Data Provided: None

The University Of Sheffield.

DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

Spring Semester 2010-2011 (2 hours)

Answers to EEE310 Introduction to VLSI Design 3 / EEE6036 VLSI Design 6, Questions 1..4

1. a. *Show that the small-signal, on-state resistance of an n-type pass transistor as shown in* ***Figure 1(a)*** *is* $r = \dfrac{1}{\beta_N \cdot (V_{DS} - V_T)}$ *when* $V_G = V_D$

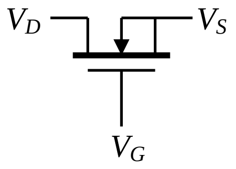


Figure 1(a): n-type Pass Transistor

In this case, $V_{DS} = V_{GS} = V_{DD} - V_S$. Consequently, $V_{DS} > V_{GS} - V_T$ and so the mode is saturated. Thus $I_{DS} = \dfrac{\beta_N}{2}(V_{GS} - V_T)^2 = \dfrac{\beta_N}{2}(V_{DS} - V_T)^2$

$\dfrac{dI_{DS}}{dV_s} = \beta_N (V_{DS} - V_T) = \dfrac{1}{r}$. Therefore $r = \dfrac{1}{\beta_N (V_{DS} - V_T)}$

(4)

b. Three *n*-type pass transistors are cascaded in series as shown in **Figure 1(b)**.

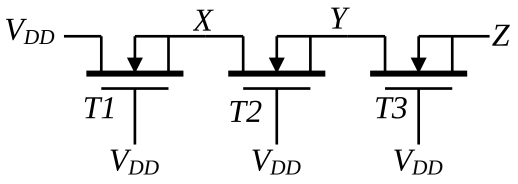


Figure 1(b): Cascaded *n*-type Pass Transistors

What will the approximate voltages at points X, Y, and Z be for the voltage conditions shown. State any assumptions that you make.

Assume that the transistors are perfect and there is no leakage or sub-threshold conduction. Additionally, for the purposes of working out the end point that point Z has an associated capacitance. As this capacitance is charged, points X, Y, and Z rise until point X reaches a value of V_{DD}-V_T. At this point T1 just cuts off preventing current flowing and point Z can charge no further. At this point, with no current flowing, T2 and T3 must also be on the point of cutting-off and this is when the voltages at Y and Z are also V_{DD}-V_T.

(4)

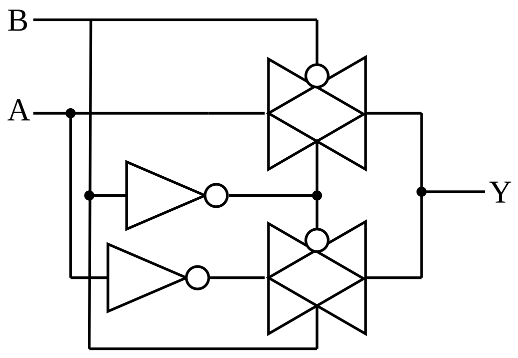
c. *State whether each transistor in* **Figure 1(b)** *would be in the ohmic or saturated region if a current were to flow.*

Assuming a current flows, point X will be at V_{DD}-V_T- Δ such that the current is equal to $\beta\Delta^2/2$. However, given that V_D=V_G. then it must be saturated. For T2, point Y will be at V_{DD}-V_T- Δ_1 and as long as $\Delta_1 > \Delta$ then $V_{DS} = \Delta_1 - \Delta$ and V_{GS} = V_T-Δ_1 and so T2 will be ohmic. Current continuity should mean that $\Delta_1 = \sqrt{2}\Delta$ and the condition is met. Similarly, T3 will also be ohmic. **(4)**

d. *In* **Figure 1(b)**, *the substrate connection is shown connected to the transistor's source connection. If this circuit were part of a digital MOS IC where would you normally expect the substrate to be connected for transistors such as these and why would this be the case.*

