Chapter 5: solutions & guide

Rob Rodgers (knave@acm.org) 4-20-1998 CMSC311-Spring 98 (Dr. Hugue)

Assigned problems: 3, 4, 5, 6, 8, 11, 13, 16, 20, 21, 24, 26

!!! PLEASE ATTEMPT TO DO THE ASSIGNMENT BEFORE CONSULTING THE ANSWERS. THINKING ABOUT WHAT THE PROBLEMS WANT YOU TO DO IS 2/3rds OF WHAT YOU'RE LEARNING !!!

Material is introduced as necessary.

Printing information: To print these notes, you need to copy this postscript file to the CS cluster machine where your CMSC311 account resides. The easiest way is to use the LYNX character mode web browser. Log into your cluster account and type "lynx http://www.cs.umd.edu/class" and proceed to the CMSC311 class homework page. Use the arrow keys to highlight this file and press D to download it. You will be asked what you want to do, select save to disk and give it a filename. After saving the file, press 'q' then 'y' to exit lynx. Finally, use the command "qpr –q csc-ps <filename>" where <filename> is the filename you gave lynx. This will print the file to the CSS building free postscript printer.

Note: I am not repeating the diagrams that are already available. FIRST, get the postscript solutions that are already posted to the web page. Use the same procedure above to download and print the other solutions. I'm going to omit portions of the answers which already appear in the provided files and provide explanations. This file is a SUPPLEMENT to the posted postscript file.

PROBLEM 3

This problem asks you to simulate the behavior of a 4 bit shift register. You are told that the contents of the register at T=0 are the bits (from left to right) 0100. The register is to be shifted six times to the right with the input for each bit being taken from 010110, left to right. Thus, we can visualize the shift register as

011010 -> 0100

at T=0. After 4 cycles, the current contents of the register have been totally replaced (shifted out) by the first four bits of our input string:

0100   0010   1010   0100   1010

The next two cycles just continue the shift pattern:

1010   1101   0110

Since we've shifted six times and our input is six bits, we can know from the beginning that our resulting contents will be the last 4 bits of the input string.

PROBLEM 4

Differentiate serial & parallel transfer. Start with the definition of parallel transfer. In parallel transfer, all of the bits (or n bits where n is the width of the parallel connection) are transferred to the destination in a synchronous fashion (in one cycle, or in two cycles, or in three cycles, but in all cases the start and end times for all bits are the same).

Serial transfer is the opposite. Each bit is transferred sequentially, the second after the first, and so on. The transfers themselves all take the same amount of time (one cycle, two cycles) but your total time to transfer n bits is n*m where m is the time it takes to transfer a single bit.

Converting parallel to serial means we want to have some register that allows us to load all the bits simultaneously and then output them one bit at a time to a 1 bit wide serial line. Thus we wand a device that has parallel load and serial output, or a shift register with parallel LOAD. To convert serial to parallel we want a device that takes as an input a single serial line and can use it to fill the entire register. Thus we know we need a register with serial LOAD (either left or right) and that means it must be a shift register (otherwise we'd only fill one bit, and it would constantly be overwritten). We need to be able to read all of the bit contents simultaneously, and so we need PARALLEL READ: a shift register with parallel read and serial load.

Serial allows us to use a single wire for the connection, but the time scales linearly with the number of bits to be transferred. Parallel takes a fixed amount of time but requires at least as many wires as will be transferred in parallel.

Note that we could design, say, 16 bit registers that transmit their contents serially in **parallel 4 bit packets**. How might this work? Might be worth trying.

PROBLEMS 5 AND 6

For each bit, A gets the sum of A+B+Carry. B gets its input from the serial line (we don't care).

t=0	1	2	3	4
A: 0101	0010	0001	0000	1000
B: 0011	x001	xx00	xxx0	xxxx
C: 0	1	1	1	0

We already have a serial shift adder given to us and want to make it into an adder/subtractor. How can we do this? We know that do do this with a ripple adder with use XOR gates to invert input B and set the initial carry state to 1 instead of zero (i.e., complement and add one). We do the same thing in this case: invert each bit of B and set carry to 0.

Results: A<B -> (1) result is in 2's comp form AND (2) the final carry value will be zero

PROBLEM 8 (see postscript for diagram)

PROBLEM 11

(a) Ring counter. This silly device uses four flipflops yet only has four states. Since the output connects to the input, we know that starting at some bit and reading left to right, we will always see 1000. In other words,

1000 -> 0100 -> 0010 -> 0001 -> 1000 (back to first state)

(b) We have four states for the ring counter, and this counter just inverts the bits. Thus we know we'll always see some section of the pattern START_INVERSE, or 1000 0111. This allows us to have a total of eight states, so:

0000 1000 1100 1110 1111 0111 0011 0001 0000

(c) Now they want us to derive the number of states, but as we saw from part b, the number of states is directly proportional to the number of bits, because each bit can be in one of two states but the relavent order of the bits is FIXED. Thus the number of states in a ring counter = the number of bits and the number of states in a switch tail counter is 2*the number of bits. How many states does an ordinary counter have? 2^bits.

PROBLEM 13

You should probably recheck page 234 concerning the behavior of ripple counters in terms of complementing. Basically, if the LSB transitions from 1 to 0, it complements the flipflop for the bit above it. This proceeds up the chain (ripples up the chain) for each bit being complemented. Thus we can see from the question:

100110011111

that the LSB is 1 and thus will go to zero and comp he next one, which is one and will go to zero, and so on, and so forth. Our total count of complemented flipflops is 6 (count from the right)

(b) is 000111111111 Same reasoning tells us that every bit up to the second zero bit (bit 10) will be complemented since all of those 1's will comp to zero. Thus, the answer is 10.

PROBLEM 16

We need to convert a D flip flop to a T flipflop. If you followed my advice in the CH4 solutions you would have already done this, as well as J-K flipflops. ANY flipflop can be made with any other. Consider a T with D flipflops:

T at Q(t+1) = T'Q(t) + TQ'(t) (complement)
But a D at Q(t+1) = D (just repeats the input)

So, we want to build some circuit that has the properties of a T flipflop. Fill out the first three columns of our table, just like always:

Q(t)	Input	Q(t+1)	needed input (to the D flipflop)
0	0	0	?
0	1	1	?
1	0	1	?
1	1	0	?

What do we need to do to get the D to output column 3? Right: we need to give it column 3. But we can do that by taking Input XOR Q(t) and feeding it to the D flipflop. As an exercise, try making a J-K from D and T flipflops, and a D from a T and J-K, and a T from a J-K.

PROBLEM 20 and PROBLEM 21 (see diagram)

PROBLEM 24

We're being asked to design a binary counter with the sequence 0, 1, 2, 3, 4 with JK flipflops. We're going to use 1 flipflop per output bit, and thus we know we need three flipflops (since 4 requires three bits).

A counter is just like any other state machine, we just use excess flipflops. Since it's like any other state machine, we need to do what?

RIGHT: We need a state table for our three flip flops (A, B and C). But let's be smart and not blindly fill out the table. Instead, write the first row:

A(t)	B(t)	C(t)	A(t+1)	B(t+1)	C(t+1)
0	0	0	0	0	1

and instead of blindly writing out ever flipflop state (half of which NEVER appear in our state machine, which has 5 states instead of 8) let's just write out the states THAT WE NEED. Thus, if a state appears in the right hand side, we need to put it into the left, but other than our start state (000) we don't ever need to consider any other states.

Here is our table:

A(t)	B(t)	C(t)	A(t+1)	B(t+1)	C(t+1)	inputs for A	inputs for B	inputs for C
0	0	0	0	0	1	?	?	?
0	0	1	0	1	0	?	?	?
0	1	0	0	1	1	?	?	?
0	1	1	1	0	0	?	?	?
1	0	0	0	0	0	?	?	?

But what are the inputs? Apply the J-K flipflop equations for A, B and C. What we can observe is that:

Aj = BC
Ak = 1 (always)

Bj = C
Bk = C (hmm)

Cj = A'
Ck = 1

Now we can build our circuit.

Part B asks us to do the same circuit, but we need to add on a 5 and 6 state. So instead of starting over, observe that this circuit is a superset of the circuit in part A, but that the last line of the table,

1	0	0	0	0	0	?	?	?

changes to:

1	0	0	1	0	1	?	?	?

and now we need to add a line for state 5 (1 0 1, appearing on the right in the newly added line)

1	0	1	1	1	0	?	?	?

(since 5 goes to 6) and since 6 goes to zero, we add the closing line,

1	1	0	0	0	0	?	?	?

Now we need to rederive the equations:

Aj = BC
Ak = B
Bj = C
Bk = A+C
Cj = A'+ B'
Ck = 1

PROBLEM 26

We want to design a "counter" with the following repeated binary sequence: 0 1 2 4 8.

First question: how many flipflops do we need? Well, we have two alternatives. One, we can observe that the counter we are designing has exactly 5 states, and given what we learned in chapter 4 we know we can build this circuit with some combinational logic and three flipflops. Two, we can use one flipflop per output bit, and thus we need 4 flipflops (8 = 1000) for the circuit. Let's keep things nice and simple and use four flipflops.

Now we need to build a state table. This should be familiar to us by now:

A(t)	B(t)	C(t)	D(t)	A(t+1)	B(t+1)	C(t+1)	D(t+1)
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0
1	0	0	0	0	0	0	0

What's missing from this table? Right: the inputs to the flipflops. Since these are D flipflops, we want the input to each flipflop to be the output desired for t+1. In other words, if we want

0 0 0 0 to transition to 0 0 0 1

we need to give A 0, B 0, C 0 and D 1, which degenerates into four rows:

0	0	0	0	0	(for A)
0	0	0	0	0	(for B)
0	0	0	0	0	(for C)
0	0	0	0	1	(for D)

Now we know we only write the equations for cases where the output equals 1, thus A, B and C don't get anything from this row, but we now have an equation for D: D = A'B'C'D'

And since D is never set to 1 in any other state, we know that this completely describes the input to D. We repeat this process for A, B and C and we additionally find that:

A = B
B = C
C = D

Thus completing our table. We can now build the circuit.