

Multiplier Circuits

Dinesh Sharma

EE Department
IIT Bombay, Mumbai

October 8, 2018

1 Shift and Add Multipliers

2 Array Multipliers

3 Speeding up Multipliers

- Booth Encoding
- Adding Partial Products
- Wallace Multipliers
- Dadda Multipliers

4 Serial Multipliers

- Bit Serial Multipliers
- Row Serial multipliers

Shift and Add Multipliers

An obvious way for implementing multipliers is to replicate the paper and pencil procedure in hardware.

The diagram illustrates the paper-and-pencil multiplication procedure using circles to represent bits. The multiplicand is 00000 (5 bits) and the multiplier is 0000 (4 bits). The first two bits of the multiplicand are highlighted in red. The partial products are shown shifted to the left, and the final 8-bit product is 00000000.

$$\begin{array}{r}
 \textcolor{red}{0} \textcolor{red}{0} 0000 \\
 \times \quad 0000 \\
 \hline
 0000 \\
 0000 \\
 0000 \\
 0000 \\
 \hline
 00000000
 \end{array}$$

- Initialize the product to 0, extend multiplicand to left by n bits filled with 0s.
- If the least significant bit of the multiplier is 1, add the multiplicand to product, else do nothing.
- Shift the multiplier right by one bit.
- Shift the multiplicand left by one bit.
- Repeat for n bits

Shift and Add Multipliers

- Each term being added to form the product is called a partial product.
- The name “partial product” is also used for individual bits of the terms being added - so beware!
- The paper-pencil procedure requires $n-1$ additions to a $2n$ bit accumulator.
- This uses a single adder, but takes long to complete the multiplication. A 32×32 multiplication will require 31 addition steps to a 64 bit accumulator.
- Multiplication can be made faster by using multiple adders and adding terms in a tree structure.

Array Multiplier

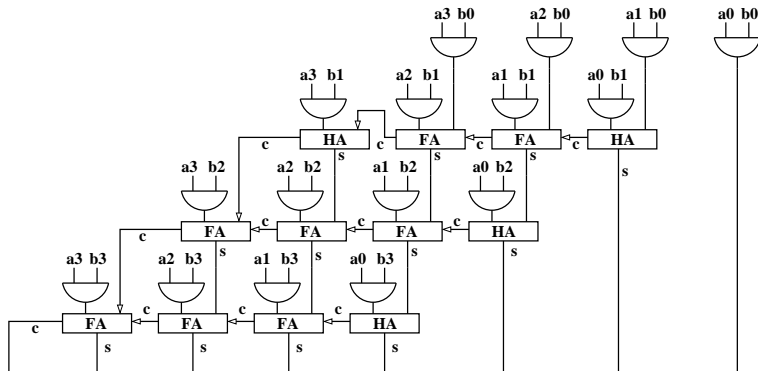
Suppose we want to multiply two n-bit numbers A and B, where

$$A = \sum_{i=0}^{n-1} 2^i a_i \quad B = \sum_{j=0}^{n-1} 2^j b_j$$

- We can regard all bits of the partial products as an array, whose (i,j)th element is $a_i \cdot b_j$. Notice that each element is just the AND of a_i and b_j .
- **All** elements of the array are available in parallel, within one gate delay of arrival of A and B.
- We can now use an array of full adders to produce the result. One input of each adder is the sum from the previous row, the other is the AND of appropriate a_i and b_j .
- This architecture is called an array multiplier.

Array Multiplier

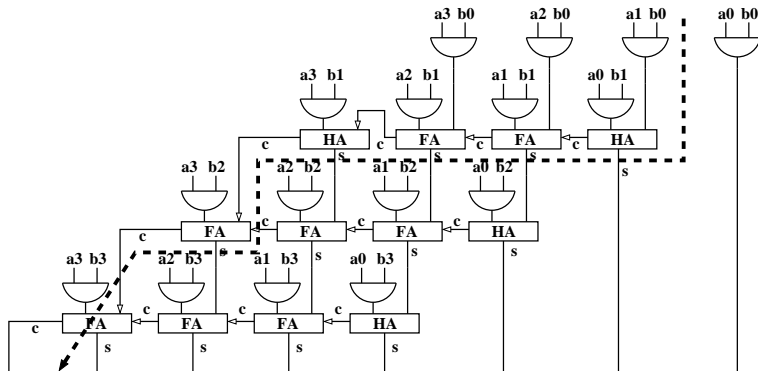
A 4X4 array multiplier is shown below.



Half adders can be used at the right end.

Critical Path through Array Multiplier

The critical path through a 4X4 array multiplier is shown below.



The critical path involves carry as well as sum outputs!

Speeding up Multipliers

The array multiplier has a regular layout with relatively short connections. However, it is still rather slow.

How can we speed up a multiplier?

There are two possibilities:

- Somehow reduce the number of partial products to be added. For example, could we multiply 2 bits at a time rather than 1?
- Since we have to add more than two terms at a time, use an adder architecture which is optimized for this.

Booth Encoding

Booth Encoding reduces the number of partial products by multiplying 2 bits at a time.

- Let the multiplicand be A and the multiplier B .
- Rather than multiplying A with successive bits of B , we can multiply it with two bits of B at a time.
- Depending on the two bits being 00, 01, 10 or 11, the partial product will be 0, A , $2A$ or $3A$.
- 0 and A can be produced trivially.
- $2A$ can be produced easily by a left shift of A .
- Generating $3A$ presents a problem!

Booth Encoding

Multiplying the Multiplicand A by 2 bits of the multiplier at a time requires the generation of 0, A , $2A$ or $3A$ as partial products.

Generating 0, A or $2A$ is easy.

- $3A$ cannot be generated directly. However, $3A$ can be expressed as $4A - A$.
- The task of adding $4A$ is passed on to the next group of 2 bits of the multiplier.
- Since the place value of the next group of 2 bits is 4 times the current one, adding $4A$ to the product is equivalent to adding 1 to the next group of 2 bits of the multiplier.
- $-A$ can be generated from A , using an adder/subtractor rather than an adder for accumulating the sum of partial products.

Modified Booth Encoding

To simplify the logic for deciding whether an additional $4A$ should be added on behalf of the less significant 2 bits in the multiplier, we express $2A$ also as $4A - 2A$.

- Since we anyway have an adder-subtractor, this requires no additional resources.
- The modified logic is: for 00, do nothing. For 01, add A .
for 10, subtract $2A$, ask the next group to add $4A$.
for 11, subtract A , ask the next group to add $4A$.
- Now the next group can just look at the more significant bit of the previous group and add 1 to the multiplier if it is '1'.

Modified Booth Encoding

- The partial product generator looks at the current 2 bits and the MSB of the previous group of 2 bits to decide its action.
- Thus, we scan the multiplier 3 bits at a time, with one bit overlapping.
- For the first group of 2 bits, we assume a 0 to the right of it.
- After handling the previous group, the multiplicand is shifted left by 2 positions. Thus, it has already been multiplied by 4.
- Therefore, adding 4 A on behalf of the previous group is equivalent to adding 1 to the multiplier corresponding to the current group.

Modified Booth Encoding

The following table summarizes the effective multiplier for generating the partial product.

Current 2-bits	Multiplier for these	Previous MSBit	Pending Increment	Total Multiplier
00	0	0	0	0
01	+1	0	0	+1
10	-2	0	0	-2
11	-1	0	0	-1
00	0	1	+1	+1
01	+1	1	+1	+2
10	-2	1	+1	-1
11	-1	1	+1	0

Notice that a 111 in the 3 bit group being scanned requires no work at all.

Modified Booth Encoding

3-bits	Multiplier
000	0
001	+1
010	+1
011	+2
100	-2
101	-1
110	-1
111	0

What happens if there is a string of '1's in the multiplier?

- There will be a -1 in the beginning, because the group begins with 110.
- Similarly, there will be a +1 at the end, because it will end with 011.
- However, for the length of continuous '1's, nothing needs to be done (add zeros).

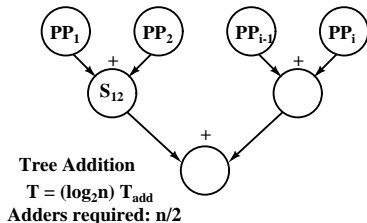
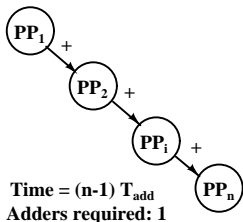
Thus Booth encoding reduces the number of partial products to half (multiplying 2 bits at a time).

It makes addition in columns of partial products fast because carry propagation during addition will be reduced.

Adding Partial Products

Multipliers can be speeded up by using special adder architectures which are optimized for adding more than two numbers.

- One option is to use tree adders rather than an accumulator.
- Several additions proceed in parallel, since all partial products are generated together.



Carry Save Adders

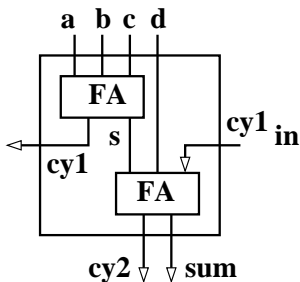
- Ordinary adders are large and complex. Also, these are slow due to rippling of carry.
- Let us consider an adder which presents its output not as one word - but two. The actual result is the sum of these.
- Obviously, an adder of this type is of no use for adding just two numbers! But it can be useful in a multiplier where we are adding multiple terms.
- For each bit column, the sum goes into one output word, while carry outs go into the other (without being added to the next more significant column).
- Now there is no rippling of carry and the output is available in constant time.
- We need a conventional adder in the end to add these two words.
- This type of adder is called a “Carry Save Adder” or CSA.

Carry Save Adders

- A Carry Save Adder (whose output is two words which must be added to produce the result) is of no use for adding just two words!
- However, we can construct a useful CSA for adding 4 bits in the same column.
- We make use of the fact that all partial product bits are available in constant time after the application of inputs.
- Since there are multiple bits to be added, we can feed three of them to a full adder.
- The sum and carry output of this adder is then available in constant time.

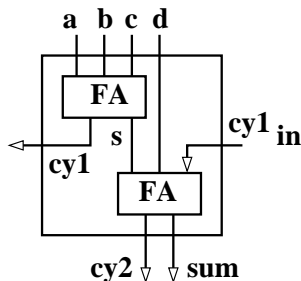
Carry Save Adders

The 4 input 2 output CSA uses two full adders as shown below:



- The first Full adder uses 3 bits of partial products of the same weight (bits in the same column).
- These are available in parallel in constant time.
- The sum output of first FA goes to the second FA.
- The carry output (**cy1**) of the first FS goes as intermediate input to the CSA used in the column to the left of this one.

Carry Save Adders

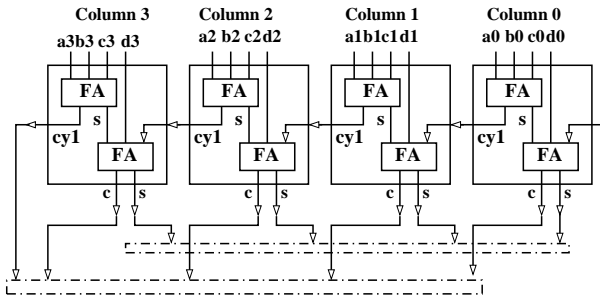


- The second FA accepts one additional bit from the partial product column, the sum output of the first Full Adder FA1 and cy1 output coming from the CSA to its right.
- All inputs to this Full Adder are also available in constant time.
- Notice that even though cy1 goes from one column to the next significant column, **it does not ripple all the way** horizontally.
- It goes to FA2 of the more significant column whose output is not required by the next column.

Tiling Carry Save Adders

The figure below shows how we can add 4 columns of 4 bits each.

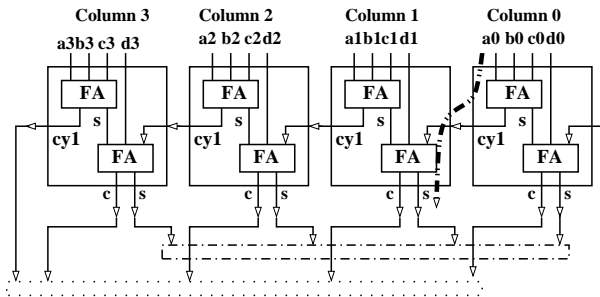
Rows are labeled as a,b,c and d. Columns are 0,1,2 and 3.



- Outputs are collected in two separate registers (shown in dotted lines).
- These must be added using a conventional adder.

Critical Path of Carry Save Adders

Critical path of a 4x4 Carry Save Adder is shown below.



One can see that the critical path has been broken up.
Addition of 4 words of 32 bits each will also have a critical path of the same length.

Wallace Multipliers

- Multipliers do not have the same number of bits in every column.
- In 1964, Wallace proposed a method for a carry save like reduction scheme which is valid for columns of variable length.
- Wallace multiplier uses adders which take multiple inputs of the same weight and produce sum outputs of the same weight and carry outputs with higher weights.
- These are combined in stages to reduce the number of terms at each weight to 2 or less.
- These two terms are then added by a conventional adder to produce the final result.

Wallace Multipliers

Wallace multipliers act in three stages:

- 1 Generate all bits of the partial products in parallel.
- 2 Collect all partial products bits with the same place value in bunches of wires and reduce these in several layers of adders till each weight has no more than two wires.
- 3 For all bit positions which have two wires, take one wire at corresponding place values to form one number, and the other wire to form another number.

Add these two numbers using a fast adder of appropriate size.

Reduction Stage of Wallace Multipliers

- We assume that Full adders and Half adders will be used.
- A full adder takes 3 inputs and produces one output of the same weight (sum) and another of higher weight (carry). This is called a (3,2) adder. It reduces the number of wires at its own weight by 2 and adds one wire at the higher weight.
- A half adder takes 2 inputs and produces one output of the same weight (sum) and another of higher weight (carry). This is a (2,2) adder. It reduces the number of wires at its own weight by 1 and adds one wire at the higher weight.
- The reduction algorithm is general and can be used with any adders of type (n,m) . For example, a carry save adder is of type (4,2).

Reduction Stage of Wallace Multipliers

- Each reduction stage looks at the number of wires for each weight and if any weight has more than 2 wires, it adds a layer of adders.
- When the numbers of wires for each weight have been reduced to 2 or less, we form one number with one of the wires at corresponding place values and another with the other wire (if present).
- These two numbers are added using a fast adder of appropriate size to generate the final product.

Reduction Stage of Wallace Multipliers

- Partial products are grouped in multiples of three rows.
- Rows which are additional over multiples of three are just passed on to the next stage.
- Wires are now reduced for each group of 3 rows.
- For any group of 3 rows, if we find 3 wires for any weight, we place a Full Adder, which generates 1 wire of the same weight and 1 wire with the next higher weight.
- If there are two wires left, we place a half adder to reduce these.
- If only one wire is left, it is carried through to the next layer.

Reduction Stage of Wallace Multipliers

- The reduction procedure is carried out for all weights, starting from the least significant weights.
- At the end of each layer, we count wires for each weight again, and if none has more than 2 wires, we proceed to the final addition stage.
- If any weight has 3 or more wires, we add another layer, and repeat this procedure till the number of wires for all weights is reduced to 2 or less.
- Now we compose one number from one of the left over wires at corresponding weights and another from the remaining wires.
- Finally, we use a conventional fast adder of appropriate size to add the two numbers.

Wallace Multiplier Example

Consider a multiplier for 4X4 bits. Partial products are generated in parallel and we have the following wires:

Bit	Terms	Wires
0	a_0b_0	1
1	a_0b_1, a_1b_0	2
2	a_0b_2, a_1b_1, a_2b_0	3
3	$a_0b_3, a_1b_2, a_2b_1, a_3b_0$	4
4	a_1b_3, a_2b_2, a_3b_1	3
5	a_2b_3, a_3b_2	2
6	a_3b_3	1

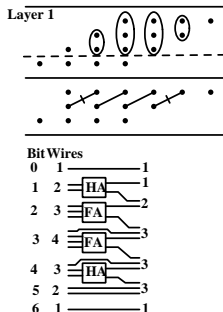
4X4 Wallace Multiplier: First Reduction

The multiplier has 4 rows of partial products, which are divided in groups of 3 and 1.

The bottom row is just passed on to the next stage.

For wires within the top 3 rows:

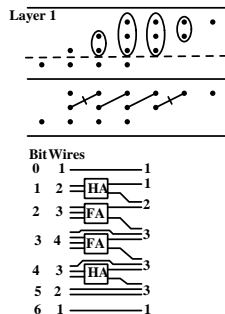
- Bit 0 has a single wire: passed through.
- Bit 1 has 2 wires: fed to a half adder.
- Bits 2 and 3 have 3 wires: fed to full adders.
- Bit 4 has 2 wires (in the group of 3): fed to a half adder.
- Bit 5 has 1 wire: passed through.



4X4 Wallace Multiplier: Outputs after First Reduction

After first reduction

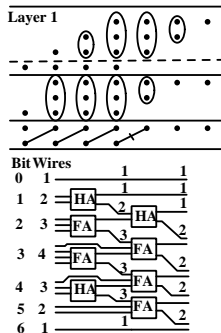
- Bits 0, 1 and 6 have a single wire.
- Bit 2 has 2 wires: carry of the half adder at bit 1 and the sum of full adder at bit 2.
- Bits 3 and 4 have 3 wires: carry of the full adder at lower weight, the sum wire from their full/half adder and a passed through wire.
- Bit 5 has 3 wires: carry of bit 4 plus 2 fed through wires.



4X4 Wallace Multiplier: Second Reduction

Since Bits 3, 4 and 5 have 3 wires each, we need another reduction layer. This will be the last reduction layer.

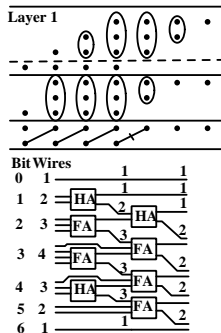
- Bits 0 and 1 have single wires. These are fed through.
- Bit 2 has 2 wires: these are fed to a half adder.
- Bits 3, 4 and 5 have 3 wires: These are fed to full adders.



4X4 Wallace Multiplier: Outputs from Second Reduction

Bits 0, 1, and 2 have single wires which carry the final result.

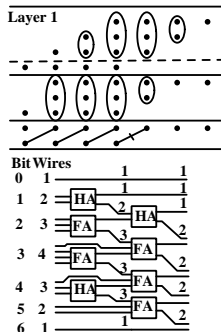
- Bit 3 has 2 wires: carry of the half adder at bit 2 and sum of the full adder at bit 3.
- Bit 4 has 2 wires: carry of the full adder at bit 3, and sum of the full adder at bit 4.
- Bit 5 has 2 wires, carry of bit 4 and sum of bit 5.
- Bit 6 has 2 wires, carry of bit 5 and 1 fed through wire.



4X4 Wallace Multiplier: Final Addition

After the second layer, no bit has more than 2 wires. Single wires at bits 0, 1 and 2 are fed through to the output.

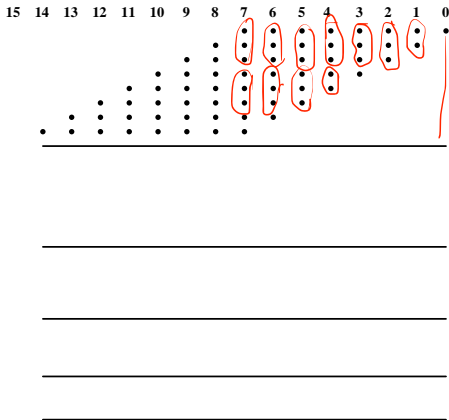
- A fast conventional adder is used to add the 2 bits each at Bits 3, 4, 5 and 6.
- Notice that we do not need a full width fast adder.
- This is because the half adders at low weights have already rippled the carry while the rest of weights were being reduced.
- This makes the final adder smaller and faster.



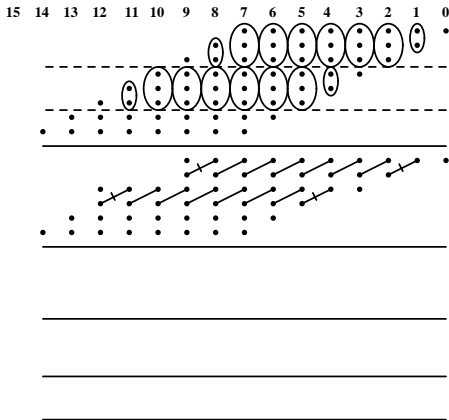
Redundant MSB in large Wallace Multipliers

- The reduction scheme as described above sometimes produces a redundant most significant bit.
- The result is still correct and the redundant bit will always be zero.
- To see this effect, let us apply the above scheme to an 8x8 multiplier.

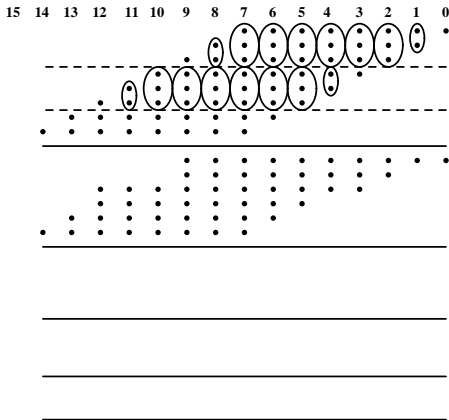
Redundant MSB in large Wallace Multipliers



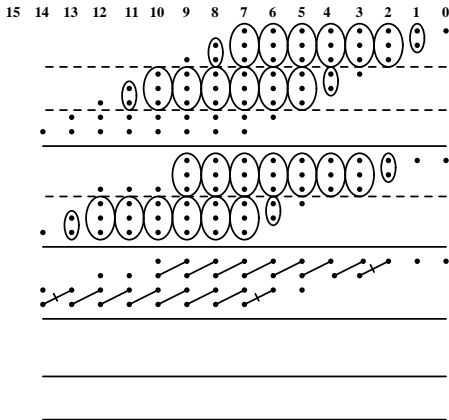
Redundant MSB in large Wallace Multipliers



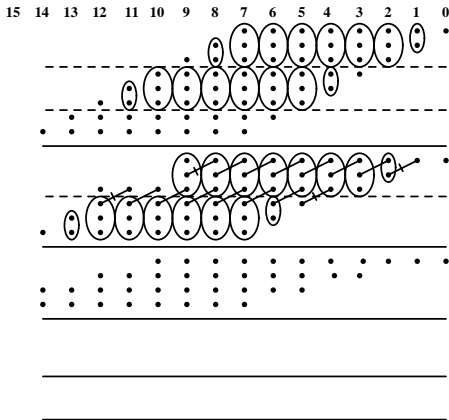
Redundant MSB in large Wallace Multipliers



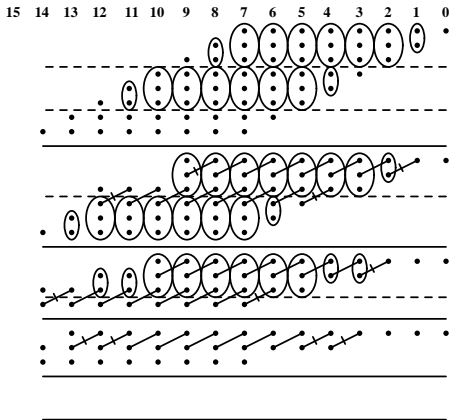
Redundant MSB in large Wallace Multipliers



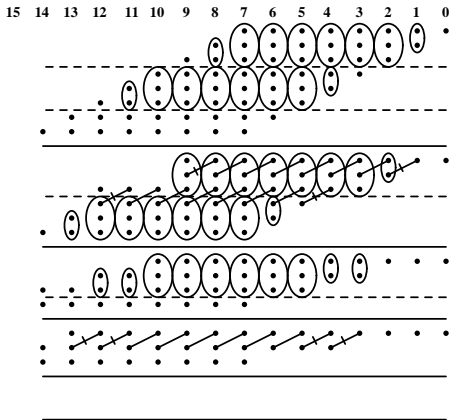
Redundant MSB in large Wallace Multipliers



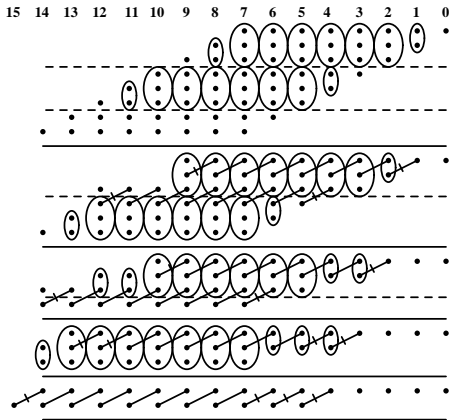
Redundant MSB in large Wallace Multipliers



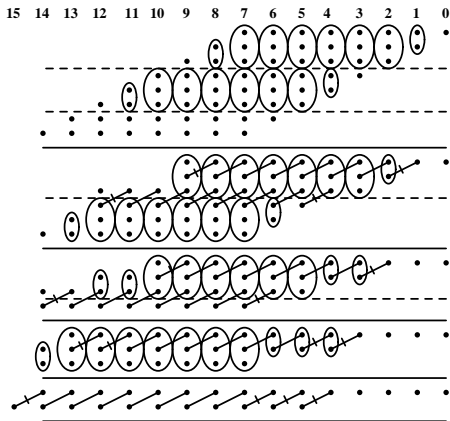
Redundant MSB in large Wallace Multipliers



Redundant MSB in large Wallace Multipliers



Redundant MSB in large Wallace Multipliers



As one can see, there are two bits at bit-14, which can produce a carry during the final addition.

When this carry is added to the bit at bit-15, it could produce another carry which will go to bit-16.

This would be an extra bit.

(Bits 0 to 16 will be 17 bits).

In practice, this does not occur, as multiplying 8 bit operands will generate at the most a 16 bit result.

Avoiding the Redundant MSB in Wallace Multipliers

One can avoid the redundant bit by modifying the reduction scheme.

- We treat all wires in a column as equivalent.
(No groups of 3 rows).
- Make bunches of 3 wires and send each to a full adder.
- Now we can be left with 0, 1 or 2 wires.
- There is nothing to do for 0 wires left.
- If one wire is left, it is passed through to next layer.
- If two wires are left, we have a more complex decision.
- We need to define the capacity of a reduction layer to describe the policy for reduction of 2 wires.

Wire capacity of a reduction layer

- We define the capacity of a layer as the maximum number of wires it can accommodate. How can we determine it?
- We know that the final reduction layer should have no more than 2 wires. Now we can work **backwards** from the final layer to the first.
- Let d_j represent the maximum number of wires for any weight in layer j , where $j = 1$ for the final adder. (Thus $d_1 = 2$).
- The maximum number of wires which can be handled in layer $j+1$ (from the end) is the integral part of $(3/2)d_j$.

Wire capacity of a reduction layer

- The maximum number of wires for any weight in layer $j+1$ (from the end) is the integral part of $(3/2)d_j$.
- $j = 1$ for the final adder. Thus $d_1 = 2$.
- We go up in j , till we reach a number which is just greater than or equal to the largest bunch of wires in any weight.
- The number of reduction layers required is this $j_{final} - 1$.
- Capacities of layers starting from last layer and moving towards the top are 2, 3, 4, 6, 9, 13, 19

Reduction of two wires

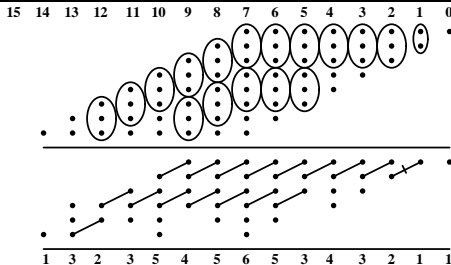
Now we can define the policy for reduction of 2 left over wires after deploying the maximum number of full adders.

- If all columns at the right have a single wire, we reduce the two wires using a half adder. (This helps in reducing the width of final adder).
- If there is a column to the right with more than one wire, we pass through the two wires to the next layer if it can accommodate these. (That is, the total number of wires do not exceed the capacity of that layer).
- If passing through the two wires would exceed the capacity of next layer, we reduce these with a half adder.

Wallace Reduction without redundant MSB

Max wires in this layer: 8, Capacity of next layer = 6.

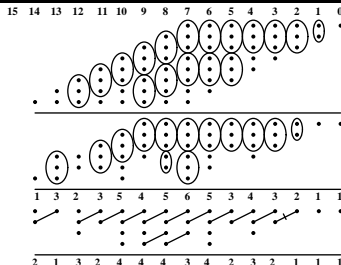
Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
FA	0	0	0	1	1	1	2	2	2	2	2	1	1	1	0	0
Remaining	0	1	2	0	1	2	0	1	2	1	0	2	1	0	2	1
HA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
PT	0	1	2	0	1	2	0	1	2	1	0	2	1	0	0	1
Sums	0	0	0	1	1	1	2	2	2	2	2	1	1	1	1	0
Carries to Higher bits	0	0	0	1	1	1	2	2	2	2	2	1	1	1	1	0
Output Wires	0	1	3	2	3	5	4	5	6	5	3	4	3	2	1	1



Wallace Reduction without redundant MSB

Second reduction layer: Capacity of next layer = 4.

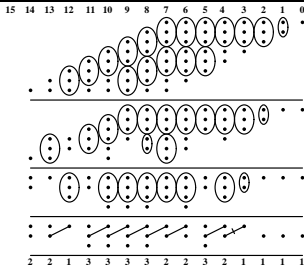
Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	1	3	2	3	5	4	5	6	5	3	4	3	2	1	1
FA	0	0	1	0	1	1	1	1	2	1	1	1	1	0	0	0
Remaining	0	1	0	2	0	2	1	2	0	2	0	1	0	2	1	1
HA	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
PT	0	1	0	2	0	2	1	0	0	2	0	1	0	0	1	1
Sums	0	0	1	0	1	1	1	2	2	1	1	1	1	1	0	0
Carries to Higher bits	0	0	1	0	1	1	1	2	2	1	1	1	1	1	0	0
Output Wires	0	2	1	3	2	4	4	4	3	4	2	3	2	1	1	1



Wallace Reduction without redundant MSB

Third reduction layer: Capacity of next layer = 3.

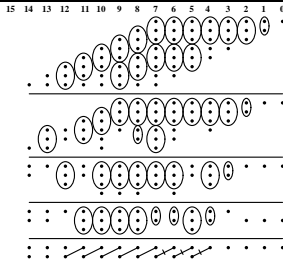
Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	2	1	3	2	4	4	4	3	4	2	3	2	1	1	1
FA	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0
Remaining	0	2	1	0	2	1	1	1	0	1	2	0	2	1	1	1
HA	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
PT	0	2	1	0	2	1	1	1	0	1	2	0	0	1	1	1
Sums	0	0	0	1	0	1	1	1	1	1	0	1	1	0	0	0
Carries to Higher bits	0	0	0	1	0	1	1	1	1	1	0	1	1	0	0	0
Output Wires	0	2	2	1	3	3	3	3	2	2	3	2	1	1	1	1



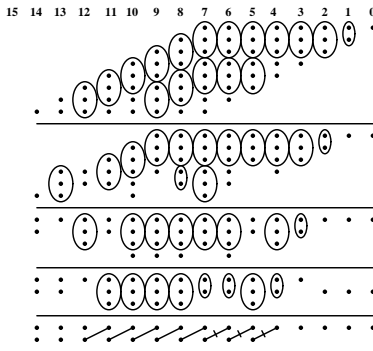
Wallace Reduction without redundant MSB

Final reduction layer: Capacity of next layer = 2.

Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	2	2	1	3	3	3	3	2	2	3	2	1	1	1	1
FA	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0
Remaining	0	2	2	1	0	0	0	0	2	2	0	2	1	1	1	1
HA	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0
PT	0	2	2	1	0	0	0	0	0	0	0	0	1	1	1	1
Sums	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
Carries to Higher bits	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
Output Wires	0	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1



Wallace Reduction without redundant MSB



We have reduced the number of wires at all bit positions to ≤ 2 without generating a bit at b15.

There are two wires at b14 and if these produce a carry, it will go to b15 and there is no redundant b16.

Dadda Multipliers

Dadda multipliers are very similar to Wallace multipliers and use the same 3 stages:

- 1 Generate all bits of the partial products in parallel.
- 2 Collect all partial products bits with the same place value in bunches of wires and reduce these in several layers of adders till each weight has no more than two wires.
- 3 For all bit positions which have two wires, take one wire at corresponding place values to form one number, and the other wire to form another number.

Add these two numbers using a fast adder of appropriate size.

The difference is in the reduction stage.

Dadda Multipliers

- Wallace multipliers reduce as soon as possible, while Dadda multipliers reduce as late as possible.
- Dadda multipliers plan on reducing the final number of wires for any weight to 2 with as few and as small adders as possible.
- We determine the number of layers required first, beginning from the **last** layer, where no more than 2 wires should be left.
- The number of layers in Dadda multipliers is the same as in Wallace multipliers.

Dadda Multipliers: Number of layers

- We work back from the final adder to earlier layers till we find that we can manage all wires generated by the partial product generator.
- We know that the final adder can take no more than 2 wires for each weight.
- Let d_j represent the maximum number of wires for any weight in layer j , where $j = 1$ for the final adder. (Thus $d_1 = 2$).
The maximum number of wires which can be handled in layer $j+1$ (from the end) is the integral part of $\frac{3}{2}d_j$.
- We go up in j , till we reach a number which is just greater than or equal to the largest bunch of wires in any weight.
- The number of reduction layers required is this $j_{final} - 1$.

Wire Reduction in Dadda Multipliers

- At each layer we know the maximum number of wires which should be left for the next layer.
- For each weight, we place the **least number of smallest** adders, such that the wires going out to the next layer do not exceed the maximum number of wires it can handle.
- At each weight, we must consider all the sum and pass through wires at this weight, as well as the wires which will be transferred through carry of the less significant weights, to the next layer.
- That is why we must begin with the lowest weight and go towards higher weights in each layer.

Dadda Multiplier: Example

Take the example of 4-bit by 4-bit multiplication multiplying $a_3a_2a_1a_0$ by $b_3b_2b_1b_0$. As before, partial products are generated in parallel and we have the following wires:

Weight	Terms	Wires
1	a_0b_0	1
2	a_0b_1, a_1b_0	2
4	a_0b_2, a_1b_1, a_2b_0	3
8	$a_0b_3, a_1b_2, a_2b_1, a_3b_0$	4
16	a_1b_3, a_2b_2, a_3b_1	3
32	a_2b_3, a_3b_2	2
64	a_3b_3	1

4x4 Dadda Multiplier: Number of Reduction Layers

- Maximum no. of wires for any weight in this example is 4.
- $d_1 = 2$, $d_2 = 3$, $d_3 = 4$. So we need 2 layers of reduction.
- The first reduction layer should reduce the number of wires at any weight to a maximum of 3.
- The second layer will then reduce these to a maximum of 2 wires.
- At each reduction layer, we scan from less significant weights to more significant ones, keeping track of additional carry wires which will be transferred *at the output* from lower weights to higher ones.

4x4 Dadda Multiplier: First Reduction Layer

Weight	Wires
1	1
2	2
4	3
8	4
16	3
32	2
64	1

- Weights 1, 2 and 4 have 3 or less wires. These are passed through.
- Weight 8 has 4 wires. No carry is anticipated from lower weights. A half adder is used to reduce the output wires to 3. (Half Adder Sum + 2 wires passed through).
- Weight 16 has 3 wires, but we anticipate a carry from the adder at weight 8. So we should reduce by 1 to keep the total **output** wires to 3. So this column is also reduced using a half adder.

Diagram illustrating the structure of Layer 1, showing a hierarchical arrangement of weights (Wt. 1, Wt. 2, Wt. 4, Wt. 8, Wt. 16, Wt. 32, Wt. 64) and their corresponding input and output nodes. The diagram shows a sequence of horizontal lines representing inputs and outputs, with boxes labeled 'HA' (Half Adder) indicating the combination of paths. The inputs and outputs are labeled with numbers 1, 2, 3, and 4, indicating the number of lines in each path.

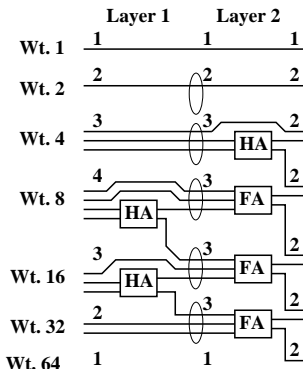
- A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

4X4 Dadda Multiplier: Second Reduction

In the second layer, we should leave no more than 2 wires at any weight, as this is the last stage.

- As before, we anticipate the number of carry wires transferred from the lower weight when planning reduction using half or full adders.
- In Dadda multipliers, we use minimum hardware during reduction. So the smallest adder which will reduce the output wires to 2 will be used.
- At the lowest weights, if the number of wires is less than or equal to 2, we just pass these through.
- So the single wire at Wt. 1, and the 2 wires at Wt. 2 are just fed through.

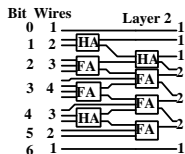
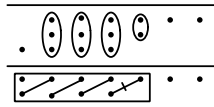
4X4 Dadda Multiplier: Second Reduction



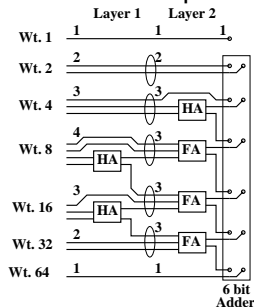
- 3 wires at Wt. 4 are reduced to 2 by a half adder: No carry in expected.
- Wt. 8 has 3 input wires. Carry will arrive from Wt. 4. Reduced using a Full adder.
- Wt. 16 has 3 input wires. Carry will arrive from Wt. 8. Reduced using a full adder.
- Wt. 32 has 3 input wires. Carry will arrive from Wt. 16. Reduced using a full adder.
- Wt. 64 has 1 input wire. Carry will arrive from Wt. 32, making it 2 output wires, which will be fed through.

4X4 Multipliers: Final Addition

Wallace Multiplier



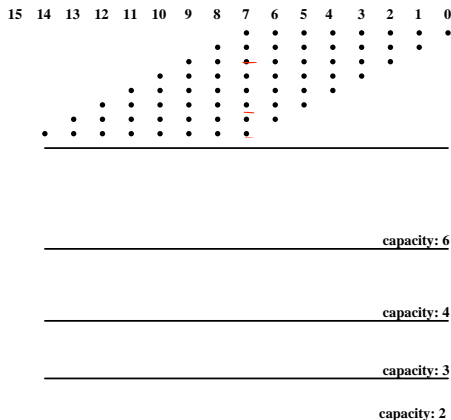
Dadda Multiplier



Notice that we have used only 3 Full Adders and 3 Half Adders during reduction, whereas Wallace multiplier requires 5 Full Adders and 3 Half Adders.

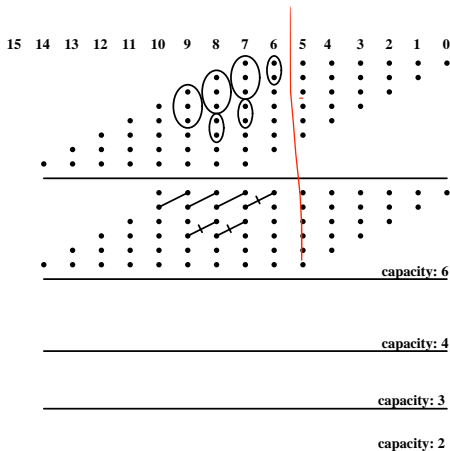
We require a 6-bit final adder for Dadda multiplier, whereas Wallace multiplier needs only a 4 bit final adder.

Dadda 8X8 Multiplier: Dot diagrams



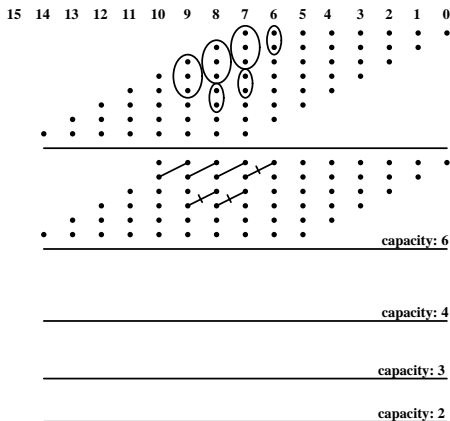
Dadda 8X8 Multiplier: First reduction

- Capacity of next layer is 6.
Since bits 0-5 have six or less wires, these are just passed through.
- bit 6 has 7 wires. To reduce to six, we place a half adder. (This gives a sum wire at bit 6 and a carry wire at bit 7).
- These outputs are shown by a dot each at bits 6 and 7, joined by a crossed line.
- Remaining 5 bits are passed through.



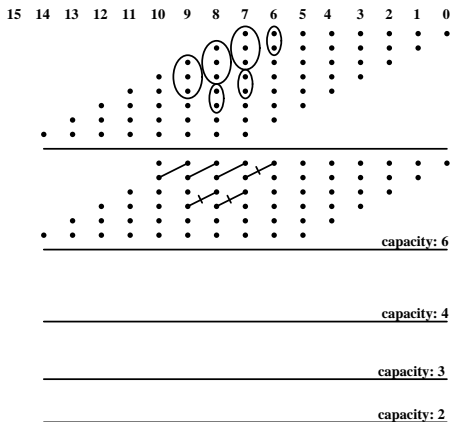
Dadda 8X8 Multiplier: First reduction

- Bit 7 has 8 wires. Of the six output places, one is already occupied by the carry of half adder at bit 6. So we should produce only 5 outputs at this bit – a reduction by 3.
- This can be done through a full and a half adder. Outputs of the full adder are shown as a dot at bit 7 (sum) and another at bit 8 (carry) joined by a line.
- Outputs of the half adder are also shown by two dots joined by a crossed line as before.



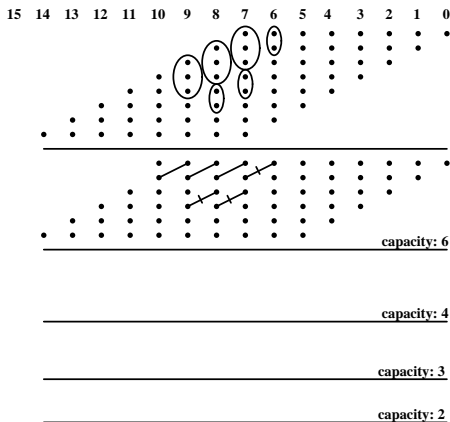
Dadda 8X8 Multiplier: First reduction

- Bit 8 has 7 wires. Of the 6 output places, 2 are occupied by carries of full and half adder.
- So we should produce only 4 outputs at this bit – again a reduction by 3.
- This can be done through a full and a half adder as before.
- Full and half adder take up 5 wires. The remaining 2 are passed through.



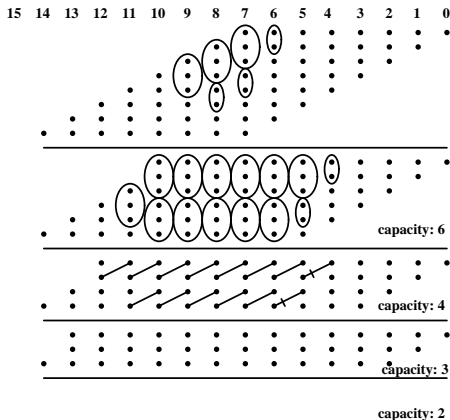
Dadda 8X8 Multiplier: First reduction

- Bit 9 has 6 wires, which should be reduced to 4 (since two places are taken up by carries of full and half adders).
- This can be done by a full adder whose outputs are shown by dots at bit 9 and 10 joined by a line.
- The remaining 3 wires are passed through.
- Wires of all the higher bits can be passed through without exceeding the limit of 6 outputs.



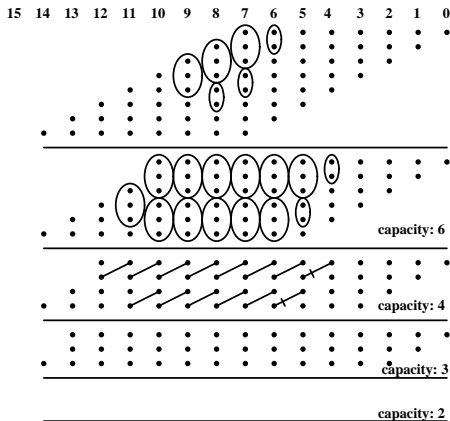
Dadda 8X8 Multiplier: Second reduction

- The output capacity of next layer is 4.
- Wires of bits 0-3 can just be passed through.
- For all bit position, we reduce the output places available by the incoming carries of previous bit.
- We place minimal number of full and half adders to reduce the total output wires to 4.
- Each full adder (FA) reduces wires by 2, half adder (HA) reduces by 1.



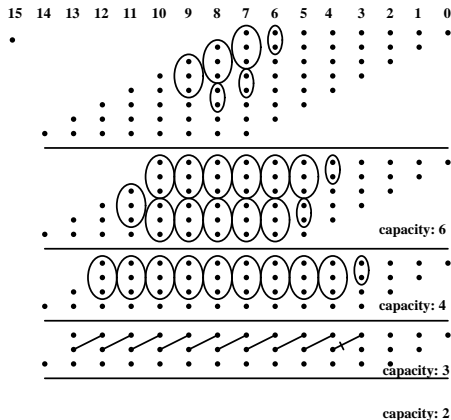
Dadda 8X8 Multiplier: Second reduction

- Bit 4 has 5 wires. Reduced to 4 by HA.
- Bit 5 has 6 wires, reduced to (4-1) by FA+HA.
- Bit 6 has 6 wires, reduced to (4-2) by 2 FAs.
- This is repeated till bit 10.
- Bit 11 has 4 wires, reduced to (4-2) by a FA.
- All other wires can be passed through.



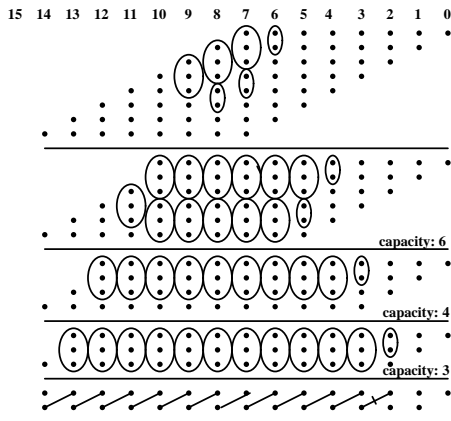
Dadda 8X8 Multiplier: Third reduction

- The reduction procedure can be repeated at each layer.
- If 2 wires (or multiples of 2) are to be reduced, we place FAs till 1 or 0 wires are left.
- If 1 wire remains, we place a Half adder.
- This layer requires a half adder at bit 4, FA + HA at bit 5, 2 FAs at bits 6-10 and a full adder at bit 11.
- Rest of the wires are just passed through.



Dadda 8X8 Multiplier: Final reduction

- Capacity of the final layer is 2.
- We continue with the same procedure, placing a half adder at bit 3 and full adders at bits 4-12.
- The remaining wires are passed through. Now we can make two words of 14 bit width and add these using a fast adder to get the final product.
- Notice there is no extra bit!



Comparison of Wallace and Dadda Multipliers

- Wallace and Dadda multipliers need the same number of layers.
- Dadda multiplier minimizes the number of adders, so it has the potential for lower complexity and power.
- Dadda multiplier uses more pass throughs and smaller adders which have lower delay. Thus it can also minimize the critical delay for reaching the final 2 wire stage.
- However, The final addition in Dadda multiplier needs wider carry propagating adders, which can slow it down.
- A careful evaluation inclusive of wiring and parasitic delays has to be made to determine which is the faster adder for a given configuration and process.

Serial Multipliers

Often, we need multipliers which have very low complexity or very low power consumption and speed is not very important.

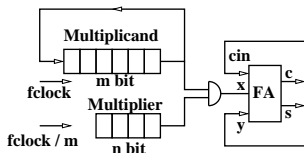
Serial multipliers are a good option in such cases.

Low complexity multipliers can be bit serial or row serial.

Bit serial multipliers require $m \times n$ clocks for completing an $m \times n$ multiplication.

Row serial multipliers require only n steps, but we require m full adders rather than just one.

Bit Serial Multipliers: Partial Product Generation

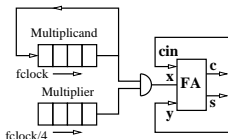


- Each bit of the multiplier needs to be ANDed with each bit of the multiplicand.
- This requires that all multiplicand bits be presented one after the other, every time a new bit from the multiplier is taken up.
- This can be managed by using a re-circulating shift register for the multiplicand, which is clocked at a rate which is m times faster than the clock to the multiplier shift register.
- The inputs y and C_{in} to the full adder have to be appropriately selected and timed to generate the correct product.

Bit Serial Multipliers: Partial Product Generation

Consider a 4×4 bit serial multiplier.

The x input to the Full Adder appears in the following order:



ck	x	ck	x	ck	x	ck	x
0	a0b0	4	a0b1	8	a0b2	12	a0b3
1	a1b0	5	a1b1	9	a1b2	13	a1b3
2	a2b0	6	a2b1	10	a2b2	14	a2b3
3	a3b0	7	a3b1	11	a3b2	15	a3b3

Bit Serial Multipliers: Partial Product addition

The arrival time of partial product bits is:


ck	x	ck	x	ck	x	ck	x
0	a0b0	4	a0b1	8	a0b2	12	a0b3
1	a1b0	5	a1b1	9	a1b2	13	a1b3
2	a2b0	6	a2b1	10	a2b2	14	a2b3
3	a3b0	7	a3b1	11	a3b2	15	a3b3

We need additions as follows:

		a3	a2	a1	a0
	×	b3	b2	b1	b0
<hr/>					
		a3b0	a2b0	a1b0	a0b0
	a3b1	a2b1	a1b1	a0b1	
a3b2	a2b2	a1b2	a0b2		
a3b3	a2b3	a1b3	a0b3		
<hr/>					

Bit Serial Multipliers: Partial Product addition

Let us put the arrival time of terms in parentheses next to each term.



			a3 b3	a2 b2	a1 b1	a0 b0
		×	a3b0(3)	a2b0(2)	a1b0(1)	a0b0(0)
			a3b1(7)	a2b1(6)	a1b1(5)	a0b1(4)
	a3b2(11)	a2b2(10)	a1b2(9)	a0b2(8)		
a3b3(15)	a2b3(14)	a1b3(13)	a0b3(12)			

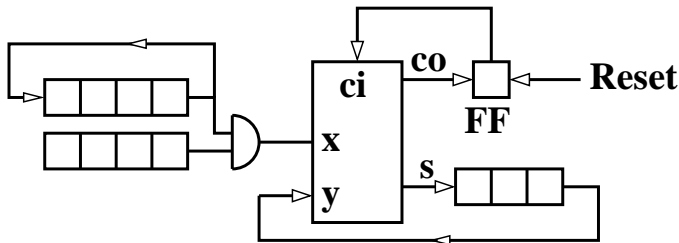
It is clear that for all additions, the earlier terms have to wait for 3 clock cycles before the later terms arrive.

We can manage this by putting a 3 bit shift register at the sum output and presenting the delayed output at the 'y' input of the full adder.

The carry output can be added immediately in the next clock, since it should go to the next column to its left.

Bit Serial Multipliers: Partial Product addition

A 3 clock delay for sum and a 1 clock delay for carry leads to the following circuit.

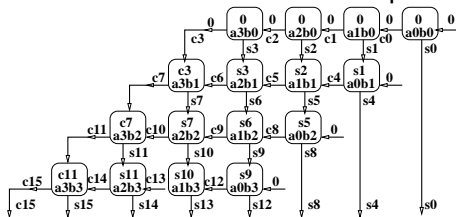


Unfortunately, it does not work!

We have to take care of a few exceptions at row ends.

Bit Serial Multiplier: Exceptions

Let us look at all the exceptions in detail.



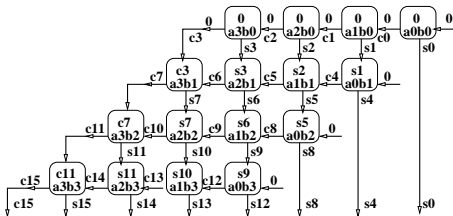
In the adjoining figure, sum and carry terms are indexed by the clock interval in which these were generated.

- At clocks 0, 4, 8 and 12, carry input should be forced to 0.
- At clocks 7, 11 and 15, the adder y input should receive carry terms (c3, c7 and c11) instead of sum terms (s4, s8 and s12).
- At these clocks, the sum terms should be taken out as result bits.

Bit Serial Multiplier: Exceptions

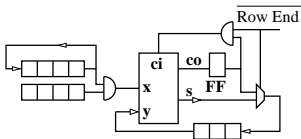
At clocks 0, 4, 8 and 12:

- Carry input should be forced to 0.
- The carry FF output (which is a 1 clock delayed version of c_{out}) should be inserted in the 3-bit shift register.
Thus C_3 (which is always 0), C_7 and C_{11} will emerge at clocks 7, 11 and 15 respectively.
- The sum terms should be taken out as result bits.



Bit Serial Multiplier: Implementation

With exception handling at the end of rows, the serial multiplier will work.



- Carry input is forced to 0 at row ends.
- The mux normally inserts the sum into the shift register. However, at row ends, it inserts the delayed carry output.
- The sum terms at row ends can be taken out as the low bits of the product.
- One can add another shift register at the output to collect these.
- The 2 more significant bits of the shift register and the last sum and carry provide the high bits of the product at the end.

Row Serial Multipliers

- We need not reduce the complexity all the way down to a single adder for serial multipliers.
- We could have a row of n adders performing additions in parallel.
- Taking the example of 4×4 multiplication, we are trying to perform the following operations:

$$\begin{array}{r}
 \text{a3 a2 a1 a0} \\
 \times \text{b3 b2 b1 b0} \\
 \hline
 \text{a3b0 a2b0 a1b0 a0b0} \\
 \text{a3b1 a2b1 a1b1 a0b1} \\
 \text{a3b2 a2b2 a1b2 a0b2} \\
 \text{a3b3 a2b3 a1b3 a0b3} \\
 \hline
 \end{array}$$

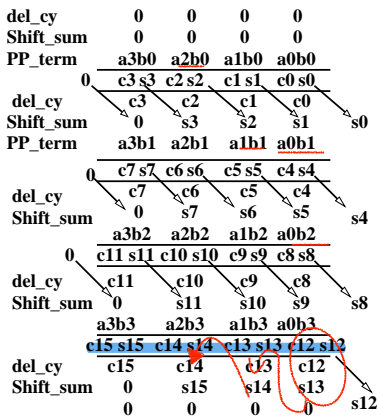
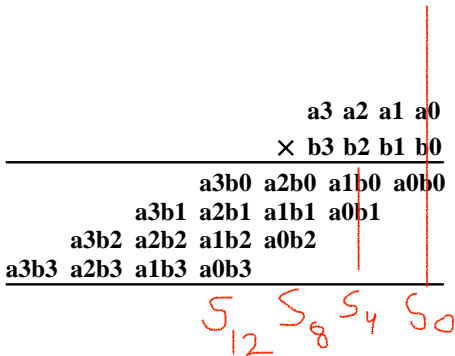
- We would like to perform 4 additions per clock interval.

Row Serial Multipliers

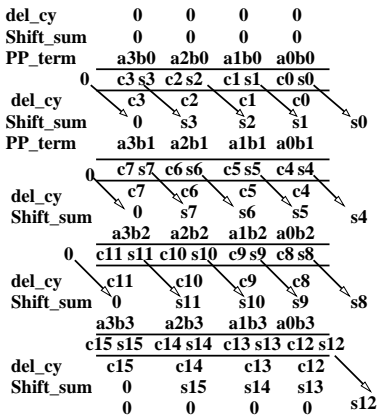
- We can use n full adders arranged in n columns.
- The carry of previous addition should remain at the same column.
- The sum from the previous addition *in the left column* is brought to this column by a shift operation.
- This sum has the same weight as the carry generated during the previous clock in this column.
- These two are added to the partial product bit for this column.

Row Serial Multipliers

The addition process using 4 adders is represented in the following figure.



Row Serial Multipliers



- Notice that the same 'a' term is used in a given adder.
- The 'b' term has to be shifted right every time to generate the right partial product bit.
- Sums have to be shifted right to be added to the carry of the previous addition in the same column.
- 4 additional clock cycles will be required to ripple the carry in the last addition. During these, the partial product bits will be 0.

Row Serial Multipliers

This scheme can be implemented as follows:

