CS29204 : Switching Circuits Laboratory

Experiment 4 Lab Report

Group Number: **5**

Members:

Chirag Ghosh (20CS10020)
Anubhav Dhar (20CS30004)
Aritra Mitra (20CS30006)
Shiladitya De (20CS30061)

Problem 1

(≥ 5 comparator)

- Design a comparator to take as input a 4-bit binary number X and output 1 if $X \geq 5$ and 0 otherwise
- Test that it works by applying appropriate inputs and checking the outputs
- Label the terminals to reflect their roles
- Save it as a regular circuit (logic file), reopen and retest
- Save it as a component (cmp file), reopen and retest

Solution

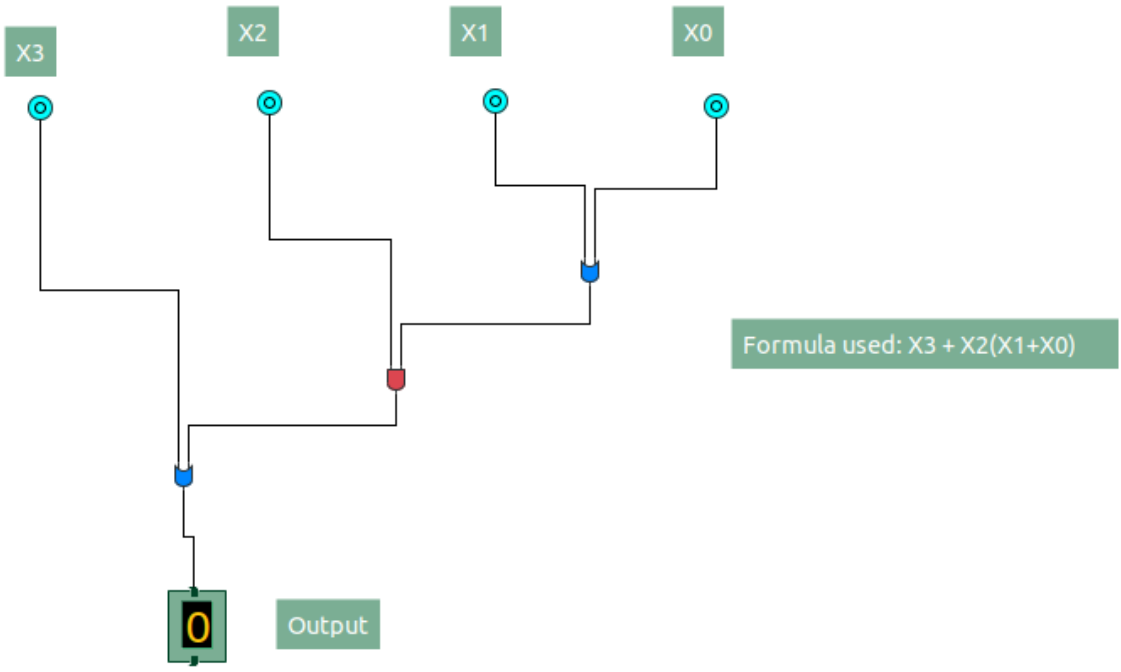
Let $X_3X_2X_1X_0$ be the 4-bit binary input. For $X_3X_2X_1X_0$ to be greater than or equal to 5, we must have one of the two conditions to be true.

- $X_3 = 1$, which implies that the input is greater than or equal to 8, and hence greater than or equal to 5.
- $X_2 = 1$ as well as at least one of X_1 and X_0 is 1. This implies that the number is strictly greater than 4, hence greater than or equal to 5.

Summing up we must have the *or* of the two conditions. Therefore the desired output must be:

$$X_3 + X_2(X_1 + X_0)$$

Circuit Diagram for Problem 1:



Problem 2

(≥ 5 3adder)