For *n*-type FETs, the substrate would be connected to the most negative point in the circuit to ensure that all the pn junctions between the S,D and the substrate are reverse biased. Additionally, in a practical implementation numbers of *n*-type FETs would be grouped into islands (either in a *p*-well or in the *p*-substrate) to save space. If each FET's substrate we connected to its Source then each FET would need to be in its own isolated well – this is impractical in a digital circuit. **(4)**

e. *Show how two transmission gates and two inverters can be combined together to create an XOR gate.*



(4)

2. *A logic circuit is shown in* ***Figure 2****.*

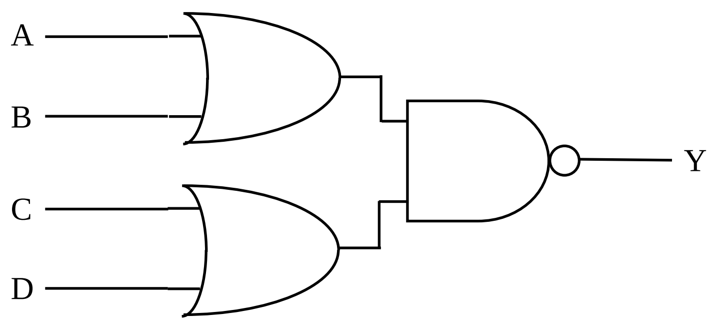


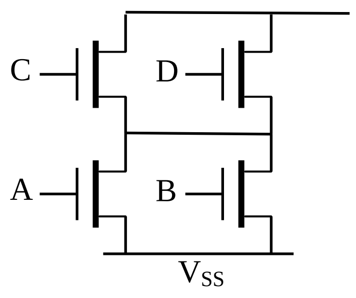
Figure 2: Logic Circuit

a. *Convert the logic circuit into a standard-CMOS transistor-level circuit.*

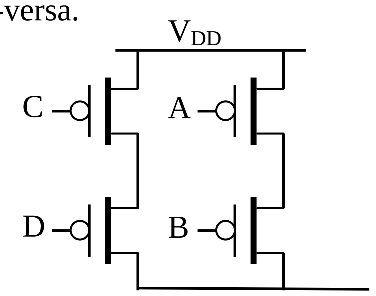
$Y = \overline{(A+B)(C+D)}$

To find the pull-down network, find $\overline{Y} = (A+B)(C+D)$ and convert true terms on RHS with + being parallel combination and . being series combination.

This leads to:



The pull-up network can be found by inspection, converting parallel to series networks and vice-versa.



(8)

b. *Size the transistors (as a multiple of a minimum-sized n-type FET) for a minimum sized logic circuit, stating any assumptions that you make.*

In making a minimum-sized logic gate, we assume that the effective resistance between the output is, worst-case, equivalent to a minimum sized *n*-type FET. Note, that *p*-type FETs will need to be 2x wider than expected because we can also assume that $\mu_h = 0.5\mu_e$.

For pull-down T_A, T_B, T_C, T_D = 2.

For pull-up T_A, T_B, T_C, T_D = 4 **(6)**

c. *Estimate the capacitance associated with each of the inputs and wires within the circuit (you can neglect the interconnect capacitance), given that the gate capacitance of a minimum-sized n-type FET is 0.7fF.*

We need to find the capacitance on the 4 inputs A…D and the output Y. Each input looks at an identical load equivalent to 6x minimum *n*-type FET and so the capacitance on each input is 4.2fF. The output Y, we can assume is loaded by the drain capacitances of the FETs connected to it (we ignore the others) and assume that the drain capacitance is $0.5C_G$. Consequently, the capacitance on Y is 0.5*8 minimum *n*-type FETs = 4 * 0.7 = 2.8aF + any load capacitance that Y is driving. **(4)**

d. *Rather than implementing the circuit in **Figure 2** as a single, combined CMOS circuit, you decide to implement it as a combination of two separate CMOS OR gates and a CMOS NAND gate. How much less efficient will it be than the single CMOS circuit in terms of numbers of transistors used.*

