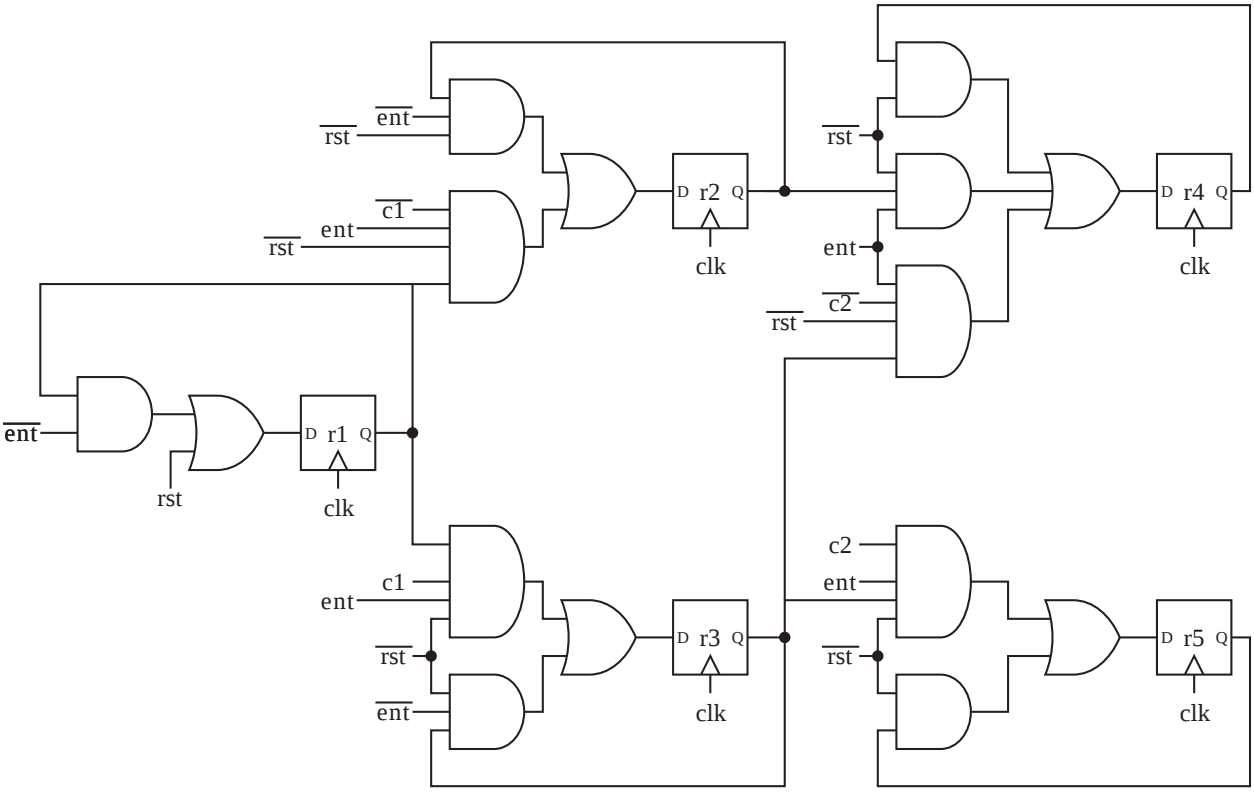
EECS 151/251A Homework 3

Due Monday, Feb 13th, 2023

Problem 1: LUT Mapping

Imagine you have an FPGA consisting of logic blocks each of which contains one 5-LUT and one FF. Partition the following circuit of four inputs $\{ent, rst, c1, c2\}$ and describe the function of each LUT in Boolean expression (you don't have to simplify). The FFs are named $\{r1, r2, r3, r4, r5\}$, and you can use these names to represent their outputs.



