EECS150 Midterm2 3/23/06

Name Solutions

SID by Philip Godoy & Friends

Spring 2006 Prof. Pister

Jack Chu
Brian Gawalt
David T. Lin
Guang Yang

	Mean / Max Possible
1	4.68 /5
2	5.65 /15
3	11.96 /20
4	7.46 /15
5	10.53 /20
6	4.69 /10
7	8.54 /15
Total	53.51 /100

1. (5 points)

a. In an 8 bit 2s complement number system, what is the representation of –3? -127?

3: 0000_0011 → (invert and add one) → -3 : 1111_1101

127: 0111_1111 → -127: 1000_0001

b. Using a standard 4 bit 2s complement representation, write down 3-5 and solve.

```
  0011  (3)
- 0101  (5)
------
  ==    (-2)
```

```
  0011  (3)
+ 1011  (-5)
------
  1110  (-2)
```

2: 0010
-2: 1110

c. What are the most positive and most negative numbers that can be represented in a 12 bit 2s complement system, and what are their binary representations?

$-2^{12} = 4096$ → most negative: −2048, 1000_0000_0000

most positive: +2047, 0111_1111_1111

2. (15) Your friend is designing a custom chip with an 8 bit adder. He only needs to represent a few small negative numbers, and sometimes needs to represent numbers a little larger than 127. He decides to change the normal 2s complement representation and allocate ¼ of the wheel to negative numbers and ¾ to positive numbers. He does this by defining that numbers with both b7 and b6 equal to 1 are negative, and all other numbers are positive. The remaining negative numbers are still represented as in normal 2s compelent: -1 is still 8'b11111111 and so on.

a. What is the range of positive and negative numbers that he can represent?

$2^8 = 256$

$\frac{256}{4} = 64$

-64 to +191

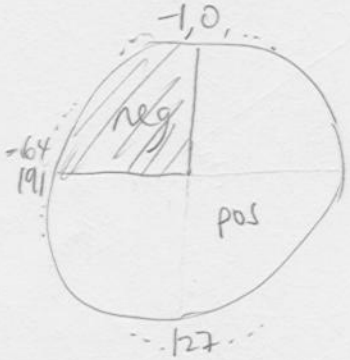
(-64): 1100_0000

(191): 1011_1111 → 255 − 64 = 191

64 × 3 = 192

b. Is it possible for the sum of one positive and one negative number to overflow in this number system?

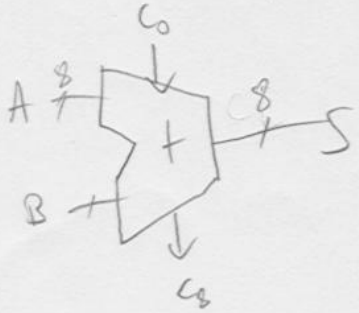
NO

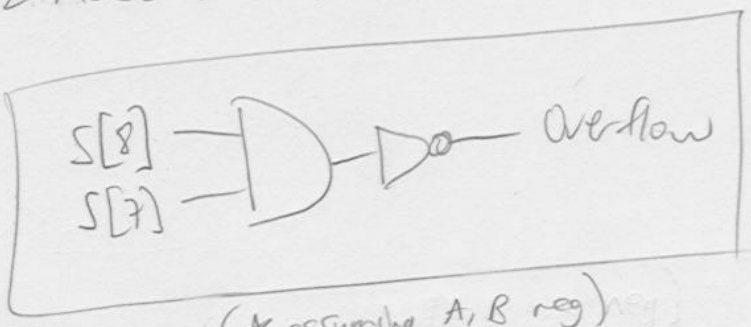


c. (Prob 2 cont.) Given two negative numbers, implement an overflow detector for detecting when their sum overflows. Use only inverters, AND, and OR gates. You don't need to check that the two numbers are negative.

d. Given two positive numbers, implement an overflow detector for detecting when the sum of two positive numbers overflows. Use only inverters, AND, and OR gates. You don't need to check that the two numbers are positive.

