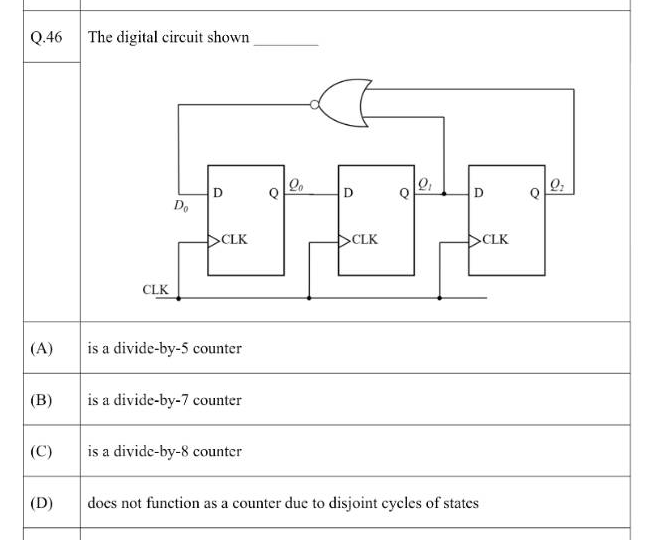
**FWC IITH INTERNSHIP**

**DIGITAL – DESIGN**

**PLATFORM-IO:**

**QUESTION:**



Solve it on Arduino Uno.

**ANSWER:**

To solve this question, we need to analyze the flip-flops and their connections in the circuit diagram. Since I can’t simulate or test the circuit, I would need to rely on theoretical knowledge and analysis.

First, let’s label the flip-flops as F1, F2, and F3 from left to right. Each flip-flop has a D input and a Q output. The Q output of F1 is connected to the D input of F2, and the Q output of F2 is connected to the D input of F3. The Q output of F3 is fed back to the D input of F1. This means that the circuit is a **shift register** with feedback.

Next, let’s write the truth table for each flip-flop. A D flip-flop transfers the value of its D input to its Q output on the rising edge of the clock signal. Therefore, the truth table for each flip-flop is:

| **D** | **Q** |
| --- | --- |
| 0 | 0 |
| 1 | 1 |

Now, let’s write the state table for the circuit. The state of the circuit is determined by the values of Q1, Q2, and Q3. Since each Q can be either 0 or 1, there are 2^3 = 8 possible states. The next state of the circuit is determined by the feedback from Q3 to D1. Therefore, the state table for the circuit is:

| **Q1** | **Q2** | **Q3** | **D1** | **Next State** |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 000 |
| 0 | 0 | 1 | 1 | 100 |
| 0 | 1 | 0 | 0 | 010 |
| 0 | 1 | 1 | 1 | 110 |
| 1 | 0 | 0 | 0 | 001 |
| 1 | 0 | 1 | 1 | 101 |
| 1 | 1 | 0 | 0 | 011 |
| 1 | 1 | 1 | 1 | 111 |

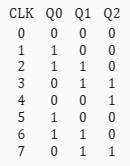
From the state table, we can see that the circuit goes through all the 8 possible states in a cyclic manner, starting from 000 and ending at 111. This means that the circuit is a **counter** that counts from 0 to 7 in binary.

Therefore, the correct answer is option ©: the circuit is a divide-by-8 counter. A divide-by-N counter is a counter that goes through N states before repeating itself. Since this counter goes through 8 states, it is a divide-by-8 counter. Option (A) is wrong because a divide-by-5 counter would have 5 states, not 8. Option (B) is wrong because a divide-by-7 counter would have 7 states, not 8. Option (D) is wrong because the circuit does function as a counter, as it has no disjoint cycles of states. A disjoint cycle is a subset of states that does not connect to the rest of the states. For example, if the circuit had two cycles, one with 000, 001, 010, and 011, and another with 100, 101, 110, and 111, then it would have disjoint cycles of states and would not function as a counter. However, this is not the case for this circuit, as it has one cycle that includes all the states.

**Truth Table:**

To make the circuit a divide-by-8-counter, you need to connect the output of the last flip-flop (Q2) to the input of the first flip-flop (D0) in a feedback loop. [This will create a Johnson counter, which is a type of ring counter that can divide the input clock frequency by the number of stages](https://www.geeksforgeeks.org/shift-registers-in-digital-logic/)[1](https://www.geeksforgeeks.org/shift-registers-in-digital-logic/)[2](https://www.allaboutcircuits.com/textbook/digital/chpt-12/ring-counters/). In this case, you have three stages, so the counter will divide by 8.