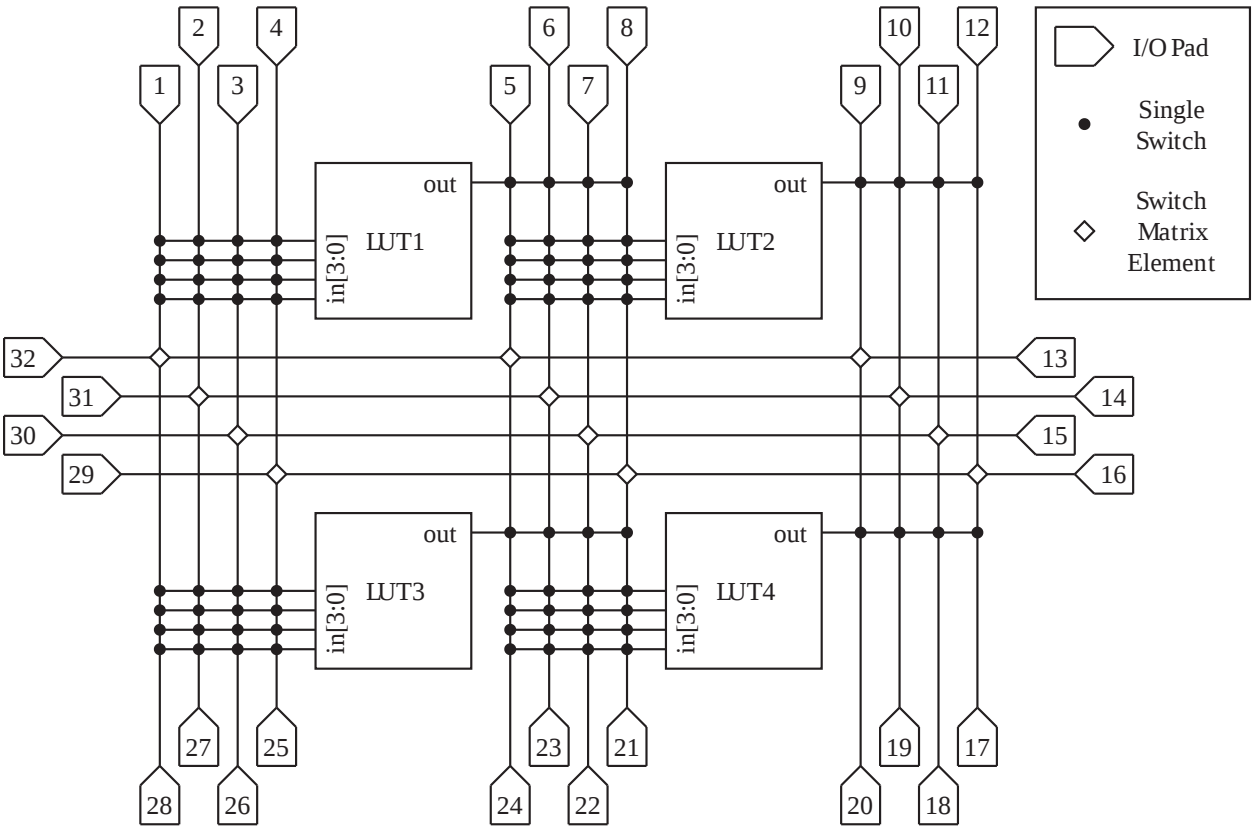
Solution:

We need six logic blocks, one extra for $r4$ besides one per FF.

$$
\begin{aligned}
r1_{next} &= r1 \cdot \overline{ent} + rst \\
r2_{next} &= r1 \cdot ent \cdot \overline{rst} \cdot \overline{c1} + r2 \cdot \overline{ent} \cdot \overline{rst} \\
r3_{next} &= r1 \cdot ent \cdot \overline{rst} \cdot c1 + r3 \cdot \overline{ent} \cdot \overline{rst} \\
tmp &= r2 \cdot ent \cdot \overline{rst} + r3 \cdot ent \cdot \overline{rst} \cdot \overline{c2} \\
r4_{next} &= tmp + r4 \cdot \overline{rst} \\
r5_{next} &= r3 \cdot ent \cdot \overline{rst} \cdot c2 + r5 \cdot \overline{rst}
\end{aligned}
$$

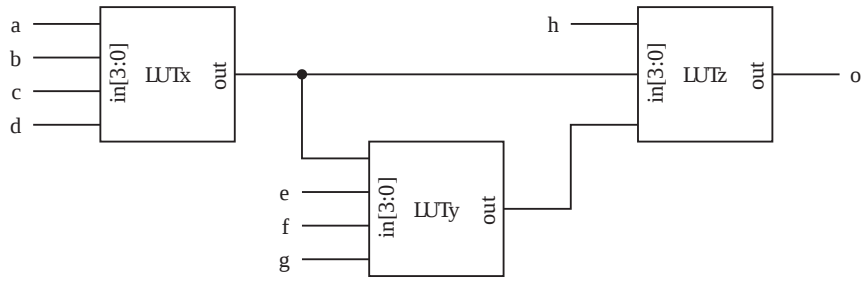
Problem 2: FPGA Placement and Routing

A 2×2 4-LUT FPGA is shown below. In this FPGA, each LUT takes input from one side and generates output to the other side. Switches are implemented as a single transistor that joints or disjoints two wires depending on its configuration bit. Each switch matrix element contains six switches, one for each pair of wire segments. External I/Os are assigned to the I/O pads according to the configuration. Unassigned I/O pads have infinite impedance (no current flows into it).