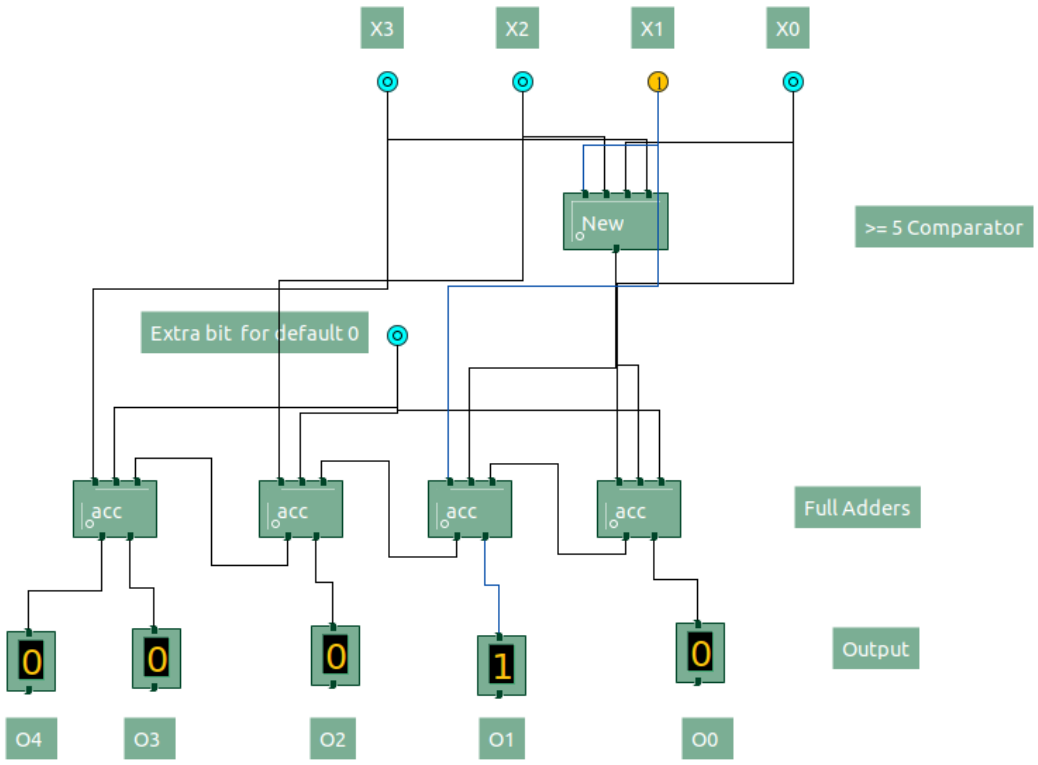
- Design a conditional adder to take as input a 4-bit binary number X and output $X + 3$ if $X \geq 5$ and X otherwise; you may use components designed earlier
- Test that it works by applying appropriate inputs and checking the outputs
- Label the terminals to reflect their roles
- Save it as a regular circuit (logic file), reopen and retest
- Save it as a component (cmp file), reopen and retest

Solution

Let $X_3X_2X_1X_0$ be the 4-bit binary input. We use the ≥ 5 comparator from the Problem 1 to determine wither $X_3X_2X_1X_0$ is greater than or equal to 5. Let q be the output of the comparator(i.e. $q = 1$ if $X_3X_2X_1X_0 \geq 5$, 0 otherwise).

Now, we need to add 3(i.e. 0011) to $X_3X_2X_1X_0$ if $q = 1$, otherwise if $q = 0$ we need to keep the value unchanged(i.e. add 0000). In other words, we are required to add $00qq$ to $X_3X_2X_1X_0$. We use four full adders to add assembled sequentially in a ripple carry adder formation to add $00qq$ to $X_3X_2X_1X_0$ and obtain the desired output.

Circuit Diagram for Problem 2:



Problem 3
(4-bit binary to BCD convertor)

