EE 486 lecture 6: Integer Addition

M. J. Flynn

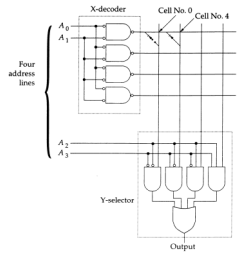


Figure 2.3: ROM model.

Addition

- The add function is fundamental to determining processor cycle time and hence overall performance.
- Add algorithms have been widely studied, so that there are many apparently different algorithms that actually differ only in some minute detail.

Add algorithms

- ripple add: the baseline,
- carry skip adders
- carry selection adds
- carry propagating adds
- Ling adds
- carry saving add
- hybrid add

Ripple adders

- the baseline: 2n gate delays
- Manchester carry chain
- carry completion

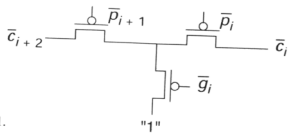
Bit logic

- (in the following V means exclusive OR)
- Sum: $s_i = a_i \text{ V } b_i \text{ V } c_i$
- Carry out: $c_{i+1} = a_i b_i + b_i c_i + a_i c_i$
- Propagate: $p_i = a_i + b_i$
- Generate: $g_i = a_i b_i$
- Carry out is also: $c_{i+1} = g_i + p_i c_i$

Manchester adders

- Use a single pass transistor to implement the p_i and another to implement g_i
- Fast serial carry chain up to a circuit limit, less than 8, usually much less. As chain get longer the capacity seen by the driver increases quadratically.

Simplified Manchester carry bit



Chain for C_{out} = 1.

Carry completion

- Asynchronous adder: **E(l**) = $\log_2$ n
- Create two carry out signals, c_i =1and c_i =0
- then c_{i+1} = $g_i + p_i c_i$ and if d_{i+1} is the complement of c_{i+1}
- then we must have ($c_{i+1} + d_{i+1}$) = 1 for each of the i bits to detect completion (ANDed)

Carry skip adders

- single level, multiple levels
- fixed block size, variable block size
- the multi level, fixed block is a type of carry look ahead
- Babbage's "carriage anticipation"is a carry skip adder

Carry Skip logic

-

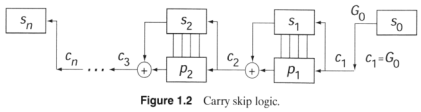


Figure 1.2 Carry skip logic.

Carry skip

- Bypass logic for carries based on bit propagate terms
- So Pi = ($p_3 p_2 p_1$) c_i has group Pi up to a fan in limit, r (r=4, above)
- In 2 gate delays the carry span a group.
- Can create a group of groups for faster add.

Carry skip

- Spans r-1 bits with r limited AND; $C_2=G_1+(p_3p_2p_1)\,C_1$ needs 2 gate delays
- If the carry is rippled in the first and last block the (single level) skip delay is the ripple delay + the skip delay
- $t = 4(r-1) +2([n/(r-1)] –2)$

Carry select

- conditional sum
- carry select (Bedrij)
- the carry select type adder is widely used as a component in a hybrid adder.

Conditional sum

- Partition the n bit operands into n/r bit digits
- For each r-limited digit, form s_E and s_N, c_E and c_N based on c_{in} =1 for s_E, c_E and c_{in} =0 for s_N, c_N
- Partition into digit pairs and determine the $s_E\,s_E$, c_E and the $s_N\,s_N$, c_N result.
- Continue now for 4 digit groups, etc

Conditional sum

- Logic within a 4 bit digit
  - $S_{N0}=A_0\ V\ B_0$; $S_{E0} = \sim S_{N0}$
  - $S_{N1}= A_1V\ B_1VG_0$; $S_{E1} = A_1V\ B_1VP_0$
  - $S_{N2}= A_2V\ B_2V(G_1+P_1G_0)$; $S_{E2}=A_2VB_2V(G_1+P_1P_0)$
  - $S_{N3}= A_3V\ B_3V(G_2+ +P_2G_1 +P_2P_1G_0)$;
  - $S_{E3}= A_3VB_3V(G_2+ +P_2G_1 +P_2P_1P_0)$;
  - $C_{N4} = G_3+ P_3G_2 +P_3P_2G_1+P_3P_2P_1G_0$
  - $C_{E4} = G_3+ P_3G_2 +P_3P_2G_1+P_3P_2P_1P_0$

Conditional sum

- For digit pairs the carry outs are
  - $C_{E2} = G_1+P_1P_0$
  - $C_{N2} = G_1+P_1G_0$
- The sum pairs, E and N are selected so that the lsd's are unaffected and the the msd's are unaffected if their carry ins differ, but if both carry in = 0 replace S_E with S_N and if both carry in = 1 replace S_N with S_E

Carry select

- Rather than selecting by pairs we can select up to a fan in limit (r-1)
- $C_4=C_{N4} +C_{E4}C_0$
- $C_8=C_{N8} +C_{E8}C_4$
- $C_{12}=C_{N12} +C_{E12}C_8 = C_{N12} +C_{E12}C_{N8} + C_{E12}C_{E8}\,C_{N4} + C_{E12}C_{E8}\,C_{E4}C_0$

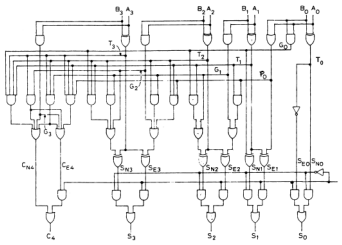


Figure 3.2: 4-bit conditional sum adder slice with carry-look-ahead (gate count= 45).

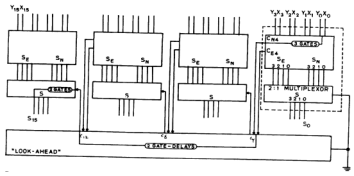


Figure 3.3: 16-bit conditional sum adder. The dotted line encloses a 4-bit slice with internal look ahead. The rectangular box (on the bottom) accepts conditional carries and generates fast true carries between slices. The worst case path delay is seven gates.

Carry select delay

- Delay consists of digit addition, carry propagation and final sum selection
- Selection is a MUX:$S_4 = S_{E4}C_0 + S_{N4}\overline{C_0}$
- Delay is $= k + 2[\log_{r-1}([n/r]-1)]$

Carry propagating adders

- The most widely studied class of adder:
  - carry look ahead
  - canonic
  - prefix adders

CLA: carry look ahead adders

- Form p, g terms for each bit, i
- Then $C_1 = G_0 + P_0C_0$; $C_2 = G_1 + P_1C_1$
- $C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$
- $C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$
- Group generate:$G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$
- Group propagate: $P_3P_2P_1P_0$

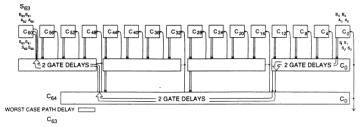
Group generate(G') and Group propagate(P')

- $C_4 = G'_0 + P'_0C_0$
- $C_8 = G'_1 + P'_1G'_0 + P'_1P'_0C_0$
- $C_{12} = G'_2 + P'_2G'_1 + P'_2P'_1G'_0 + P'_2P'_1P'_0C_0$
- $G''_0 = G'_3 + P'_3G'_2 + P'_3P'_2G'_1 + P'_3P'_2P'_1G_0$
- $P''_0 = P'_3P'_2P'_1P'_0$
- $C_{16} = G''_0 + P''_0C_0$
- $C_{64} = G''_3 + P''_3G''_2 + P''_3P''_2G''_1 + P''_3P''_2P''_1G''_0$ + propagate and end around

CLA delay

- C_{64} is not the worst case delay path, that's S_{63}. C_{48} is formed at the same time as C_{64} but must form C_{60} which then forms C_{63}, then S_{63}
- $C_{60} = G'' + P''C_{48}$
- $C_{63} = G' + P' C_{60}$

CLA delay



CLA delay

- Requires $[\log_r n]$ look ahead levels to form C_{64} and C_{48}, but $[\log_r n] - 1$ levels to form C_{63} .Thus it takes $2([\log_r n]) - 1$ levels at 2 gates per level. Plus another gate to form p, g and a final gate to select S_{63}.
- CLA delay = $4\ [\log_r n]$

Canonic and prefix adders

- Prefix is canonic with r = 2
- Canonic uses an irregular implementation so that each higher order sum bit has its carry in in the same time as the high order carry out.

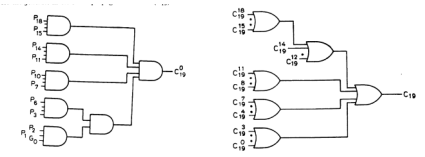
Fast counters

- Suppose we have xxxxx + 1= sssss. To find s_n and c_{out} we need $x_n V c_{in}$ and $x_n c_{in}$, but c_{in} is just the AND of all x terms and the input "1".
- So delay is $[\log_r n] + 1$ (the AND gates plus the final V)

Canonic adders

- The c_{in} is now an AND tree, to generate a carry from each of the lower order bits and the OR tree which ORs them together. Both have n inputs.
- So delay = $2[\log_r n] + 2$, the 2 trees plus a delay to form the sum (V) and a delay to form p,g.
- Need a "separate" tree pair for each sum bit, but many terms can be shared (prefixed).

AND and OR trees



Ling adder

- Redefines carry as pseudo carry to eliminate delay in forming p and g.
- Consider a 6 bit adder with r = 3 and t=aVb
- $s_6=t_6Vc_6$; $c_6=G_1+P_1G_0$
- $G_1 =g_5+p_5g_4+p_5p_4g_3$
  $G_0 =g_2+p_2g_1+p_2p_1g_0$
- $P_1= p_5p_4p_3$

Ling adders

- Since $g_5 = p_5g_5$, all terms in G_1 can be written to have the common factor p_5
- This defines the pseudo carry, H
- So $s_6=t_6V(p_5h_6)$; $h_6=H_1+P_1H_0$
- $H_1 =g_5+g_4+p_4g_3$
  $H_0 =g_2+g_1+p_1g_0$
- here $P_1= p_4p_3p_2$

Ling adders

- We can eliminate the p,g delay, compare
- $G_0= a_2b_2+ a_1a_2b_1 + b_2a_1b_1 + a_2a_1a_0b_0 + a_2b_1a_0b_0 + b_2a_1a_0b_0 + b_2b_1a_0b_0$
- $H_0= a_2b_2+ a_1b_1 + a_1a_0b_0 + b_1a_0b_0$
- We can also generate CLA type H',P' and higher level terms, but the final sum and carry must include p_nh_n

Hybrid adders

- Most modern adders do not follow a pure strategy but use a combination of techniques. Perhaps Ling or a Manchester chain to start, a look ahead then a carry select for the last selection.
- Results are not clearly defined; adders with minimum gate delays usually are larger and the layout (wires) determine the best design