A NAND gate costs 4 transistors whilst an OR gate is formed from a NOR gate (4 transistors) + an inverter (2 transistors). Consequently, the total number of transistors is 16 transistors as opposed to 8 for the direct implementation (50%). Also, the area of the transistors in the direct implementation is 24L (L is the length of the transistors) whilst for the minimum sized equivalent constructed from NAND and OR gates, the area is 34L (74%). **(2)**

3. a. i) *What is fault-coverage when applied to testing?*

Fault coverage is the proportion of faults that will be uncovered by a set of tests. So if fault coverage is 95% then 5% of faulty ICs will, apparently, pass the tests. **(3)**

ii) *Why is fault-coverage so important in ensuring that systems manufactured using ICs can be sold?*

Fault coverage is very important because the cost of not discovering a fault early on in the manufacturing of a system escalates the cost of the fault: either repairing/reworking the problem, discarding any sub-system the faulty IC is in or field returns. **(1)**

iii) *Show the effect of fault coverage using an IC yield of 0.7, with a fault-coverage of 98%.*

You should assume that the ICs are tested, then 20 ICs are assembled into a system where the manufacturing yield on the system assembly is 99.5% and, again, the fault-coverage on testing the systems is, again, 98%.

This is compounded by the yield of ICs, which is 0.7. As an example, we manufacture ICs with a yield of 0.7 and test them with a fault coverage of 0.98. This means that of the 0.3 of faulty ICs, we only discover 0.294. We discard the ICs that fail the tests and this leaves us with 0.706 of which 0.06 are faulty. Consequently, the yield of tested ICs is 0.7/0.706 = 0.9915. If we assemble 20 of these devices into a single sub-system and assume that the manufacturing process has a 0.995 yield in its own right, the overall yield will be 0.9915^{20}*0.995 = 0.843*0.995=0.84. Again, assume that we test with a fault coverage of 0.98. So of the 0.161 faulty systems, we find 0.158, leaving 0.0032 faulty systems out of 0.84. Discarding (or reworking elsewhere) the systems that failed the test gives a final yield of 1-0.0032/0.84 = 0.9961. However, assuming no rework, assuming we start with 10^6 ICs we discard $3\text{x}10^5$ after IC test and

$7x10^5*0.158 = 1.1x10^5$ after system test i.e. 41% of the original ICs. **(4)**

iv) *How do IC designers achieve a high value of fault-coverage?*

By designing for testability. FFs linked to combinatorial logic can be replaced by scan FFs that allow test information to be loaded and unloaded. The logic may be such that certain faults may masked by other parts of the logic and designers can add logic to allow these to be weeded out. **(2)**

v) *What sort of things tends to limit fault-coverage?*

Parts of the logic may prove to be difficult to test for particular reasons e.g. inserting scan test FFs compromise timing and cannot be used. Consequently, any faults in associated logic cannot be discovered. **(2)**

b. *A section of combinatorial logic (between flip-flops) is shown in **Figure 3**.*