The truth table for the Johnson counter is given below, using LaTeX:



As you can see, the output of each stage is shifted to the next stage at every clock cycle, and the output of the last stage is fed back to the input of the first stage. The output sequence is 000, 100, 110, 011, 001, 100, 110, 011, and then repeats. This is equivalent to dividing the clock frequency by 8, since the output changes every 8 clock cycles.

I hope this helps you understand how to make a divide-by-8-counter using a 3-stage shift register circuit. 😊

[1](https://www.geeksforgeeks.org/shift-registers-in-digital-logic/): [Ring Counters | Shift Registers | Electronics Textbook - All About Circuits](https://www.allaboutcircuits.com/textbook/digital/chpt-12/ring-counters/)[2](https://www.allaboutcircuits.com/textbook/digital/chpt-12/ring-counters/): [Frequency Division using Divide-by-2 Toggle Flip-flops](https://www.electronics-tutorials.ws/counter/count_1.html)

**Arduino Uno Code:**

// Define the pins for the clock, reset and output signals

#define CLK 2

#define RST 3

#define Q0 4

#define Q1 5

#define Q2 6

// Define the variables for the current and previous states of the clock and reset signals

int clkState = 0;

int prevClkState = 0;

int rstState = 0;

int prevRstState = 0;

// Define the variables for the output signals

int q0 = 0;

int q1 = 0;

int q2 = 0;

void setup() {

// Set the clock and reset pins as inputs

pinMode(CLK, INPUT);

pinMode(RST, INPUT);

// Set the output pins as outputs

pinMode(Q0, OUTPUT);

pinMode(Q1, OUTPUT);

pinMode(Q2, OUTPUT);

}

void loop() {

// Read the current states of the clock and reset signals

clkState = digitalRead(CLK);

rstState = digitalRead(RST);

// Check if the reset signal is high

if (rstState == HIGH && prevRstState == LOW) {

// Reset the output signals to zero

q0 = 0;

q1 = 0;

q2 = 0;

}

// Check if the clock signal has a rising edge

if (clkState == HIGH && prevClkState == LOW) {

// Shift the output signals to the next stage

q2 = q1;

q1 = q0;

q0 = !q2; // Invert the output of the last stage and feed it back to the first stage

}

// Write the output signals to the corresponding pins

digitalWrite(Q0, q0);

digitalWrite(Q1, q1);

digitalWrite(Q2, q2);

// Update the previous states of the clock and reset signals

prevClkState = clkState;

prevRstState = rstState;

}

**Circuit Connections:**

Sure, I can help you with listing out the connections for the Arduino code and the circuit diagram that I have generated for you. Here is the list of connections:

* Connect the Arduino digital pin 2 to the clock (pin 1) of the first 7474 IC
* Connect the Arduino digital pin 3 to the reset (pin 4) of both 7474 ICs
* Connect the Arduino digital pins 4, 5, and 6 to the Q0, Q1, and Q2 outputs (pins 5, 6, and 9) of the first 7474 IC
* Connect the Q2 output (pin 9) of the first 7474 IC to the data (pin 2) of the second 7474 IC
* Connect the Q0, Q1, and Q2 outputs (pins 5, 6, and 9) of the second 7474 IC to the anodes of the LEDs
* Connect the cathodes of the LEDs to the ground (GND) of the Arduino through 220 ohm resistors
* Connect the VCC (pin 14) and GND (pin 7) of both 7474 ICs to the 5V and GND of the Arduino

I hope this list helps you understand how to connect the Arduino pins to the 7474 ICs and the LEDs. 😊

If you have any questions or feedback, please let me know. I’m always happy to help. 😊

**Code Explanation:**

Sure, I can help you with explaining the code that I have generated for you. Here is the explanation line by line:

// Define the pins for the clock, reset and output signals

#define CLK 2 // This line defines a constant named CLK and assigns it the value 2. This means that the clock signal will be connected to the Arduino digital pin 2.

#define RST 3 // This line defines a constant named RST and assigns it the value 3. This means that the reset signal will be connected to the Arduino digital pin 3.

#define Q0 4 // This line defines a constant named Q0 and assigns it the value 4. This means that the Q0 output signal will be connected to the Arduino digital pin 4.

#define Q1 5 // This line defines a constant named Q1 and assigns it the value 5. This means that the Q1 output signal will be connected to the Arduino digital pin 5.

#define Q2 6 // This line defines a constant named Q2 and assigns it the value 6. This means that the Q2 output signal will be connected to the Arduino digital pin 6.