(a) What is the minimum number of switches needed to connect the output of LUT4 to the input of LUT1. (*Hint*: We need at least three switches to connect the output of LUT1 to the input of LUT4)

(b) *Critical path delay* is the maximum delay between input and output for a given placement and routing. Let delay of each switch be 1 and delay of each LUT be 4. Given a LUT mapping below, when we place LUTx to LUT1, LUTy to LUT2, and LUTz to LUT3, while assigning *a*-*d* to pad 1-4, *e*-*g* to pad 5-8, *h* to pad 25, and *o* to 21, what will be the critical path delay? (*Hint*: delay from *h* to *o* is 6.)



(c) Is there a better placement and routing in terms of critical path delay? If any, write it down and report its critical path delay. Otherwise explain why.

(d) We cannot place LUTx to LUT2 and LUTy to LUT1. Why?

Solution:

(a) Five (one at the output of LUT4, three in switch matrix elements, and one at the input of LUT1).

(b) 21 from *a*-*d* to *o* (12 for total LUT delay, 1 at the input of LUT1, 2 from LUT1 to LUT2, 5 from LUT2 to LUT3, and 1 at the output of LUT3).

(c) We can reduce the critical path delay to 20 by moving LUTz to LUT4 along with *h* to 21 and *o* to 17.

(d) Because if we do so, the middle vertical wire segments (connected to pad 5-8) must be used for *a*-*d*, while the output of LUT1 also needs to use one of them.

Problem 3: Bit-Stream Reverse Engineering

Imagine you obtained a bit-stream from somewhere and want to figure out what it is.

1. The bit-stream is `0x1777A5A5965A9696`. This bitstream is fed in from right to left (i.e. `6` is fed first and `1` is fed last).
2. Every LUT in the FPGA has N inputs (the value of N is part of the mystery). The LUTs are numbered 0, 1, 2, ...
3. Each LUT has an output labeled y_i and inputs labeled x_{i_j}, where i is the LUT number and j is the input number.
4. For programming, the LUTs are connected in a shift register. The bit-stream will be shifted in from LUT0 and then pushed to the subsequent LUTs. (This implies that the left most `1` must be part of the function for LUT0.)

5. The shift register is ordered in the ascending order of input for each LUT. That is, the first register stores the output for an input 00...0, the second register stores the output for an input 00...01 (only x_{i_0} is 1), and so on. (This implies that if $N = 2$, LUT0 programmed with `1` works as an AND gate.)

6. One of the LUTs is programmed as a 2-input gate. Furthermore, it is the only LUT programmed as a 2-input gate.

(a) How many LUTs would the above bit stream program?

(b) Write down the function of each LUT in Boolean expression (no need to simplify but notice there are some patterns that can be concisely expressed with XORs).

Solution:

(a) Four 4-LUTs. Based on the condition that there is only one LUT implementing a 2-input gate, $N = 2$ is invalid because both LUT1 and LUT2 implement 2-input OR gates (`7`), and $N = 3$ is invalid because LUT1 implements a 2-input OR gate (`77`) while LUT2 implements a 2-input XNOR gate (`A5`). For $N = 5$ and $N = 6$ (6 is the maximum because of the length of bit-stream), there are no LUTs implementing 2-input gates. $N = 4$ is the only choice where only LUT1 implements a 2-input gate.

(b)

$$
\begin{aligned}
y_0 &= (x_{0\_0} + x_{0\_1}) \cdot (x_{0\_2} + x_{0\_3}) + x_{0\_0} \cdot x_{0\_1} \\
y_1 &= \overline{x_{1\_0} \oplus x_{1\_2}} \\
y_2 &= \overline{x_{2\_0} \oplus x_{2\_2} \oplus (x_{2\_1} + x_{2\_3})} \\
y_3 &= \overline{x_{3\_0} \oplus x_{3\_1} \oplus x_{3\_2}}
\end{aligned}
$$

251A only — *Optional* Challenge Question for 151

After a while, you realized it takes two M-bit inputs a and b (M is unknown) and calculates sum of a, b, and some constant c (the result may be truncated). What is c?