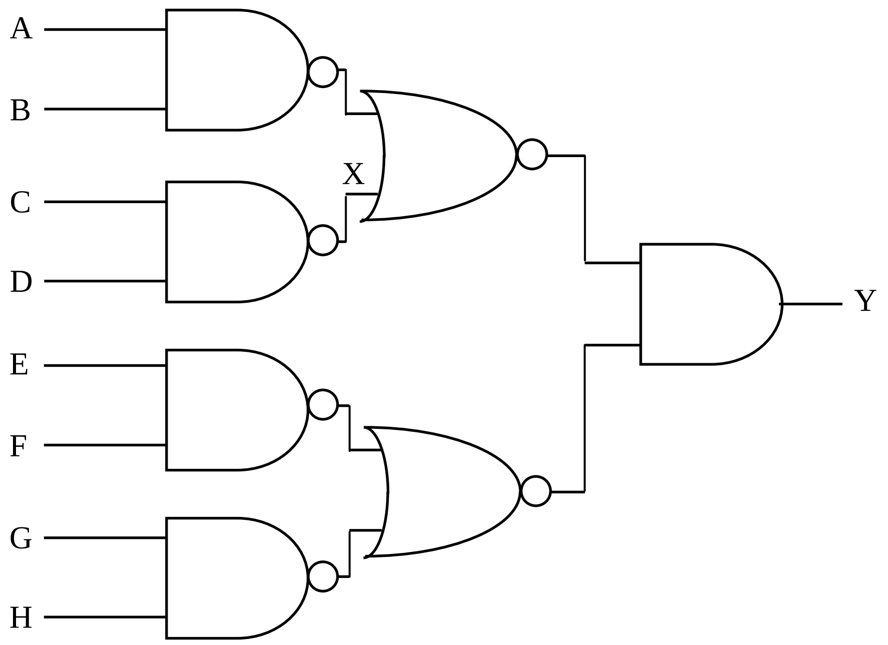


Figure 3: Combinatorial Logic to be Tested

i) *If this logic were to be tested exhaustively then how many individual tests would be required?*

There are 8 inputs and, therefore, 256 combinations of inputs to check. So the number of tests required is 256. **(2)**

ii) *Using a stuck-at fault-modelling approach, how many individual tests would be required?*

There are no internal wires (nodes) in the NAND gates or the NOR gates, but there is an internal node in the AND gate between the NAND and INV making it up. Consequently, the total number of nodes in circuit is 8 inputs, 4 outputs from the NANDs, 2 outputs from the NORs, one internal node in the AND gate and one output node (in fact, it would not be possible to distinguish a fault on the internal node in the AND gate from one on its output. Consequently, the total number of testable nodes is 15. Each node must be tested for stuck and 1 and 0 so there are 30 tests. **(2)**

iii) *In fault-modelling what is the difference between sensitisation and propagation.*

Sensitisation is the setting of inputs required to drive a node being tested to the opposite state to that to which the node is assumed to be stuck. Propagation is the setting of inputs such that the cases where the node is in the state to which it is driven and the node being stuck at the opposite state can be distinguished between. **(2)**

iv) *Identify the input conditions that would be required to test node X for both stuck at 1 and stuck at 0.*

If we are testing for X being stuck at 1 then we must try to drive it to 0. This can be done by setting C and D to 1. To propagate this fault then A and B must also be set to 0 to ensure that the output of the upper NOR gate is $\overline{X}$. Additionally, the output of the lower NOR must be set to 1 to ensure that the output of the AND gate is also $\overline{X}$. This can be achieved by E..H = 1.

If we are testing for X being stuck at 0 then we must try to drive it to 1. This can be done by setting C and D to 0. To propagate this fault is the same as for the stuck at 1 case. **(2)**

4. a. i) *Describe the Synchronous Design Methodology and explain why it results in reliable designs.*

The SDM sets a framework in which synchronous circuit design become feasible. All FFs in a design are deemed to be clocked at the same point in time. The input to a FF, which is derived (possibly via combinational logic) from FF outputs elsewhere in the circuit. This approach means that, due to propagation delays though logic and wiring, the value of the input when a rising clock edge arrives will be the value set at the FF outputs (driving this input) at the last rising clock edge. Furthermore, this means that the values set at a FF output at one clock edge have the time equivalent to one clock cycle to propagate to and become properly set up at a FF input elsewhere in the circuit. The propagation delay along any possible path between two FFs (or FF and input/output) can, therefore, be analysed, against possible variation with PVT, to determine that the data at a FF input will always be valid when the next clock edge arrives. If the relative phase of all clock signals was not controlled in this way then data set at a FF output at one rising clock edge could propagate to another FF and be set up there before the same rising clock edge arrived at this FF. Under a different set of conditions, the same data might not arrive and be seen only at the next rising clock edge. This would mean that the circuit would behave differently under different conditions i.e. it would be unreliable. **(4)**

ii) *How does synchronous design relate to a Register Transfer Level description of circuits?*

An RTL statement within a clocked process (in VHDL for example) might be:

```
Process (clk)
begin
 y <= a + b;
end process;
```