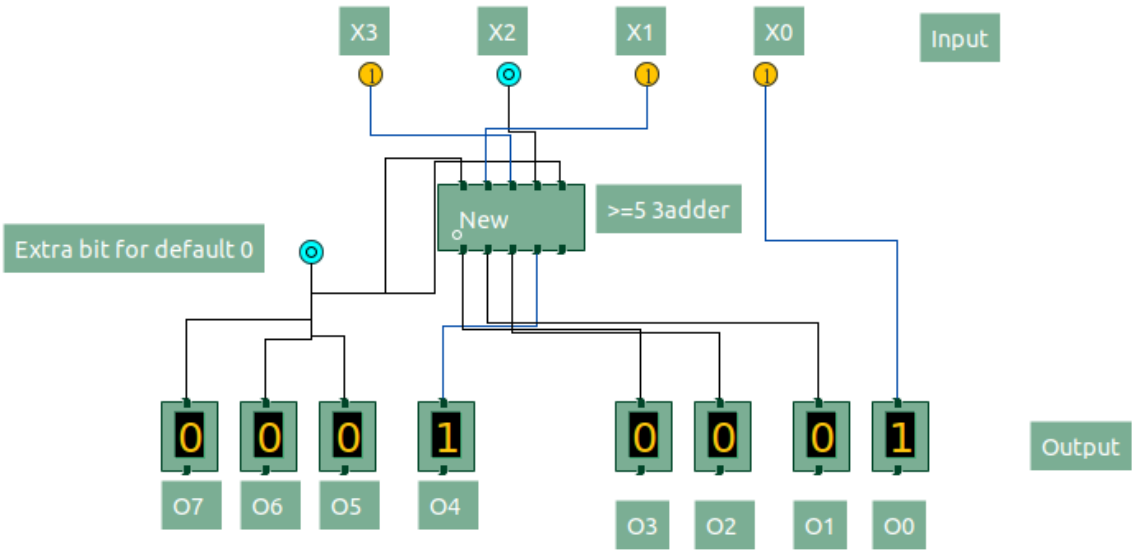
- Using the conditional adder modules, design a combinational 4-bit binary to BCD convertor
- Test that it works by applying appropriate inputs and checking the outputs
- Label the terminals to reflect their roles
- Save it as a regular circuit (logic file), reopen and retest

Solution

To convert a 4-bit binary to a 2-digit BCD number, we use the double-dabble algorithm.

We use a ≥ 5 comparator on the first 3 bits to get whether the number is greater than or equal to 5. If it is less than 5, then the 3 bits are kept as it is, but if the first three bits represent a value greater than or equal to 5, then we add 3 to the number. This entire process is done by a single ≥ 5 3adder. We finally perform one left shift to incorporate the least significant bit of the binary input, completing the conversion to a 2-digit BCD number.

Circuit Diagram for Problem 3:



Problem 4
(6-bit binary to BCD convertor)

- Using the conditional adder modules, design a combinational 6-bit binary to BCD convertor
- Test that it works by applying appropriate inputs and checking the outputs
- Label the terminals to reflect their roles
- Save it as a regular circuit (logic file), reopen and retest

Solution

To convert a 6-bit binary to a 2-digit BCD number, we use the double-dabble algorithm.

