University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS150 J. Wawrzynek
Spring 2000 E. Caspi

Homework #9 – Solution

5.13 A *hierarchical* carry lookahead adder (CLA) uses several key computations. With 4-bit CLAs, these computations are:

(a)	Given A_{0-3}, B_{0-3}	– P_{0-3}, G_{0-3}	take 1 gate delay	(Katz. Fig. 5.11)
(b)	Given C_0, P_{0-3}, G_{0-3}	– C_{1-4}	takes 2 gate delays	(Katz. Fig. 5.12)
(c)	Given C_0, P_{0-3}, G_{0-3}	– S_{0-3}	takes 3 gate delays	(Katz. Fig. 5.12)
(d)	Given P_{0-3}	– $P^{(1)}_0$	takes 1 gate delay	(Katz p. 254)
(e)	Given P_{0-3}, G_{0-3}	– $G^{(1)}_0$	takes 2 gate delay	(Katz p. 254)

In Katz Figure 14, each of the top "Adders" implements (a) (b) (c), whereas the bottom "Lookahead Carry Unit" implements (b), (d), and (e). The timing for (b) is a bit more subtle than shown above, since C_1 takes 2 gate delays past C_0 and P_0 but only 1 gate delay past G_0.

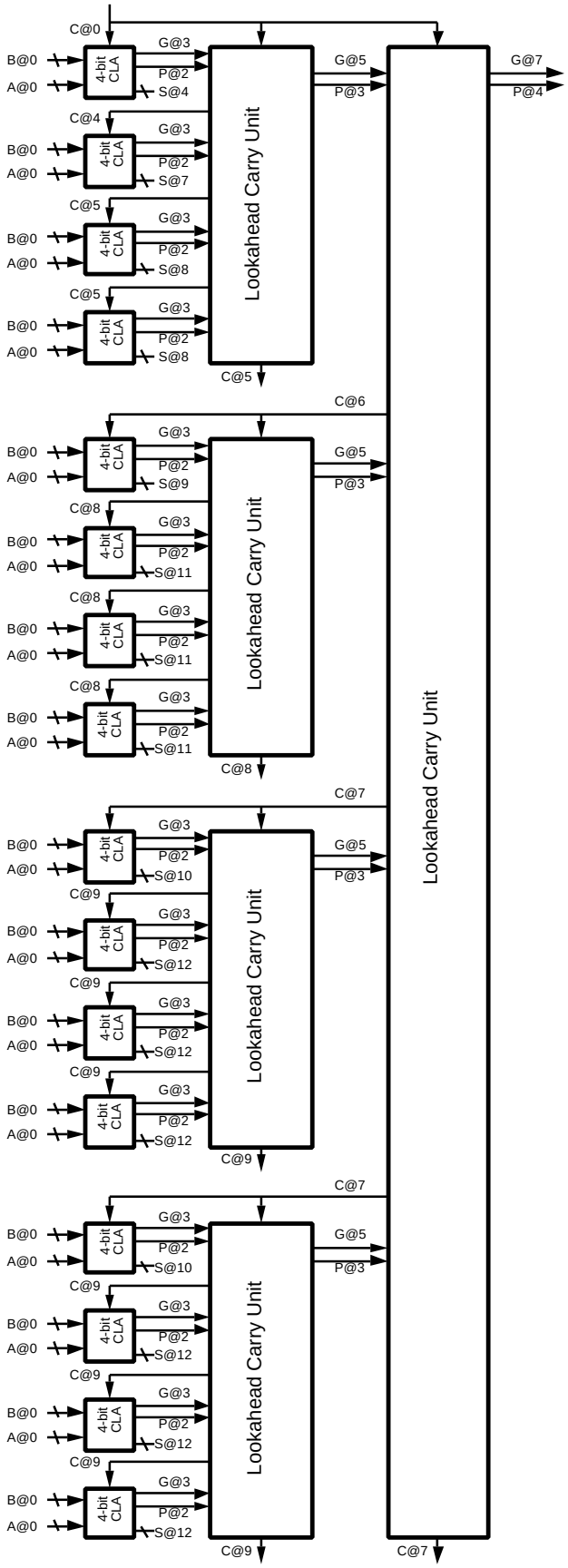
Computations (d) and (e) create the group propagate and generate signals. We denote the first-level group signals with a superscript: $^{(1)}$. In a CLA hierarchy, second-level group signals ($P^{(2)}_0$, $G^{(2)}_0$) are computed from the first-level ones ($P^{(1)}_{0-3}$, $G^{(1)}_{0-3}$) using another "Lookahead Carry Unit." Thus we add second-level computations:

(f)	Given $C^{(1)}_0$, $P^{(1)}_{0-3}$, $G^{(1)}_{0-3}$	– $C^{(2)}_{1-4}$	takes 2 gate delays
(g)	Given $P^{(1)}_{0-3}$	– $P^{(2)}_0$	takes 1 gate delays
(h)	Given $P^{(1)}_{0-3}$, $G^{(1)}_{0-3}$	– $G^{(2)}_0$	takes 2 gate delays

And similarly, third-level computations:

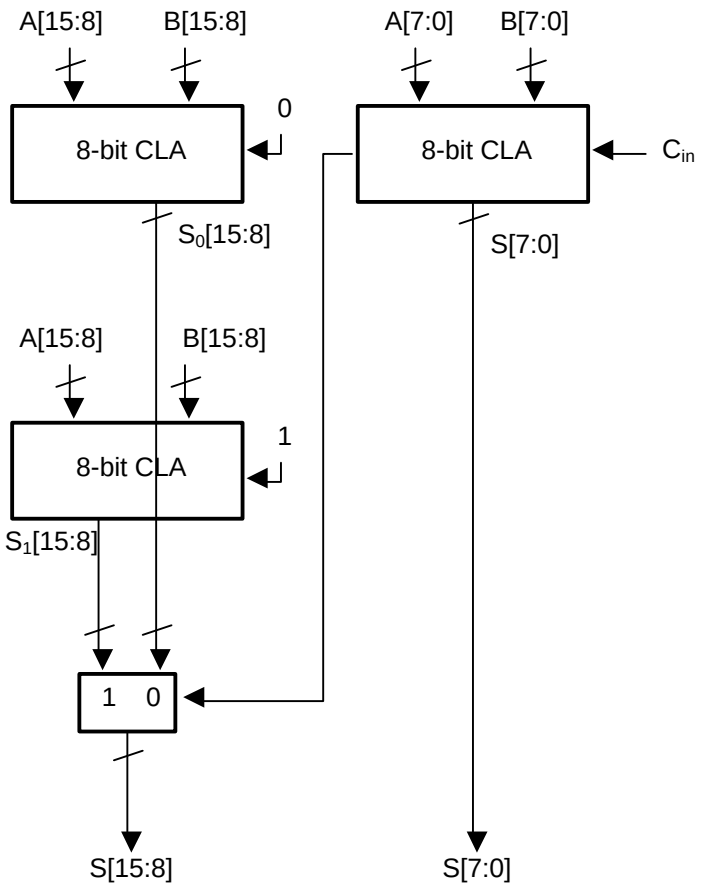
(i)	Given $C^{(2)}_0$, $P^{(2)}_{0-3}$, $G^{(2)}_{0-3}$	– $C^{(3)}_{1-4}$	takes 2 gate delays
(j)	Given $P^{(2)}_{0-3}$	– $P^{(3)}_0$	takes 1 gate delays
(k)	Given $P^{(2)}_{0-3}$, $G^{(2)}_{0-3}$	– $G^{(3)}_0$	takes 2 gate delays

A three-level hierarchy of 4-bit CLAs makes a 64-bit adder (4^3=64). The following page shows a block diagram for such an adder. The arrival time *t* of each signal is denoted by "@*t*". Note that we do not actually need the results of (h), but they are present for symmetry.


C@0
B@0
A@0
4-bit CLA
G@3
P@2
S@4
C@4
S@7
C@5
S@8
Lookahead Carry Unit
G@5
P@3
C@5
C@6
S@9
C@8
S@11
C@8
C@7
S@10
C@9
S@12
C@9
C@7
S@10
C@9
S@12
C@9
Lookahead Carry Unit
G@7
P@4
C@7

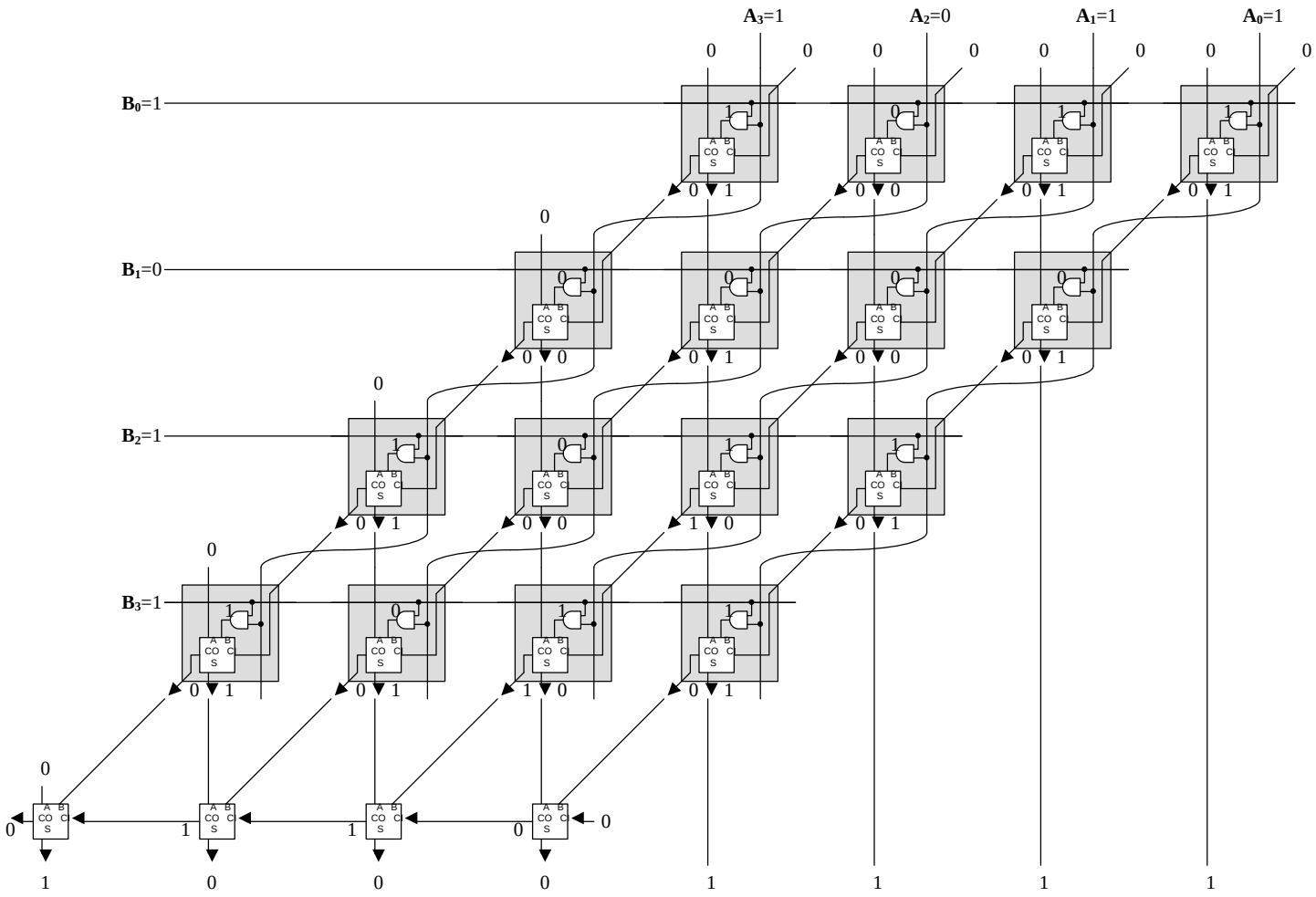
5.14 A 16-bit carry-select adder using three 8-bit CLAs is shown below. According to Katz' timing analysis for CLAs, an 8-bit CLA emits Carry and Sum after 3 and 4 gate delays, respectively[1]. The critical path of this carry-select structure is 6 gate delays, namely 4 for the left CLAs' sums plus 2 for the multiplexers. Critical path comparison:

- 16-bit carry-select adder: 6 gate delays
- 16-bit hierarchical CLA (Katz Figure 5.14): 8 gate delays
- 16-bit ripple-carry adder (Katz p.256): 32 gate delays



[1] Katz' analysis ignores the increased delay of high fan-in gates; a more realistic estimate is 7-10 gate delays, since certain gates in Katz' Figure 5.12 would need 9 inputs and a tree implementation.

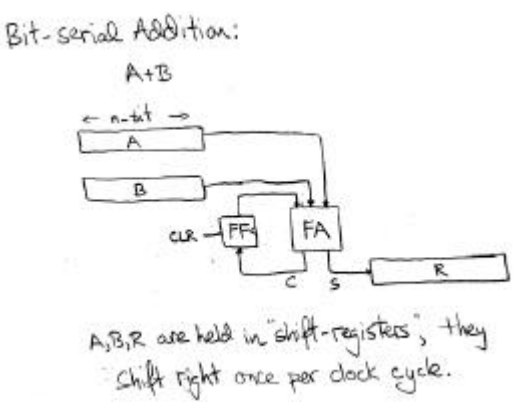
5.25 The array multiplier of Katz Figure 5.29 uses carry-save addition between rows but omits any final ripple-carry stage. In the following calculation, we include such a stage, since the multiplier cannot function without it. We trace the multiplication 11×13=143, or in binary: $(A=1011_2)\times(B=1101_2)=(10001111_2)$.

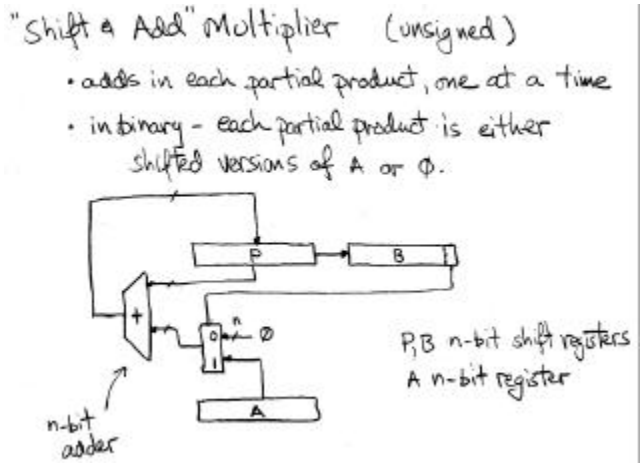


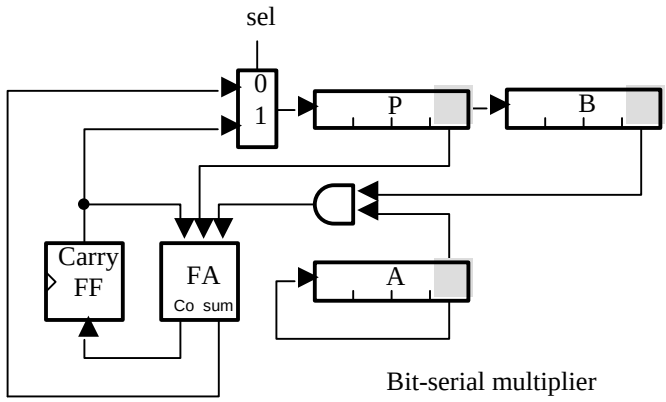
5.27 A full adder (FA) has 2 gate delays for Sum and Carry output. The longest path in Katz Figure 5.28 is through 6 FAs, e.g. on the right/bottom periphery. Together with the partial-product AND gates, this makes a critical path of <u>13 gate delays</u>.

Bit-Serial Multiplier

An $n \times n$ bit-serial multiplier computes and adds one partial-product (PP) bit at a time using a single full adder (FA). The PP bits are computed in order: A_0B_0, A_1B_0, …, A_nB_0, A_0B_1, A_1B_1, …, A_nB_1, …, A_0B_n, A_1B_n, …, A_nB_n. This is equivalent to scanning the array of Figure 5.29 from top-row to bottom-row, right to left. The trick is to plug the bit-serial adder from 4/6/00 lecture into the shift-add multiplier of the same lecture (both shown below). In the solution shown here, the product register "P" of the multiplier also serves as the resultant register "R" of the adder. Registers "A," "B," and "P" are n-bit right-shift registers. The n-bit multiplexer of the multiplier, once shrunk to single-bit width, can be replaced by an AND gate. The circuit requires n(n+1) cycles to compute A×B. This is <u>20 cycles</u> in the case of a 4×4 multiplier.







Bit-serial multiplier

<u>Control algorithm:</u>

- P←0
- A←multiplicand
- B←multiplier
- Do *n* times:
  - Carry←0
  - Do *n* times:
    - Add in FA: (A_{LSB} AND B_{LSB}) + P_{LSB} + Carry
    - Carry←FA_{Co}
    - Right-shift P, shifting in FA_{sum} (sel=0)
    - Right-rotate A
  - Right-shift P and B together – into B shift P_{LSB} – into P shift Carry (sel=1)
- Result is in {P,B}