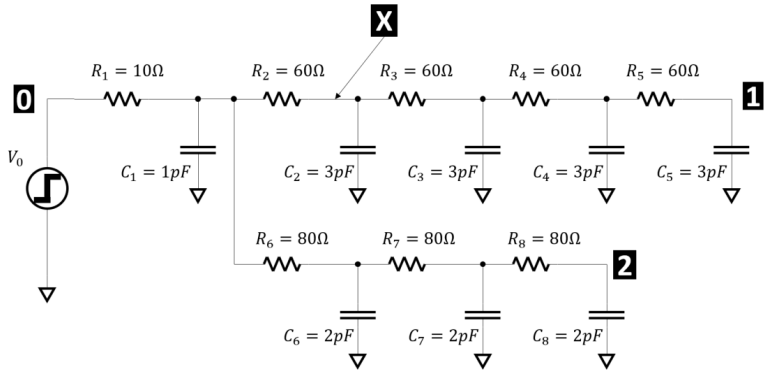
EECS 168 Introduction to VLSI Design Sheldon Tan

Homework 4

1. For the RC circuit shown below, the required arrival times at node 1 and 2 are respectively $2ns$ and $1.2ns$.




1) Compute the required arrival time at the source node 0.
If we insert one buffer with $C_{buf} = 2pF$, $R_{buf} = 10\Omega$, and $D_{buf} = 5ps$ at node X, what is the new required arrival time at node

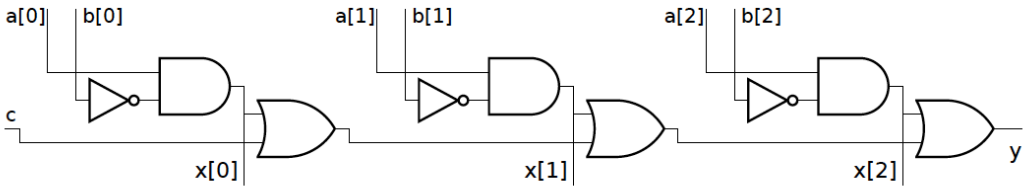
2. Draw the schematic of the logic circuit described by the Verilog code below:

```
module pie( x, y, a, b, c);
        input a, b, c;
        wire t1, t2;
        output x,y;

        xor #2 x1(t1,a,b);
        not #1 n1(x,t1);
        and #3 a1(t2,x,c);
         or  #3 o1(y,t2,b);
endmodule
```

(b) What is the critical path of this design from primary inputs to outputs and what is the delay of the critical path in terms of unit delay?

3. Notice that the logic below consists of three repeated parts.
   (a) Write a Verilog explicit structural description of the logic which consists of two modules, one module, name it **part-logic**, will be for the part that's repeated, the other, name it **whole-logic**, will instantiate part-logic three times and interconnect them appropriately. Write the **part-logic** using the Verilog primitives. Choose appropriate inputs and outputs for the two modules based on the diagram.
   (b) Repeat (a) by writing the **part-logic** using the continuous assignment statement.




4. The Hamming encode/decoder in the lecture detected and corrected one error bit. By adding a thirteen bit which is the exclusive-OR of the other twelve bits, double bit errors can be detected (but not corrected). Add this feature to the example and modify the noisy channel so that sometimes two bits are in error. Change the $display statement to indicate the double error. See the additional supplemental information about the 13-bit Hamming code. Please show all the Verilog codes for your design.

The basic Hamming code can detect and correct an error in only a single bit. Some multiple-bit errors are detected, but they may be corrected erroneously, as if they were single-bit errors. By adding another parity bit to the coded word, the Hamming code can be used to correct a single error and detect double errors. If we include this additional parity bit, the previous 12-bit coded word becomes $001110010100P_{13}$, where $P_{13}$ is evaluated from the exclusive-OR of the other 12 bits. This produces the 13-bit word 0011100101001 (even parity). When this word is read from memory, the check bits and also the parity bit $P$ are evaluated over the entire 13 bits. If $P = 0$, the parity is correct (even parity), but if $P = 1$, the parity over the 13 bits is incorrect (odd parity). The following four cases can occur:

| | |
|---|---|
| If $C = 0$ and $P = 0$ | No error occurred. |
| If $C \neq 0$ and $P = 1$ | A single error occurred that can be corrected. |
| If $C \neq 0$ and $P = 0$ | A double error occurred that is detected but cannot be corrected. |
| If $C = 0$ and $P = 1$ | An error occurred in the $P_{13}$ bit. |

Note that this scheme will detect more than two erroneous bits in many cases, but is not guaranteed to detect all such errors.

A modified Hamming code to generate and check parity bits for a single-error-correction, double-error-detection scheme is most often used in real systems. The modified code uses a different parity check bit scheme that balances the number of inputs to the logic for each check bit and thus the number of inputs to each circuit that does the checking. The balancing minimizes the delay through the error correction and detection circuits. These circuits can be used in a RAM subsystem to add check bits during write operations and to correct single errors and detect double errors during read operations. (See Problem A.6-5-6.)