The value y, on the LHS, is stored in a register and is assigned after the rising edge of clk derived from values a and b on the RHS, which in this case pass through an adder (combinatorial logic). a and b will be stored in other registers and the values used to assign y at a given rising edge of clk will be values assigned to a and b at the previous rising edge. **(2)**

b. **i)** *Languages used for hardware design, like VHDL, allow you to specify a delay between an input (on the RHS) in a logical expression changing state and the corresponding output (LHS) changing state. Why is this meaningless in logic synthesis?*

VHDL allows you to make signal assignments with ‘after x ns’ appended to make the output signal change x ns after the input giving rise to the change happens as a way of simulating systems realistically. However, in synthesis, what is important is the functional behaviour of the HDL. There will be delays but these will be the consequence of synthesis and place and route. It is meaningless for a designer to put delays into the HDL code because these are not the delays that the system will experience when placed and routed. **(3)**

ii) *When functionally simulating logic designs, where the delay between inputs to and outputs from logical expressions is not specified, how would the simulator ensure that causality is obeyed?*

For any assignment where no time delay is specified, the simulator still needs some notion of causality to ensure that cause gives rise to effect and not vice versa. Simulators act by maintaining a list of events (i.e. signals changing state) in strict time order (early to late). When simulating, the simulator reads the first, earliest event on the list, which occurs at a particular time, t, and it traces all transitions associated with this time *before* moving on to consider the next event in the list (at a later time). However, if this transition causes an output to change state with no time delay then the simulator assumes that this is an event at time $t+\Delta$ where Δ is an infinitesimal delay that will, ultimately be deemed to be 0. However, this signal change at $t+\Delta$ may, in turn, give rise to another transition and this will be deemed to take place at $t+2\Delta$. The simulator will track all the transitions in strict order (i.e. $t+\Delta$ precedes $t+2\Delta$) until no more transitions are generated. At this point. All of these transitions are deemed to have taken place at time t. Having exhausted time t, the simulator gets the next event from the list – at some later time. In a clocked system, the events will all stem from an initial clock transition. **(3)**

iii) *In this regard, what might happen if a simulator were to encounter the following statement (written in VHDL):*

a <= not a;

The simulator would hang because each transition will always give rise to a transition Δ into the future and so there will be no escaping from the time at which the initial transition occurred. **(2)**

iv) *What is the difference between functional simulation and timing* *simulation?*

Functional simulation is concerned with the behaviour of the HDL code – with reference to a specification, usually. However, there is no guarantee that it will actually behave as intended when implemented – especially given constraints such as clock frequencies. Once the design has been synthesised and place and routed, the actual times associated with the logic gates and interconnect used in the implementation can be abstracted (back annotated) – these will depend on the exact logic used, the loading, and the length of interconnect. A VHDL (or more normally Verilog) netlist can be generated that is an exact match for the implementation and delays, derived from the back annotation, can be added. At this point a relatively accurate timing simulation can be undertaken that takes account of the characteristics of the implementation. This can be a very slow process because the designs being simulated are extremely large. **(2)**

v) *Timing simulation can often be avoided by using static timing analysis. What is static timing analysis and why is it used?*

Rather than doing a full timing simulation, after back annotation the times between a clock signal, the delay to the Q output changing, the delay through the interconnect and logic up to the setup time at the D input of the next flip flop can be generated from every conceivable path through the circuit – regardless of whether that path is actually exercised when the system is running. This gives rise to a set huge list of delays. For a given constraint, i.e. clock frequency, all of these delays can be assessed : is the clock to clock delay implied by a logic path greater than the defined clock period. If such a critical path (or paths) exists then some redesign is needed to meet the specification. Logic delays can also, conceivably, be to small and could result in a D input changing before the hold time has expired. The difference is that the time cost of assembling the list of logic delays is significantly less than the cost of undertaking a full timing simulation and can be used iteratively to fix many of the problems associated with an implementation or optimising the design (buffer sizing) without needing to resort to a full timing simulation – at least until most of the problems have been fixed using static timing analysis. **(4)**

NLS