+1$	d	-1
1	1	0	-1	$+1$	0	$+1$	1	-1	-1	-1
1	1	1	-0	$+0$	0	$+0$	1	-2	d	-0

4 A different arrangement for partial products

1. Several proofs are possible. Any correct proof gets the full mark. Here is one solution.
   Since $\overline{a_j x_i} = 1 - (a_j x_i)$ then we can rewrite all the elements containing the complement of a bit in this manner to get

						a_4	a_3	a_2	a_1	a_0
					$\times$	x_4	x_3	x_2	x_1	x_0
						$1 - a_4x_0$	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			
		a_4x_4	$1 - a_3x_4$	$1 - a_2x_4$	$1 - a_1x_4$	$1 - a_0x_4$				
1			0	0	1					
p_9		p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

   which is then rearranged as

						a_4	a_3	a_2	a_1	a_0
					$\times$	x_4	x_3	x_2	x_1	x_0
						$-a_4x_0$	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	1	1	1				
			1	1	1	1				
1			0	0	1					
p_9		p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

   The position of those resulting ones is such that they add up to overflow as a carry that is neglected and produce a zero in the range of bits representing the product. Hence, the array is equal to:

						a_4	a_3	a_2	a_1	a_0
					$\times$	x_4	x_3	x_2	x_1	x_0
						$(-a_4)x_0$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
					$(-a_4)x_1$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
				$(-a_4)x_2$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
			$(-a_4)x_3$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
		$(-a_4)(-x_4)$	$a_3(-x_4)$	$a_2(-x_4)$	$a_1(-x_4)$	$a_0(-x_4)$				
p_9		p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

   where we used the fact that $a_4x_4 = (-a_4)(-x_4)$.

   Since in the two's complement format the most significant bit is negatively valued, this resulting array effectively produces the product of two five bits operands.

2. The easiest re-arrangement is to move the location of the constants.

					a_4	a_3	a_2	a_1	a_0
				$\times$	x_4	x_3	x_2	x_1	x_0
				1	$\overline{a_4x_0}$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$\overline{a_4x_1}$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$\overline{a_4x_2}$	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		$\overline{a_4x_3}$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
1	a_4x_4	$\overline{a_3x_4}$	$\overline{a_2x_4}$	$\overline{a_1x_4}$	$\overline{a_0x_4}$				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

3. It has $n+1$ rows and $2n$ columns.

4. The values of i go from 0 to n while the values of j go from 0 to $2n-1$. The bit $pp_{i,j}$ at row i and column j is given by

Range of i	Range of j	$pp_{i,j}$
$i < n-1$	$0 \le j < i$	0
	$i \le j < i+n-1$	$a_{j-i}x_i$
	$j = i+n-1$	$\overline{a_{j-i}x_i}$
	$i+n-1 < j$	0
$i = n-1$	$0 \le j < i$	0
	$i \le j < i+n-1$	$\overline{a_{j-i}x_i}$
	$j = i+n-1$	$a_{j-i}x_i$
	$i+n-1 < j$	0
$i = n$	$j = n$	1
	$j = 2n-1$	1
	all other j	0

---

5 Sum of squares

1. With the given specification, $x \in \{-2^{n-1}, \cdots, +(2^{n-1}-1)\}$ and hence $x^2 \in \{0, \cdots, 2^{2n-2}\}$. Thus x^2 is fully represented in $(2n-1)$ bits. Similarly for y^2. The value of z is unsigned so we have

$$\begin{aligned} x^2 &\in \{0, \cdots, 2^{2n-2}\} \\ y^2 &\in \{0, \cdots, 2^{2n-2}\} \\ z &\in \{0, \cdots, 2^{2n-1}\} \end{aligned}$$

and $s \in \{0, \cdots, 2^{2n}\}$ which means that s is represented in $2n+1$ bits.

2. The resulting array when $n = 4$ is

				x_3	x_2	x_1	x_0
			$\times$	x_3	x_2	x_1	x_0
				$\overline{x_3x_0}$	x_2x_0	x_1x_0	x_0x_0
			$\overline{x_3x_1}$	x_2x_1	x_1x_1	x_0x_1	
		$\overline{x_3x_2}$	x_2x_2	x_1x_2	x_0x_2		
	x_3x_3	$\overline{x_2x_3}$	$\overline{x_1x_3}$	$\overline{x_0x_3}$			
1			1				
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

which reduces to

				x_3	x_2	x_1	x_0
			$\times$	x_3	x_2	x_1	x_0
				$2\overline{x_3x_0}$	$2x_2x_0$	$2x_1x_0$	x_0
			$2\overline{x_3x_1}$	$2x_2x_1$	x_1		
		$2\overline{x_3x_2}$	x_2				
	x_3						
1			1				
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

then to

				x_3	x_2	x_1	x_0
			$\times$	x_3	x_2	x_1	x_0
1	$\overline{x_3x_2}$	$\overline{x_3x_1}$	$\overline{x_3x_0}$	x_2x_0	x_1x_0		x_0
	x_3		x_2x_1		x_1		
			x_2				
			1				
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

and finally to

				x_3	x_2	x_1	x_0
			$\times$	x_3	x_2	x_1	x_0
1	$\overline{x_3x_2}$	$\overline{x_3x_1}$	$\overline{x_3x_0}$	x_2x_0	x_1x_0		x_0
	x_3	x_2x_1	$x_2\overline{x_1}$		x_1		
			1				
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

3. For s we have

	1	$\overline{x_3x_2}$	$\overline{x_3x_1}$	$\overline{x_3x_0}$	x_2x_0	x_1x_0		x_0
		x_3	x_2x_1	$x_2\overline{x_1}$		x_1		
				1				
	1	$\overline{y_3y_2}$	$\overline{y_3y_1}$	$\overline{y_3y_0}$	y_2y_0	y_1y_0		y_0
		y_3	y_2y_1	$y_2\overline{y_1}$		y_1		
				1				
	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

which reduces to

1		$\overline{x_3x_2}$	$\overline{x_3x_1}$	$\overline{x_3x_0}$	x_2x_0	x_1x_0		x_0
		x_3	x_2x_1	$x_2\overline{x_1}$		x_1		
			1					
		$\overline{y_3y_2}$	$\overline{y_3y_1}$	$\overline{y_3y_0}$	y_2y_0	y_1y_0		y_0
		y_3	y_2y_1	$y_2\overline{y_1}$		y_1		
	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

4. Should be written in the full answer.