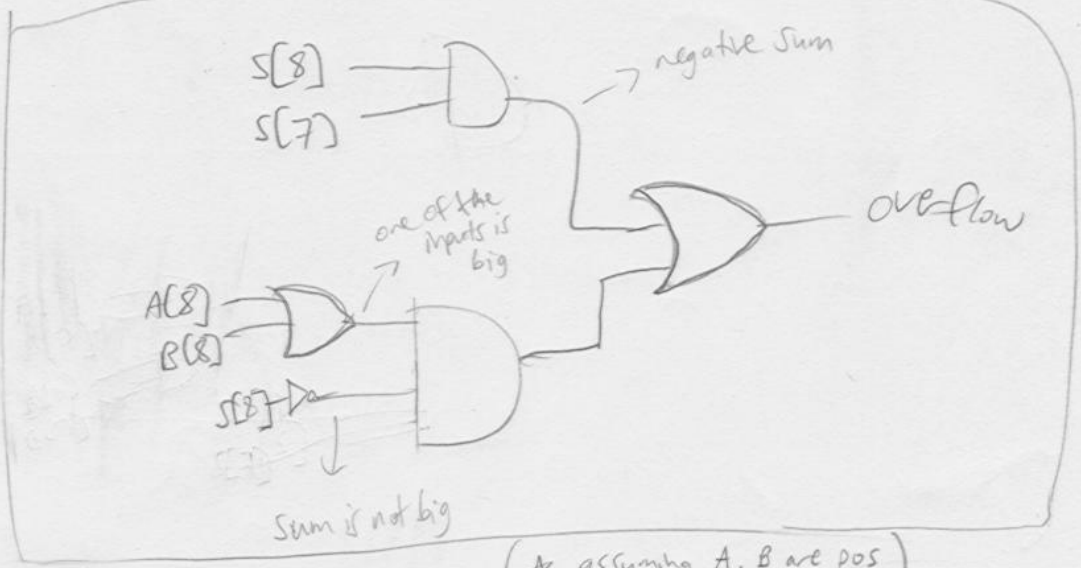
(c) Overflow when (neg) + (neg) = (pos);
pos # if 2 MSB's ≠ 2'b11





(* assuming A, B neg)

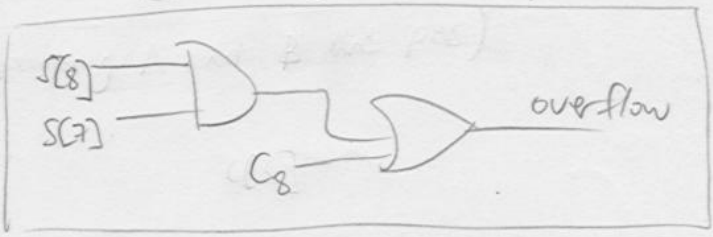
(d) 2 cases: ① (pos) + (pos) = (neg)
② (big pos) + (big/med pos) = (med/small pos)



(* assuming A, B are pos)

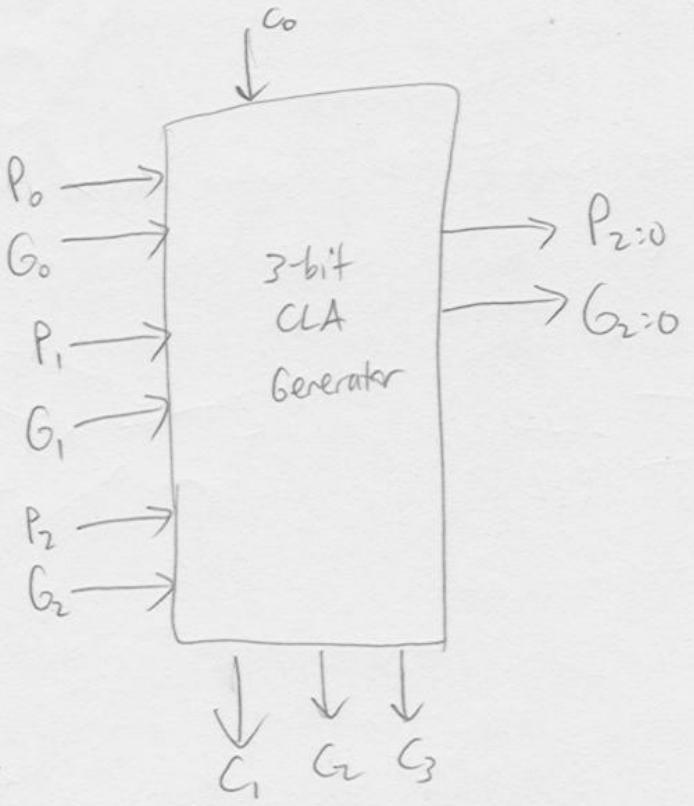
- big positive:
  10 XX_XXXX
- med positive:
  01 XX_XXXX
- small positive:
  00 XX_XXXX
- negative:
  11 XX_XXXX

Better Sol'n:
→ carry-out indicator case #2 (means you looped around the # wheel)



3. (20) In this problem you will design a 9 bit carry lookahead adder:

a. Using inverters and AND and OR gates with arbitrary numbers of inputs, design a 3 bit carry lookahead generator. Assume that single bits generate P and G terms as normal. Your lookahead generator should have inputs Pi and Gi for I=0,1,2, and generate C1, C2, and the group P2:0 and group G2:0 terms.



$X_2X_1X_0$
$+\ Y_2Y_1Y_0$

$$P_{2:0} = P_2P_1P_0$$

$$G_{2:0} = G_2 + P_2G_1 + P_2P_1G_0$$

$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0)$$

$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$

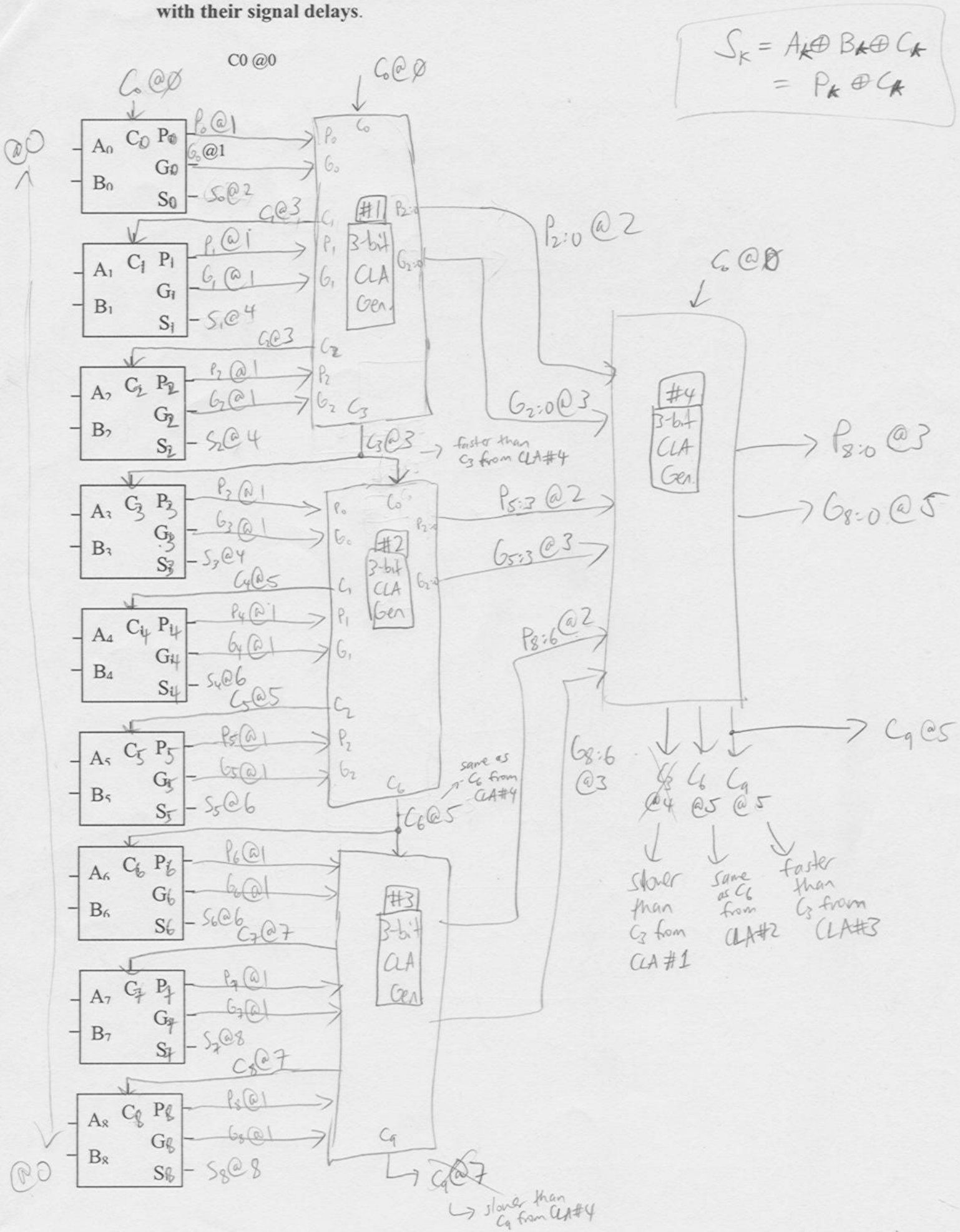
$$C_3 = G_2 + P_2C_2$$

$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

(also, $C_3 = G_{2:0} + P_{2:0}C_0$)

↳ C_3 also generated at next level of CLA logic

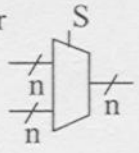
b. Carefully draw all of the connections for a hierarchical 9 bit adder using your carry lookahead parts, and **label all of the outputs of your lookahead parts and all sums with their signal delays**.



4. (15) You have implemented an 8 bit ripple carry adder block on an FPGA and measured its worst-case delay at 1ns. You have 2-to-1 MUXes with a delay of 0.1ns.

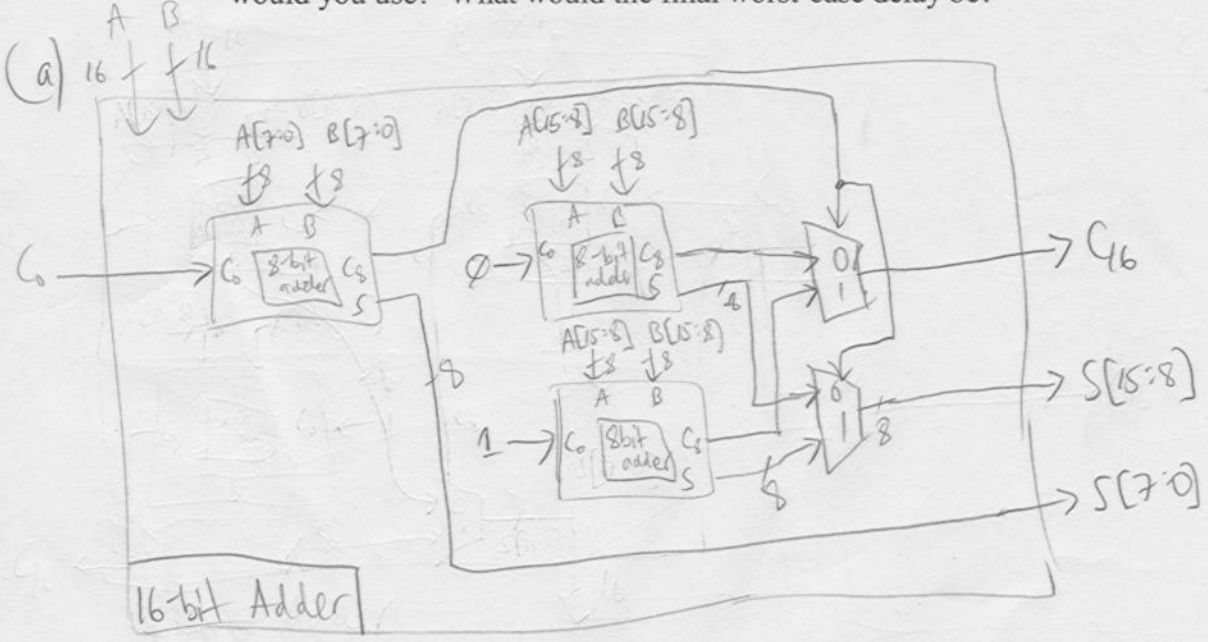
a. Carefully draw a 32 bit carry-select adder using only your 8 bit ripple carry adder and 2-to-1 MUXes as building blocks.

b. What is the worst case delay through your adder?



c. If the delay in your ripple-carry adder is linear in the number of bits, what is the fastest combination of ripple and carry select to implement a 32 bit adder? (i.e. should you use a 4 bit ripple carry adder and more muxes? A 16 bit ripple and fewer muxes?). How many bits in each ripple carry adder, and how many layers of muxes would you use? What would the final worst-case delay be?

(a)



→ 32-bit carry-select adder has same structure as above, just replace the 8-bit adders w/ the 16-bit adders implemented above

(b) Worst-case Delay: $T_{16\text{-bit}} = T_{8\text{-bit}} + T_{mux} = 1.1\text{ ns}$

1 ns, 0.1 ns

$T_{32\text{-bit}} = T_{16\text{-bit}} + T_{mux} = \boxed{1.2\text{ ns}}$

(c) $T_{1\text{-bit}} = \frac{T_{8\text{-bit}}}{8} = 0.125\text{ ns}$

$T_{mux} = 0.1\text{ ns}$

↳ muxes are faster; use more muxes

carry-ripple bits	levels of muxes	Delay
32	0	$32 \times \frac{1}{8} = 4$ ns
16	1	$16/8 + 1/10 = 2.1$ ns
8	2	$8/8 + 2/10 = 1.2$ ns
4	3	$4/8 + 3/10 = 0.8$ ns
2	4	$2/8 + 4/10 = 0.65$ ns
1	5	$1/8 + 5/10 = \boxed{0.625\text{ ns}} = T_{best}$

5. (20) Design a 4 bit ripple-carry ALU which implements the following function: When M=0 (arithmetic)

S1	S0	F
0	0	A
0	1	A+1
1	0	A+B
1	1	A-B

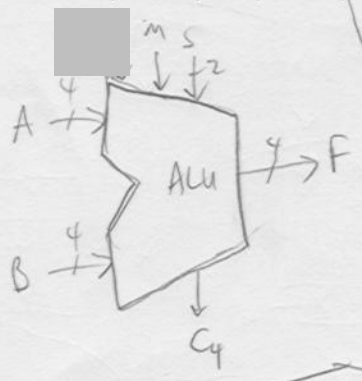
And when M=1 (logic)

S1	S0	F
0	0	A NOR B
0	1	/A NOR /B
1	0	B
1	1	/B

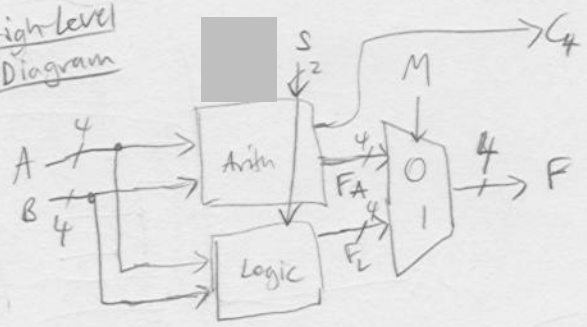
★ Note: There are other correct solutions. Those using 4:1 muxes are not as efficient.

Draw the whole ALU carefully first, and then show the details of the bit slices. You may use inverters, AND, OR, XOR, and MUXes with arbitrary number of inputs.

Module Ports



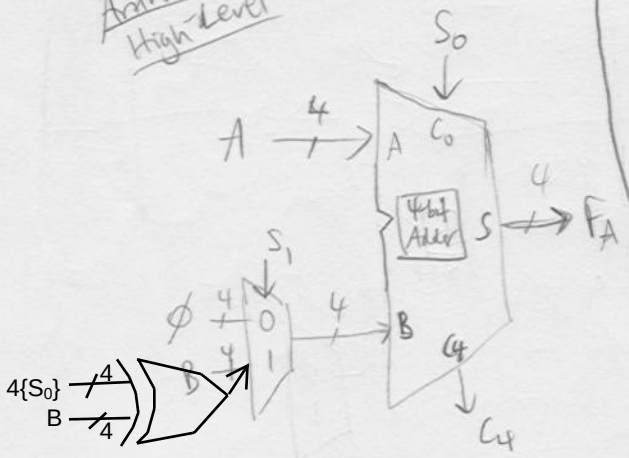
High-level Diagram



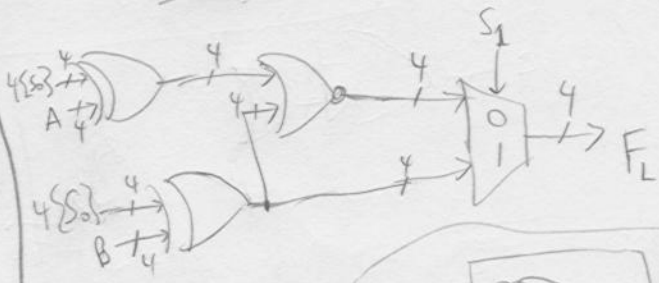
$\overline{A+B} = \overline{A} \cdot \overline{B}$

A B	NOR(A,B)
0 0	1
0 1	0
1 0	0
1 1	0

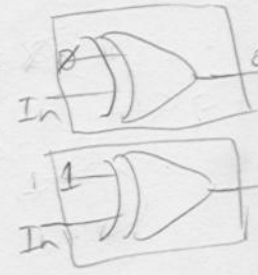
Arithmetic High-Level



Logic High-Level



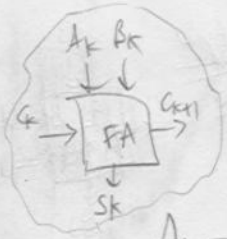
Note:



is a pass gate

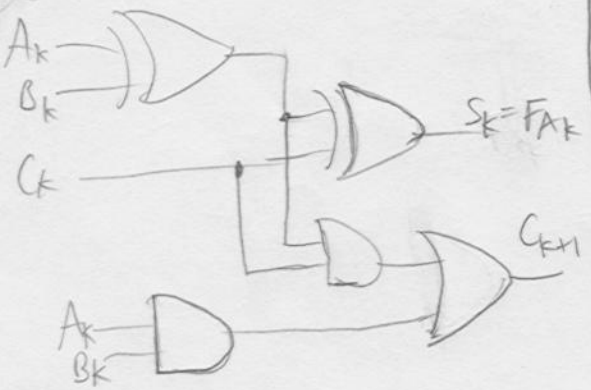
is an inverter

Full Adder

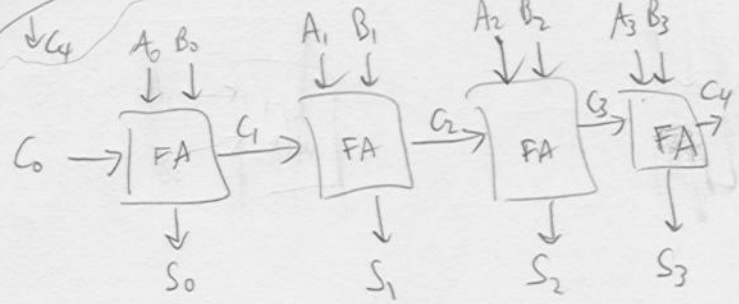


$S_k = A_k \oplus B_k \oplus C_k$

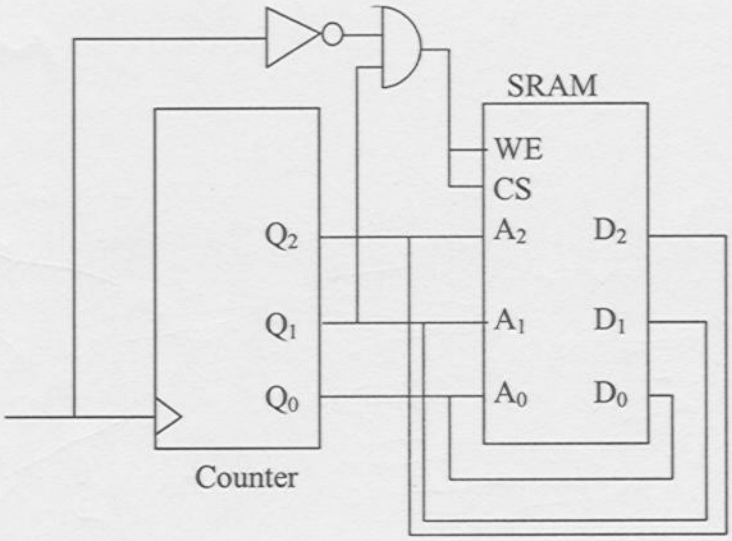
$C_{k+1} = A_k \cdot B_k + (A_k \oplus B_k) C_k$



4-bit Adder



6. (10) In the following circuit, the 8x3 bit SRAM starts with all zeros stored in its memory. The counter is enabled and counts normally. Assume that the clock period is long compared to any SRAM timing constraints. What is the contents of the memory after a few dozen clock ticks? Does it ever change after that?



Addr	MEM(Addr)
000	000
001	000
010	010
011	011
100	000
101	000
110	110
111	111

Doesn't change after few dozen clock ticks.

7. (15) Project related questions:

a. Describe how the N64 communication protocol arbitrates send and receive between the N64 controller and the FPGA.

- challenge/response (FPGA is master, N64 is slave)

- N64 only sends data after FPGA sends it a command

b. Why must active video data to the ADV1794 be clipped? What should the data be clipped to?

- So video data won't be confused w/ SAV/EAV timing reference signals

- Clipped to range 0x10-0xF0

c. What is the purpose of the IO register between the FPGA and the ADV1794 for the video data?

- IO register is at edge of FPGA to speed up signal propagation from FPGA to video codec (minimizes risk of timing errors)