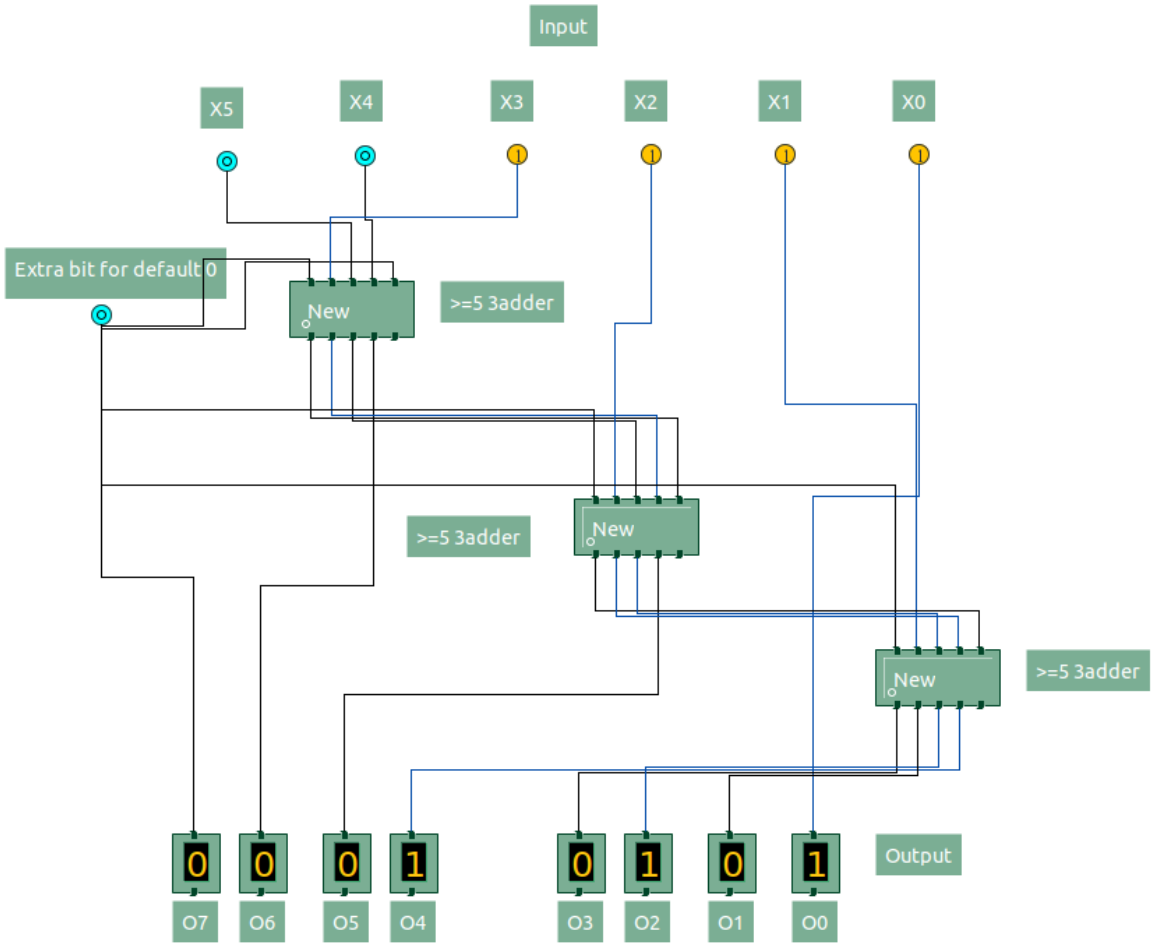
We initially consider the first three bits of the input. If the number formed is greater than 5 then we add 3 to this number (achieved by using the ≥ 5 3adder) followed by a left shift. Other wise we simply left shift the number as it is and proceed on to the next step.

We now again check the least significant digit(nibble) of the BCD formed by the previous left shift. We again use the ≥ 5 3adder to add 3 to the nibble if it has a value greater than or equal to 5 and then we perform the left shift once again.

After the previous left shift, we check for the third time the least significant digit (nibble) of the BCD formed by the previous left shift. We again use the ≥ 5 3adder to add 3 to the nibble if it has a value greater than or equal to 5 and then we perform the final left shift to achieve the final output.

In all three steps checking only the least significant nibble suffices as the most significant nibble can never have a value which exceeds 3 (hence is never greater than or equal to 5) in the intermediate steps of the conversion. Therefore, we can implement a 6-bit binary to 2-digit BCD number with three ≥ 5 3adders.

Circuit Diagram for Problem 5 [P.T.O]:


Input
X5
X4
X3
X2
X1
X0
Extra bit for default 0
New
>=5 3adder
New
>=5 3adder
New
>=5 3adder
0
0
0
1
0
1
0
1
Output
O7
O6
O5
O4
O3
O2
O1
O0

Problem 5
(7-bit binary to BCD convertor)

- Using the conditinal adder modules, design a combinational 7-bit binary to BCD convertor
- Test that it works by applying appropriate inputs and checking the outputs
- Label the terminals to reflect their roles
- Save it as a regular circuit (logic file), reopen and retest

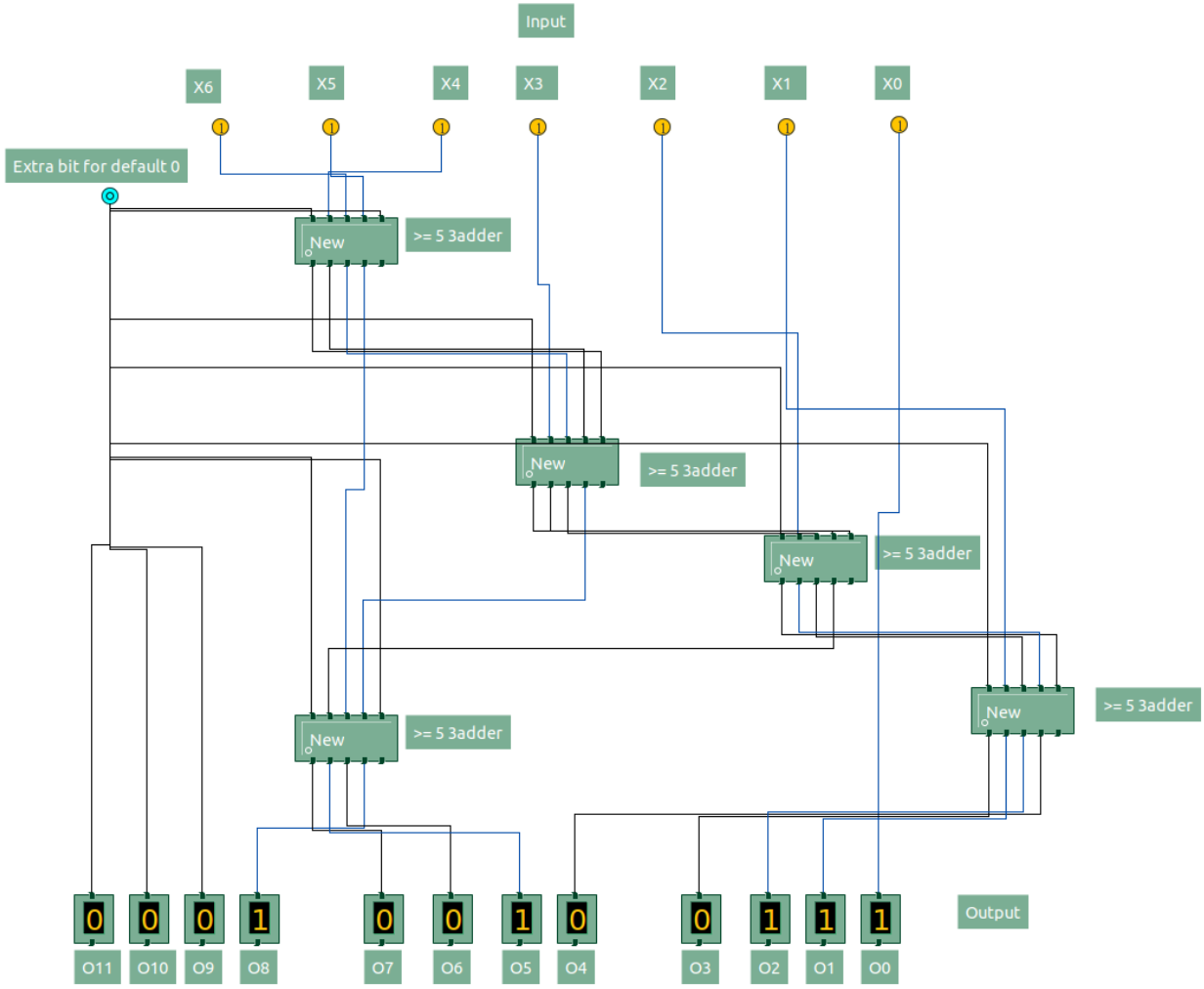
Solution

To convert a 7-bit binary to a 3-digit BCD number, we use the double-dabble algorithm.

For the first 6-bits of the input, the procedure is exactly the same as Problem 4. We use three ≥ 5 3adders for the first 6-bits of the binary input sequentially interleaved with left shifts to complete the first few steps of the double-dabble algorithm.

However, after the last left shift, unlike the last part, now the least significant digit (nibble) and the second least significant digit (nibble) can both have values greater than or equal to 5. Therefore, we would need two more ≥ 5 3adders to check and add 3 to both these nibbles. We perform a final left shift to obtain our final 3-digit BCD output. Hence we used a total of five ≥ 5 3adders.

Circuit Diagram for Problem 5 [P.T.O]:


Input
X6
X5
X4
X3
X2
X1
X0
Extra bit for default 0
New
>= 5 3adder
New
>= 5 3adder
New
>= 5 3adder
New
>= 5 3adder
New
>= 5 3adder
0
0
0
1
0
0
1
0
0
1
1
1
Output
O11
O10
O9
O8
O7
O6
O5
O4
O3
O2
O1
O0