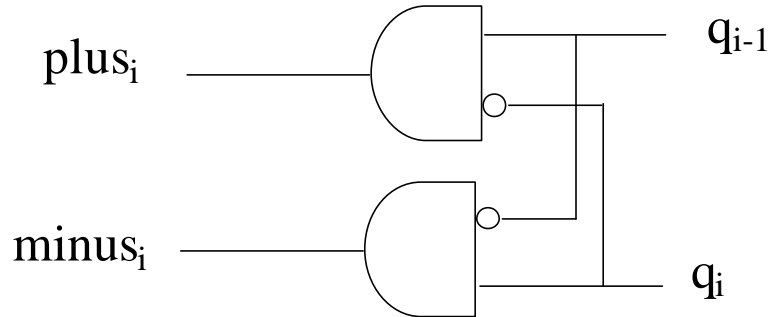
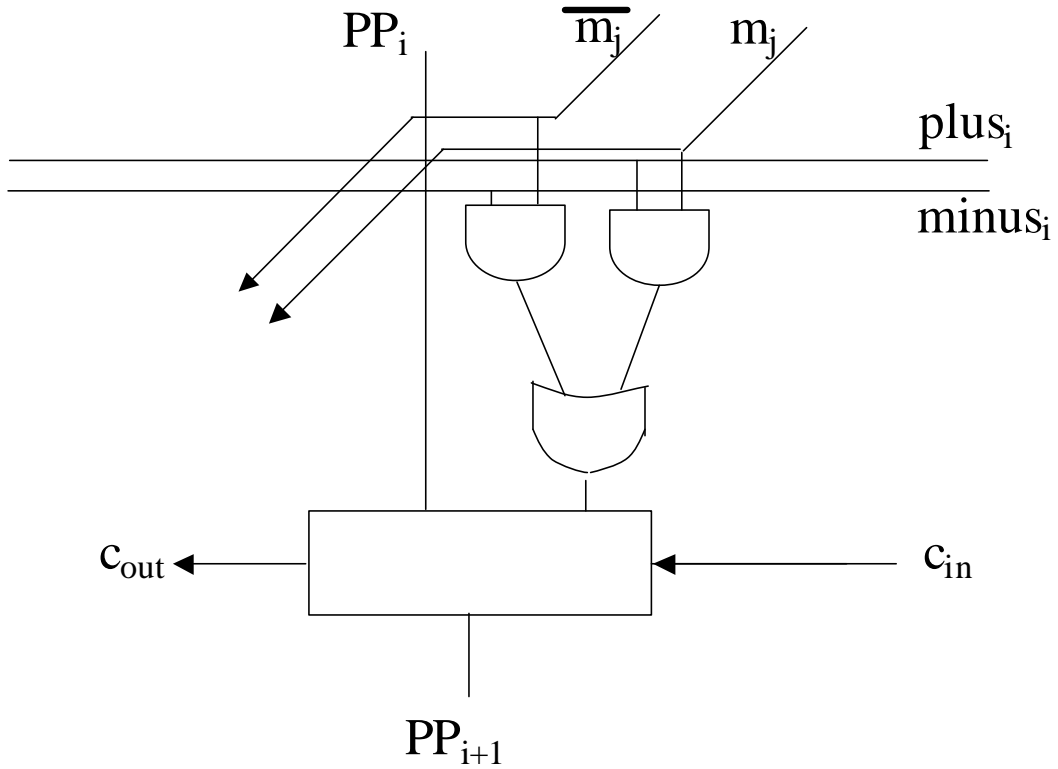


1) [15] Array multipliers:

- a) Draw a gate-level design of a Booth encoder cell. Label the inputs and outputs of the cell as we did for Lab #0.



- b) Draw a gate-level design of the general multiplier cell used in a Booth multiplier. You do not need to show the details of the full-adder block.



- c) For a 32-bit x 32-bit unsigned multiplier, how many levels are required in a Wallace tree? Show how you found the result. You do not have to show the tree or the connections, but must show the technique you used to find the answer.

A 32-bit multiplier will require 32 rows. A Wallace tree does a 3-2 reduction at each level:

- Level 1: 32 rows becomes 10x2 with 2 rows left over
 - Level 2: 22 rows becomes 7x2 with 1 row left over
 - Level 3: 15 rows becomes 5x2 with 0 rows left over
 - Level 4: 10 rows becomes 3x2 with 1 row left over
 - Level 5: 7 rows becomes 2x2 with 1 row left over
 - Level 6: 5 rows becomes 1x2 with 2 rows left over
 - Level 7: 4 rows becomes 1x2 with 1 row left over
 - Level 8: 3 rows becomes 1x2 with 0 rows left over
 - Level 9: 2 rows becomes 1 with a Carry-Lookahead Adder
- 9 levels are required.

You might also argue that you need 31 rows, since the first row degenerates to only AND gates:

- Level 1: 31 rows becomes 10x2 with 1 row left over
- Level 2: 21 rows becomes 7x2 with 0 rows left over
- Level 3: 14 rows becomes 4x2 with 2 rows left over

This now becomes the same as the answer above, so there is no saving in terms of # of levels

- d) For a 32-bit x 32-bit Booth multiplier, how many levels are required in a Wallace tree? Show how you found the result. You do not have to show the tree or the connections, but must show the technique you used to find the answer.

A Booth multiplier has the same number of rows as an unsigned multiplier, so the answer is the same as for part c.

- e) What would be the result from part d if the multiplier operand were the hexadecimal constant \$760, assuming you wanted to use as few rows as possible? Booth Bit-Pair Recoding is **not** to be done.

\$760 is %0 ... 0 1 1 1 0 1 1 0 0 0 0 0.

The Booth encoding is +1 -1+1 -1

So 4 rows are required.

- Level 1: 4 rows becomes 1x2 with 1 row left over
 - Level 2: 3 rows becomes 1x2 with 0 rows left over
 - Level 3: 2 rows becomes 1 with a Carry-Lookahead Adder
- 3 levels are required.

If you consider the degenerate first row not requiring a level, there IS a one level reduction in the answer.

f) What is the advantage of Booth Bit-Pair Recoding? (briefly)

It halves the number of rows and thus speeds up the multiplier by roughly a factor of two.

g) Can a Wallace tree be used with Booth Bit-Pair Recoding? Explain.

Yes. The Booth Bit-Pair Recoding technique still generates independent partial products that can be grouped into a Wallace tree.

2) [13] Clock circuits

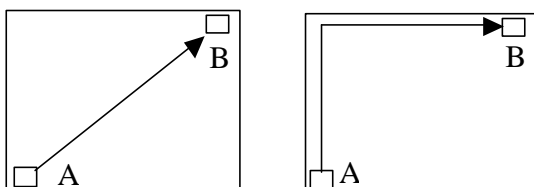
a) Assume you have an integrated circuit chip that is 2cm x 2cm, the clock is generated in one corner of the chip, and the data from one flip-flop may have to be propagated to another, from any one place on the chip to any other.

- Draw a picture of the worst-case timing path between flip-flops.
- What is the absolute minimum clock period?

Assume setup times of 0.1 ns, flip-flop propagation times of 0.2 ns, no other combinational logic in the paths and signal speed through circuit wires of $0.7c$ where c is the speed of light in a vacuum (300,000 kilometers / sec). You need not reduce the result to a single number. State any other assumptions made as part of your solution.

You need not worry about the clock, since it will be distributed using an H-Tree or similar technique to reduce skew.

Two paths were considered correct:



The path on the right is probably more realistic for an IC laid out on a rectangular grid. Steps of vertical and horizontal lines across the chip would result in this same path length.

For this case: The propagation time from A to B would be

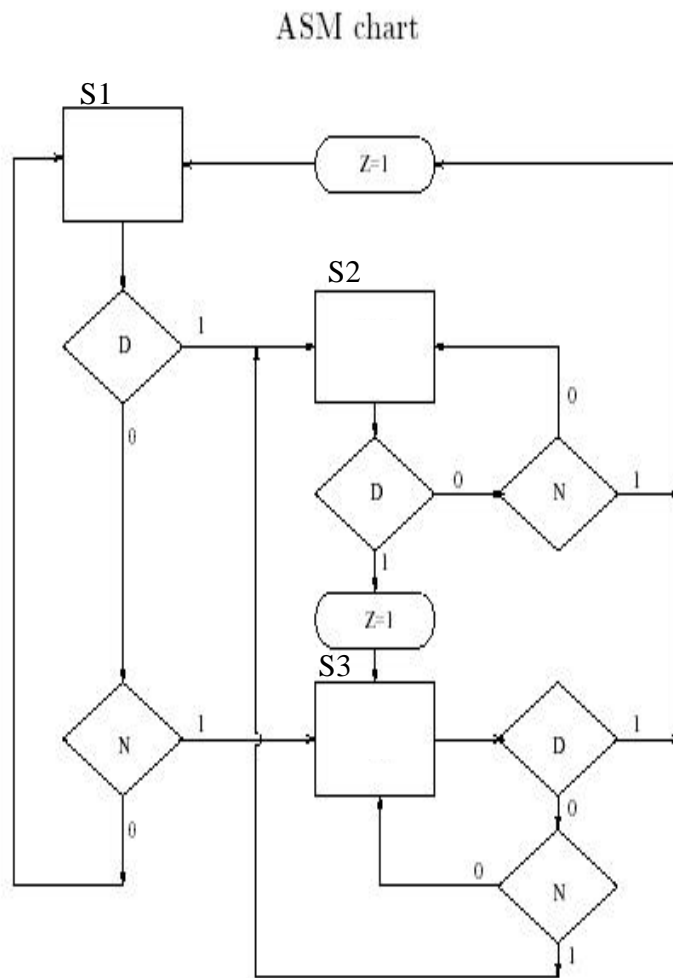
$$t_{AB} = d/s = (4 \times 10^{-2}) / (.7 \times 3 \times 10^8)$$

therefore

$$\text{Min clock} = t_{AB} + .2 + .1$$

b) **Briefly** describe a situation in which clock skew can be beneficial.
 clock skew is beneficial when the clock takes longer to reach the FF at the end of a path than it does to reach the FF at the beginning of a path.

3) [12] The ASM chart below represents a finite state machine:



a) Is this a Moore or a Mealy machine? Mealy.
 How do we know? -> Output is in a conditional output box.

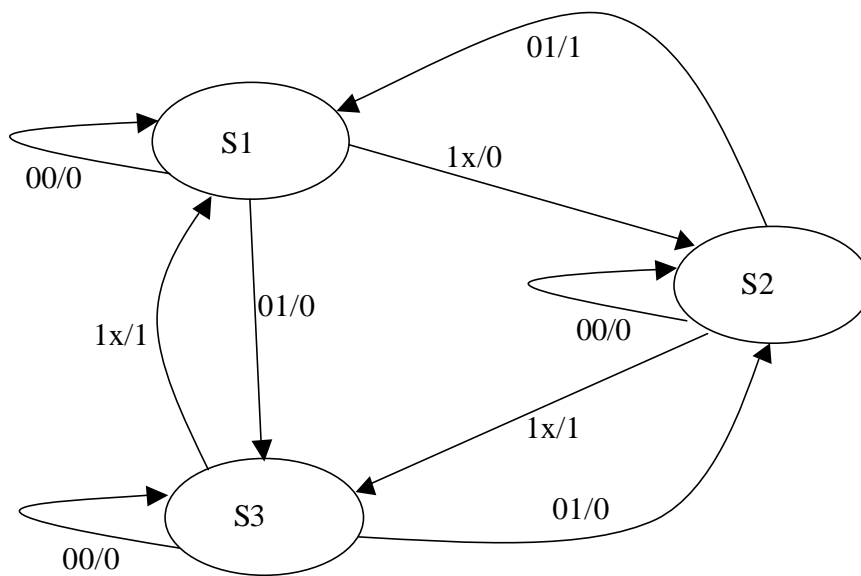
b) What are the inputs and the outputs?

Inputs: D, N

Output: Z

c) Draw a state diagram (with bubbles and arcs) for the machine

key: DN/Z



4) [20] ASM Charts

A serial input w comes into a computer at $1/3$ the system clock frequency. The input data is to be assembled by your circuit. The idle state of the input is high; the data received is 8 bits in length; the transmission begins with a low (start) bit, followed by the data LSB first, the rest of the data, and then a low (stop) bit. The circuit should provide an 8-bit output B and a done signal D .

a) Give pseudo-code for a solution to this problem

Data is coming in serially and needs to be held for parallel output. The logical choice for B , then, is a shift register - right shifting since the bits come in LSB first.

There are three more problems: How to start the data collection, how to stop the data collection and how to deal with the $1/3$ frequency.

How to start: Since there is a done signal, there must be some 'handshaking'. This has been done in class using an 's' signal input. The idle level of the input is high, the start bit is low, so we must wait for the signal to go low before we start to assemble the input.

How to stop: The usual approach would be to count the number of input bits and to stop when the entire message has come in and been assembled.

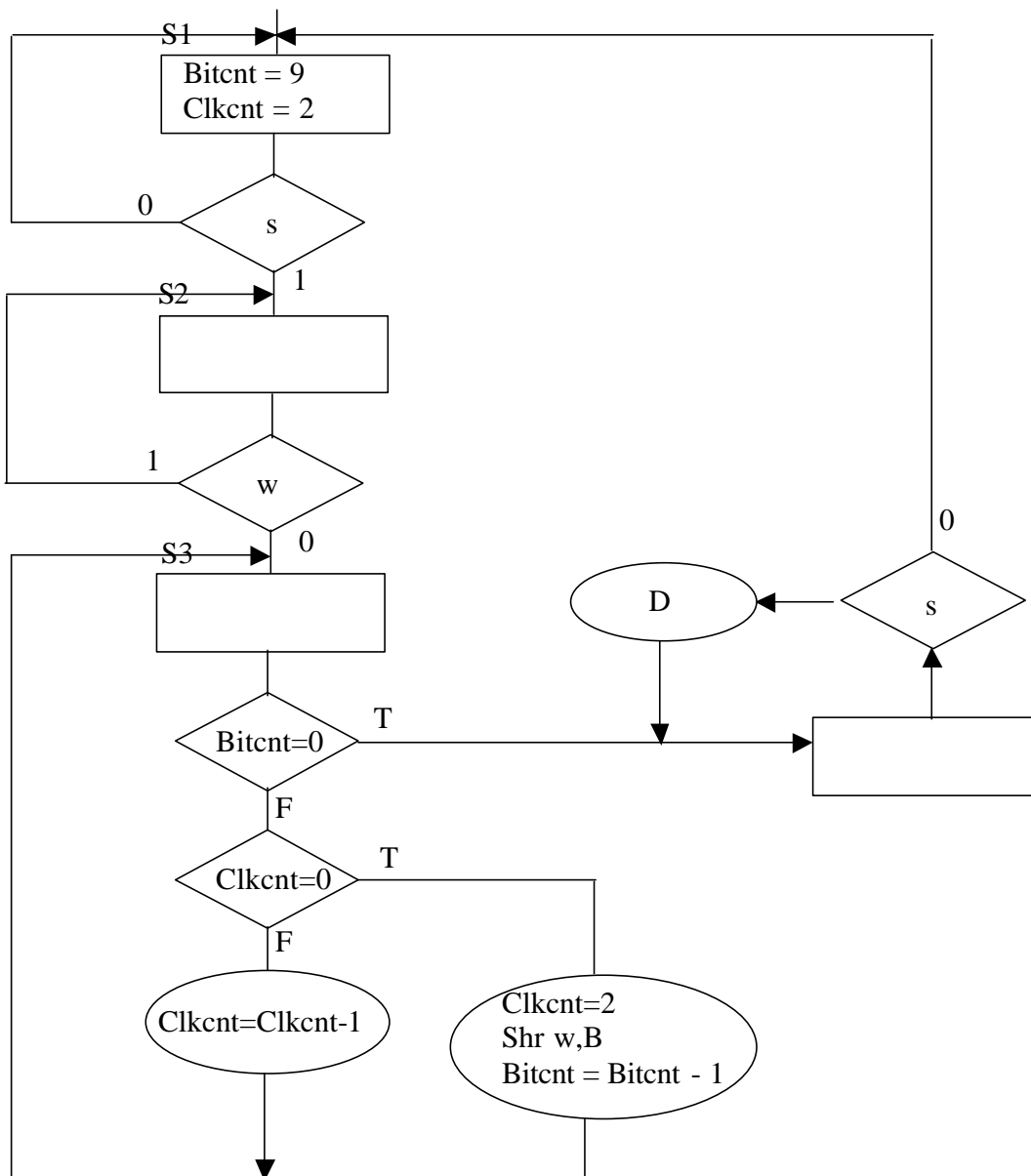
How to deal with the $1/3$ frequency: The usual approach would be to have a counter that only made the circuit active one clock in three. We do **not** want to gate the clock.

```
while ~s
    D = 0           // done signal
    Bitcnt = 9     // counts bits yet to assemble + start bit
    Clkcnt = 2    // enable every 3rd clock
while w           // wait for data to go low for start bit
while Bitcnt     // loop until all bits assembled
    if Clkcnt == 0
        Clkcnt = 2
        Shr w,B
        Bitcnt = Bitcnt - 1
    else
        Clkcnt = Clkcnt - 1
while s
    D = 1
```

Aside: Note that there are alternate methods here which may have some advantages. Together the following eliminates the counters.

- If shift register B is preloaded with 1's then one can stop when the shift out from B is a zero (the start bit).
- By tripling the number of bits in B, the register can shift every system clock. The output in this case would be every 3rd bit of B.

b) Draw an ASM chart for your pseudo-code



c) Show the data path circuit