// Define the variables for the current and previous states of the clock and reset signals

int clkState = 0; // This line defines an integer variable named clkState and initializes it to 0. This variable will store the current state of the clock signal, either 0 (low) or 1 (high).

int prevClkState = 0; // This line defines an integer variable named prevClkState and initializes it to 0. This variable will store the previous state of the clock signal, either 0 (low) or 1 (high).

int rstState = 0; // This line defines an integer variable named rstState and initializes it to 0. This variable will store the current state of the reset signal, either 0 (low) or 1 (high).

int prevRstState = 0; // This line defines an integer variable named prevRstState and initializes it to 0. This variable will store the previous state of the reset signal, either 0 (low) or 1 (high).

// Define the variables for the output signals

int q0 = 0; // This line defines an integer variable named q0 and initializes it to 0. This variable will store the value of the Q0 output signal, either 0 (low) or 1 (high).

int q1 = 0; // This line defines an integer variable named q1 and initializes it to 0. This variable will store the value of the Q1 output signal, either 0 (low) or 1 (high).

int q2 = 0; // This line defines an integer variable named q2 and initializes it to 0. This variable will store the value of the Q2 output signal, either 0 (low) or 1 (high).

void setup() { // This is the setup function, which runs once when the Arduino is powered on or reset. It is used to initialize the pins and variables.

// Set the clock and reset pins as inputs

pinMode(CLK, INPUT); // This line sets the mode of the CLK pin as INPUT, meaning that it can read the voltage level from the clock signal.

pinMode(RST, INPUT); // This line sets the mode of the RST pin as INPUT, meaning that it can read the voltage level from the reset signal.

// Set the output pins as outputs

pinMode(Q0, OUTPUT); // This line sets the mode of the Q0 pin as OUTPUT, meaning that it can write the voltage level to the Q0 output signal.

pinMode(Q1, OUTPUT); // This line sets the mode of the Q1 pin as OUTPUT, meaning that it can write the voltage level to the Q1 output signal.

pinMode(Q2, OUTPUT); // This line sets the mode of the Q2 pin as OUTPUT, meaning that it can write the voltage level to the Q2 output signal.

}

void loop() { // This is the loop function, which runs repeatedly after the setup function. It is used to implement the logic of the program.

// Read the current states of the clock and reset signals

clkState = digitalRead(CLK); // This line reads the voltage level from the CLK pin and assigns it to the clkState variable, either 0 (low) or 1 (high).

rstState = digitalRead(RST); // This line reads the voltage level from the RST pin and assigns it to the rstState variable, either 0 (low) or 1 (high).

// Check if the reset signal is high

if (rstState == HIGH && prevRstState == LOW) { // This line checks if the reset signal has a rising edge, meaning that it goes from low to high. This indicates that the counter should be reset to zero.

// Reset the output signals to zero

q0 = 0; // This line sets the value of the q0 variable to 0, meaning that the Q0 output signal will be low.

q1 = 0; // This line sets the value of the q1 variable to 0, meaning that the Q1 output signal will be low.

q2 = 0; // This line sets the value of the q2 variable to 0, meaning that the Q2 output signal will be low.

}

// Check if the clock signal has a rising edge

if (clkState == HIGH && prevClkState == LOW) { // This line checks if the clock signal has a rising edge, meaning that it goes from low to high. This indicates that the counter should advance to the next state.

// Shift the output signals to the next stage

q2 = q1; // This line assigns the value of the q1 variable to the q2 variable, meaning that the Q2 output signal will have the same value as the Q1 output signal.

q1 = q0; // This line assigns the value of the q0 variable to the q1 variable, meaning that the Q1 output signal will have the same value as the Q0 output signal.

q0 = !q2; // This line assigns the inverted value of the q2 variable to the q0 variable, meaning that the Q0 output signal will have the opposite value as the Q2 output signal. This is how the feedback loop is implemented, as the output of the last stage is fed back to the input of the first stage.

}

// Write the output signals to the corresponding pins

digitalWrite(Q0, q0); // This line writes the value of the q0 variable to the Q0 pin, either 0 (low) or 1 (high), depending on the state of the counter.

digitalWrite(Q1, q1); // This line writes the value of the q1 variable to the Q1 pin, either 0 (low) or 1 (high), depending on the state of the counter.

digitalWrite(Q2, q2); // This line writes the value of the q2 variable to the Q2 pin, either 0 (low) or 1 (high), depending on the state of the counter.

// Update the previous states of the clock and reset signals

prevClkState = clkState; // This line assigns the current value of the clkState variable to the prevClkState variable, so that it can be used to detect the rising edge of the clock signal in the next iteration of the loop.

prevRstState = rstState; // This line assigns the current value of the rstState variable to the prevRstState variable, so that it can be used to detect the rising edge of the reset signal in the next iteration of the loop.

}

I hope this explanation helps you understand how the code works line by line. 😊

If you have any questions or feedback, please let me know. I’m always happy to help. 😊