Solution:

3'b101. The LSB of the result is XOR or XNOR of $a[0]$ and $b[0]$, depending on the first bit of c. Since we have only an XNOR gate (y_1), $c[0]$ is determinted to be 1. Then, the carry-out from the LSB turned out to be $a[0] + b[0]$. Now we have no more 2-input gates, this carry-out should be included in the calculation of the second LSB. After refactoring, y_0 satisfies the condition to be the carry-out of second LSB under $c[1] = 0$. (Bug: we wanted y_2 to be the sum of second LSB but somehow it was negated...) Lastly, y_3 is calculating the next bit with 3-XNOR, which implies $c[2] = 1$.

Problem 4: Functional Completeness Property

Prove that a collection of 2-to-1 multiplexers can implement any Boolean function. Assume constant 0 and 1 are freely available.

Solution:

Using the decomposition law from homework 1 (*Boole's expansion theorem*, shown below), any k-input function can be decomposed into two $(k-1)$-input functions with a multiplexer. We can further decompose these $(k-1)$-iput functions using the same law, and at the end we have only multiplexers and 0-input functions that are constant 0 or 1. Therefore, any function can be implemented only with multiplexers and constants.

Boole's expansion theorem:

$$x(i_0, \ldots, i_k) = \begin{cases} y(i_0, \ldots, i_{k-1}) & \text{if } \; i_k = 0 \\ z(i_0, \ldots, i_{k-1}) & \text{if } \; i_k = 1 \end{cases}$$

Alternatively, we can construct an NAND gate with two multiplexers, which also has the functional completeness property.

Problem 5: Boolean Algebra

Prove the equivalence of Boolean expressions through transformation for each of the following. Show all steps where you used postulates or theorems other than associative and commutative laws. Also write down the postulate or theorem you used for each of those steps. (Answers using truth tables are not acceptable.)

(a) $x + \overline{x}y = x + y$

(b) $xy + \overline{x}z + yz = xy + \overline{x}z$

(c) $\overline{\overline{x} \cdot \overline{y} + z} + z + xy + wz = x + y + z$

Solution:

(a)

$$\begin{aligned} x + \overline{x}y &= x \cdot 1 + \overline{x}y && a = a \cdot 1 \\ &= x(y + \overline{y}) + \overline{x}y && 1 = a + \overline{a} \\ &= xy + x\overline{y} + \overline{x}y && \textit{distributive} \\ &= xy + xy + x\overline{y} + \overline{x}y && a = a + a \\ &= x(y + \overline{y}) + y(x + \overline{x}) && \textit{distributive} \\ &= x \cdot 1 + y \cdot 1 && 1 = a + \overline{a} \\ &= x + y && a = a \cdot 1 \end{aligned}$$

(b)

$$\begin{aligned}
xy + \overline{x}z + yz &= xy + \overline{x}z + yz \cdot 1 && a = a \cdot 1 \\
&= xy + \overline{x}z + yz(x + \overline{x}) && 1 = a + \overline{a} \\
&= xy + xyz + \overline{x}z + \overline{x}zy && \textit{distributive} \\
&= xy \cdot 1 + xyz + \overline{x}z \cdot 1 + \overline{x}zy && a = a \cdot 1 \\
&= xy(1 + z) + \overline{x}z(1 + y) && \textit{distributive} \\
&= xy \cdot 1 + \overline{x}z \cdot 1 && 1 = 1 + a \\
&= xy + \overline{x}z && a = a \cdot 1
\end{aligned}$$

(c)

$$\begin{aligned}
\overline{\overline{x} \cdot \overline{y} + z} + z + xy + wz &= \overline{\overline{x} \cdot \overline{y}} \cdot \overline{z} + z + xy + wz && \overline{a + b} = \overline{a} \cdot \overline{b} \\
&= (\overline{\overline{x}} + \overline{\overline{y}})\overline{z} + z + xy + wz && \overline{a \cdot b} = \overline{a} + \overline{b} \\
&= (x + y)\overline{z} + z + xy + wz && \overline{\overline{a}} = a \\
&= z + x + y + xy + wz && \text{from (a)} \\
&= x \cdot 1 + xy + y + z \cdot 1 + zw + y && a = a \cdot 1 \\
&= x(1 + y) + y + z(1 + w) && \textit{distributive} \\
&= x \cdot 1 + y + z \cdot 1 && 1 = 1 + a \\
&= x + y + z && a = a \cdot 1
\end{aligned}$$

