Multipliers - Carry Save and Wallace Tree

Dr DC Hendry

November 2007

Outline I

Higher Performance Multipliers

- The basic multiplier structures seen so far are usable, and in their sequential formats were widely used.

Higher Performance Multipliers

- The basic multiplier structures seen so far are usable, and in their sequential formats were widely used.
- As available silicon area increased however, such structures were seen as occupying a small area, yet were slow compared to target clock speeds.

Higher Performance Multipliers

- The basic multiplier structures seen so far are usable, and in their sequential formats were widely used.
- As available silicon area increased however, such structures were seen as occupying a small area, yet were slow compared to target clock speeds.
- This lecture will look at a variety of techniques that increase the speed of a multiplier.

Higher Performance Multipliers

- The basic multiplier structures seen so far are usable, and in their sequential formats were widely used.
- As available silicon area increased however, such structures were seen as occupying a small area, yet were slow compared to target clock speeds.
- This lecture will look at a variety of techniques that increase the speed of a multiplier.
- Not all increase the area of the multiplier, rather just reflect clever thinking about the requirements and operation of the circuits.

Higher Performance Multipliers

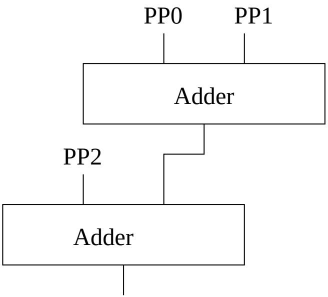
- The basic multiplier structures seen so far are usable, and in their sequential formats were widely used.
- As available silicon area increased however, such structures were seen as occupying a small area, yet were slow compared to target clock speeds.
- This lecture will look at a variety of techniques that increase the speed of a multiplier.
- Not all increase the area of the multiplier, rather just reflect clever thinking about the requirements and operation of the circuits.
- Remember that underlying technologies are always changing, the best way to build a circuit changes!

Carry Save Structures

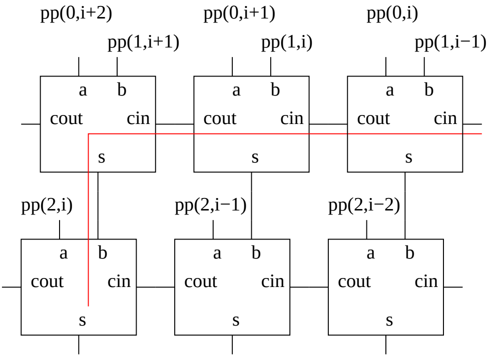
Consider the addition of the partial products PP_0, PP_1 and PP_2 (although these partial products are considered, the same argument applies to any three partial products).

Carry Save Structures

Consider the addition of the partial products PP_0, PP_1 and PP_2 (although these partial products are considered, the same argument applies to any three partial products).



Carry Save Structures



Carry Save Structures

- The final result generated by the array of adders is unchanged if we add the carry from adding PP_0 to PP_1 into the adder shown, or that adding in PP_2.

Carry Save Structures

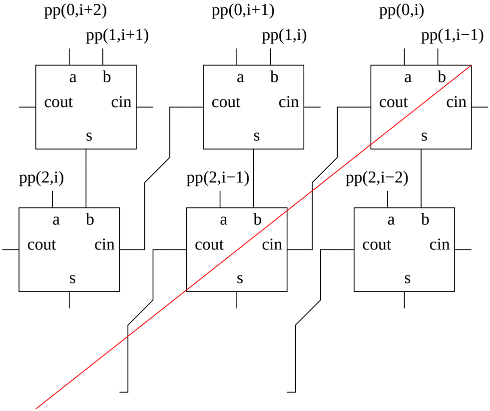
- The final result generated by the array of adders is unchanged if we add the carry from adding PP_0 to PP_1 into the adder shown, or that adding in PP_2.
- Each bit of the product is obtained by summing all partial product bits entering that *column* (in any order) plus the carry ins to that column.

Carry Save Structures

- The final result generated by the array of adders is unchanged if we add the carry from adding PP_0 to PP_1 into the adder shown, or that adding in PP_2.
- Each bit of the product is obtained by summing all partial product bits entering that *column* (in any order) plus the carry ins to that column.
- Note that for an array multiplier, the computation of the sum bit from an adder cell may also be on the critical path.

Carry Save Structures

- The final result generated by the array of adders is unchanged if we add the carry from adding PP_0 to PP_1 into the adder shown, or that adding in PP_2.
- Each bit of the product is obtained by summing all partial product bits entering that *column* (in any order) plus the carry ins to that column.
- Note that for an array multiplier, the computation of the sum bit from an adder cell may also be on the critical path.
- This leads us to the carry save architecture.


pp(0,i+2)
pp(1,i+1)
pp(0,i+1)
pp(1,i)
pp(0,i)
pp(1,i−1)
a
b
cout
cin
s
pp(2,i)
pp(2,i−1)
pp(2,i−2)

Carry Save Final Structure

- Note how the critical path in the design is now “diagonal”, as opposed to a “Manhattan” path, so reducing delay.

Carry Save Final Structure

- Note how the critical path in the design is now “diagonal”, as opposed to a “Manhattan” path, so reducing delay.
- The first row of adders may become “half adders”.

Carry Save Final Structure

- Note how the critical path in the design is now “diagonal”, as opposed to a “Manhattan” path, so reducing delay.
- The first row of adders may become “half adders”.
- At the final adder we do however have a row of carry signals still to be added into the product.

Carry Save Final Structure

- Note how the critical path in the design is now “diagonal”, as opposed to a “Manhattan” path, so reducing delay.
- The first row of adders may become “half adders”.
- At the final adder we do however have a row of carry signals still to be added into the product.
- This is done with a final high speed adder.

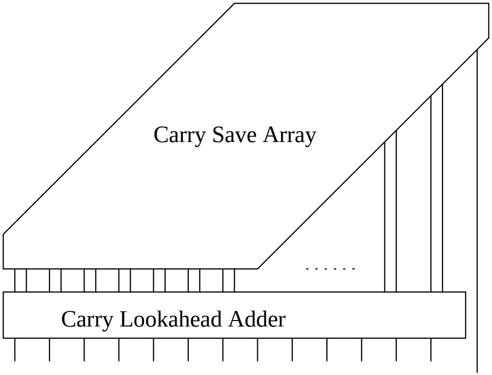
Carry Save Final Structure

- Note how the critical path in the design is now "diagonal", as opposed to a "Manhattan" path, so reducing delay.
- The first row of adders may become "half adders".
- At the final adder we do however have a row of carry signals still to be added into the product.
- This is done with a final high speed adder.
- Usually referred to in the literature as the CLA - Carry Lookahead Adder - but can be any fast adder.

Carry Save Final Structure

- Note how the critical path in the design is now "diagonal", as opposed to a "Manhattan" path, so reducing delay.
- The first row of adders may become "half adders".
- At the final adder we do however have a row of carry signals still to be added into the product.
- This is done with a final high speed adder.
- Usually referred to in the literature as the CLA - Carry Lookahead Adder - but can be any fast adder.
- Since signal arrival times are not uniform, a variation on CLA may be used - the synthesis tool may be used for this.

Carry Save Final Structure

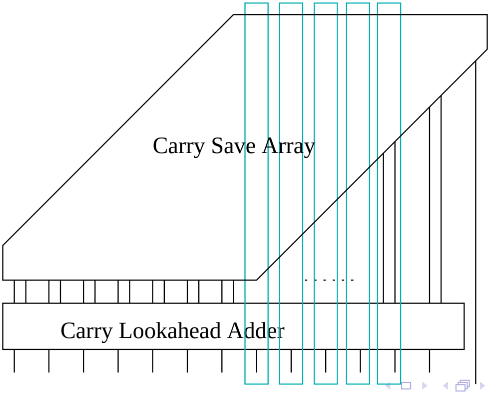


Wallace Trees

In a Wallace tree multiplier the addition of bits within one column is rearranged still further. Let's first re-visit the columns involved in the computation of a product.

Wallace Trees

In a Wallace tree multiplier the addition of bits within one column is rearranged still further. Let's first re-visit the columns involved in the computation of a product.



Wallace Trees

- Each column is characterised by the inputs to that column, and the outputs from that column.

Wallace Trees

- Each column is characterised by the inputs to that column, and the outputs from that column.
- The inputs to a column are the bits of the partial product (Booth or non-Booth encoded) plus ..

Wallace Trees

- Each column is characterised by the inputs to that column, and the outputs from that column.
- The inputs to a column are the bits of the partial product (Booth or non-Booth encoded) plus ..
- the carry bits from one column to the right plus ..

Wallace Trees

- Each column is characterised by the inputs to that column, and the outputs from that column.
- The inputs to a column are the bits of the partial product (Booth or non-Booth encoded) plus ..
- the carry bits from one column to the right plus ..
- the sum bits that are generated within the column.

Wallace Trees

- Each column is characterised by the inputs to that column, and the outputs from that column.
- The inputs to a column are the bits of the partial product (Booth or non-Booth encoded) plus ..
- the carry bits from one column to the right plus ..
- the sum bits that are generated within the column.
- The outputs from a column are the carry bits to the column one to the left plus ..

Wallace Trees

- Each column is characterised by the inputs to that column, and the outputs from that column.
- The inputs to a column are the bits of the partial product (Booth or non-Booth encoded) plus ..
- the carry bits from one column to the right plus ..
- the sum bits that are generated within the column.
- The outputs from a column are the carry bits to the column one to the left plus ..
- the last two sum bits in that column that are passed to the CLA.

Wallace Tree

- All bits from partial products are available at the same time.

Wallace Tree

- All bits from partial products are available at the same time.
- So in the Wallace tree multiplier we use a *tree* of adder cells.

Wallace Tree

- All bits from partial products are available at the same time.
- So in the Wallace tree multiplier we use a *tree* of adder cells.
- The carry in comes CSA fashion from the previous column.

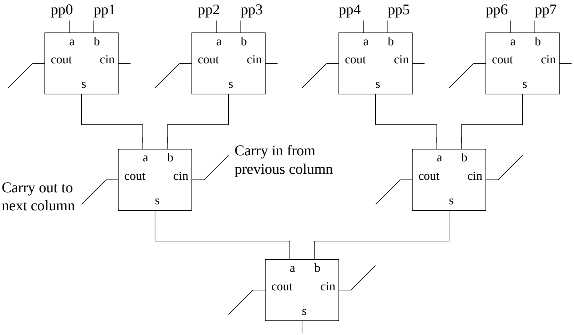
Wallace Tree

- All bits from partial products are available at the same time.
- So in the Wallace tree multiplier we use a *tree* of adder cells.
- The carry in comes CSA fashion from the previous column.
- The carry out goes CSA fashion to the next column.

Wallace Tree

- All bits from partial products are available at the same time.
- So in the Wallace tree multiplier we use a *tree* of adder cells.
- The carry in comes CSA fashion from the previous column.
- The carry out goes CSA fashion to the next column.
- In comparison to the basic array multiplier, the delay from partial products to final sum bits in a column is $O(ln(n))$ rather than $O(n)$.

Wallace Tree



Delay Trees

- In a delay tree structure, an algorithm is used to determine the ordering of connections to adder cells.

Delay Trees

- In a delay tree structure, an algorithm is used to determine the ordering of connections to adder cells.
- A list based data structure is constructed in software for every column (that is every bit) of the product.

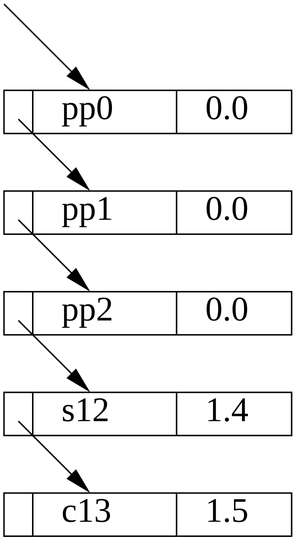
Delay Trees

- In a delay tree structure, an algorithm is used to determine the ordering of connections to adder cells.
- A list based data structure is constructed in software for every column (that is every bit) of the product.
- Each item on this list is a signal identifier (name) and an *arrival* time.

Delay Trees

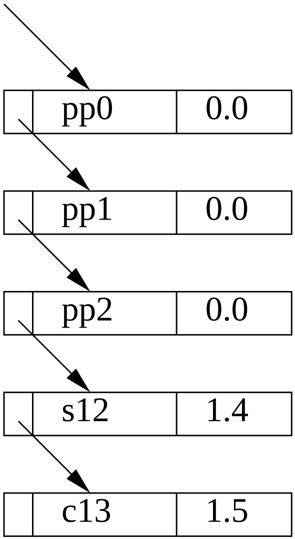
- In a delay tree structure, an algorithm is used to determine the ordering of connections to adder cells.
- A list based data structure is constructed in software for every column (that is every bit) of the product.
- Each item on this list is a signal identifier (name) and an *arrival* time.
- The arrival time is when that signal is valid assuming new input data is applied to the circuit at time zero.

Delay Trees



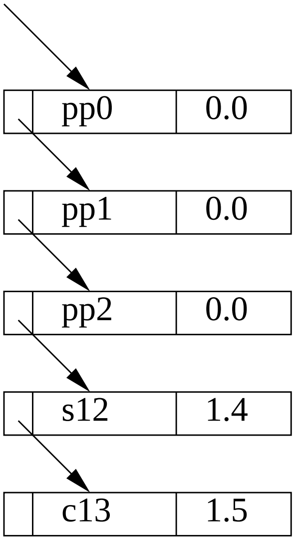
- One such list is constructed and processed for each column starting from the lsb.

Delay Trees



- One such list is constructed and processed for each column starting from the lsb.
- Each list is initialised with the list of partial product bits entering that column - with zero arrival time.

Delay Trees



- One such list is constructed and processed for each column starting from the lsb.
- Each list is initialised with the list of partial product bits entering that column - with zero arrival time.
- The list is always ordered on arrival time.

Delay Trees

- For each list now proceed as follows:

Delay Trees

- For each list now proceed as follows:
- Take the first three signals off of the list and wire these signals as the inputs to a full adder with outputs s and c.

Delay Trees

- For each list now proceed as follows:
- Take the first three signals off of the list and wire these signals as the inputs to a full adder with outputs s and c.
- Add s to the current list with arrival computed from the arrival times of the three inputs and the delay through the adder.

Delay Trees

- For each list now proceed as follows:
- Take the first three signals off of the list and wire these signals as the inputs to a full adder with outputs s and c.
- Add s to the current list with arrival computed from the arrival times of the three inputs and the delay through the adder.
- Add c to the list for the adjacent more significant column.

Delay Trees

- For each list now proceed as follows:
- Take the first three signals off of the list and wire these signals as the inputs to a full adder with outputs s and c.
- Add s to the current list with arrival computed from the arrival times of the three inputs and the delay through the adder.
- Add c to the list for the adjacent more significant column.
- Repeat this until only two or three signals left in the list.

Delay Trees

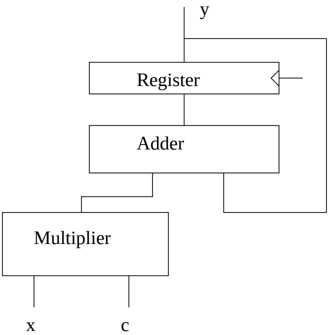
- For each list now proceed as follows:
- Take the first three signals off of the list and wire these signals as the inputs to a full adder with outputs s and c.
- Add s to the current list with arrival computed from the arrival times of the three inputs and the delay through the adder.
- Add c to the list for the adjacent more significant column.
- Repeat this until only two or three signals left in the list.
- When two pass to CLA, when three use half adder and pass to CLA.

Fused Structures/MAC

One of the most common operations in DSP is the accumulation of a number of products, that is, computation of $y = y + x * c$ where x and c are inputs.

Fused Structures/MAC

One of the most common operations in DSP is the accumulation of a number of products, that is, computation of $y = y + x * c$ where x and c are inputs. We could use the following structure:



Fused Structures/MAC

- The longest delay path in this circuit starts at x/c, works through the multiplier, then the adder and into the register.

Fused Structures/MAC

- The longest delay path in this circuit starts at x/c, works through the multiplier, then the adder and into the register.
- The total delay is actually less than the delay of the multiplier plus the delay of the adder.

Fused Structures/MAC

- The longest delay path in this circuit starts at x/c, works through the multiplier, then the adder and into the register.
- The total delay is actually less than the delay of the multiplier plus the delay of the adder.
- The lsb of the multiplier will be available early, so the adder can start propagating valid data before the msb of the multiplier product is available.

Fused Structures/MAC

- The longest delay path in this circuit starts at x/c, works through the multiplier, then the adder and into the register.
- The total delay is actually less than the delay of the multiplier plus the delay of the adder.
- The lsb of the multiplier will be available early, so the adder can start propagating valid data before the msb of the multiplier product is available.
- But we can still do better.

Fused Structures

- The signals for y are available early in the clock cycle, at the same time as x and c.

Fused Structures

- The signals for y are available early in the clock cycle, at the same time as x and c.
- Remembering that the multiplier delay tree simply adds together a number of partial products, we just add the signals for y to those to be added together.

Fused Structures

- The signals for y are available early in the clock cycle, at the same time as x and c.
- Remembering that the multiplier delay tree simply adds together a number of partial products, we just add the signals for y to those to be added together.
- The overall delay of the resulting design is only slightly longer than that of the multiplier unit itself.

Fused Structures

- The signals for y are available early in the clock cycle, at the same time as x and c.
- Remembering that the multiplier delay tree simply adds together a number of partial products, we just add the signals for y to those to be added together.
- The overall delay of the resulting design is only slightly longer than that of the multiplier unit itself.
- The same idea can be extended to equations such as $y = a * x + b * z$ and so on.

Fused Structures

- The signals for y are available early in the clock cycle, at the same time as x and c.
- Remembering that the multiplier delay tree simply adds together a number of partial products, we just add the signals for y to those to be added together.
- The overall delay of the resulting design is only slightly longer than that of the multiplier unit itself.
- The same idea can be extended to equations such as $y = a * x + b * z$ and so on.
- These are referred to as *fused* multipliers.