Problem 6: K-Maps

Consider a 4-input function represented by the following truth table.

a	b	c	d	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

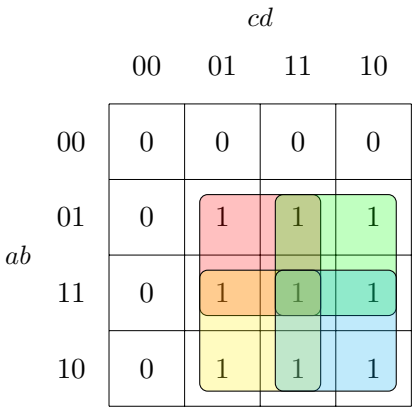
(a) Derive the minimul SOP form using a K-map. Also, how many 2-input gates do you need to

directly implement that SOP? (An N-input AND/OR gate requires $(N-1)$ 2-input AND/OR gates.)

(b) Implement this function using only three 2-input gates.

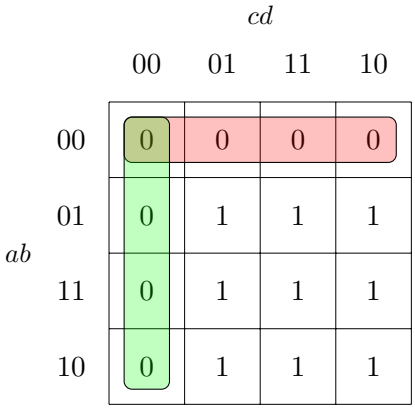
Solution:

(a) Create a K-map and group as many minterms as possible to have the simplest SOP form.



This gives us the SOP expression $x = a \cdot c + a \cdot d + b \cdot c + b \cdot d$. We need four 2-input AND gates and one 4-input OR gate to directly implement this. Converting the 4-input OR gate into three 2-input OR gate, seven 2-input gates are needed.

(b) Create a K-map and group as many **max**terms as possible to have the simplest POS form.



This gives us the POS expression $x = (a + b) \cdot (c + d)$. The direct implementation of this POS uses only two OR gates and one AND gate. Therefore, it requires only three 2-input gates.

Problem 7: K-Maps with Don't-cares

The function of a 2-bit encoder is shown below. - stands for Don't-care. Derive the minimal SOP (in terms of number of products) for each output using a K-map.

a	b	c	d	x	y
0	0	0	1	0	1
0	0	1	0	0	0
0	1	0	0	1	0
1	0	0	0	1	1
others				-	-

Solution:

For x,the minimul SOP is $\overline{c} \cdot \overline{d}$ derived from the following grouping.

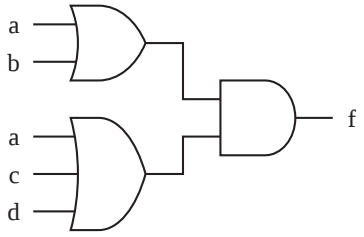
ab \ cd	00	01	11	10
00	-	0	-	0
01	1	-	-	-
11	-	-	-	-
10	1	-	-	-

For y,the minimul SOP is $\overline{b} \cdot \overline{c}$ derived from the following grouping.

ab \ cd	00	01	11	10
00	-	1	-	0
01	0	-	-	-
11	-	-	-	-
10	1	-	-	-

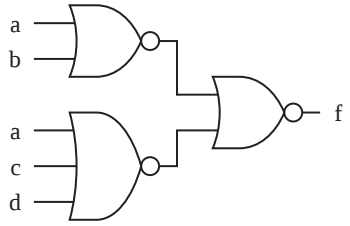
Problem 8: NOR/NAND Network

A NOR network is a gate-level circuit that uses only NOR gates. NAND networks use NAND gates instead. Draw a functionally equivalent NOR network and NAND network for the following circuit. (*Hint*: You may use 1-input NOR/NAND gates as inverters.)



Solution:

NOR network:



NAND network: