ANSWERS

A8.8 D-type flip-flops and clocks

Boolean algebra and logic circuits · A level · OCR H446 1.4.3, AQA 7517 4.6.4.1 · about 20 min

What this lesson is about

Clock signals, edge triggering, the D-type flip-flop as one bit of memory, registers and a divide-by-two counter.

- Memory needs feedback
- The clock
- The D-type flip-flop
- Reading a timing diagram
- What flip-flops build

Questions

5 marks in all

1. When does a positive edge-triggered D-type flip-flop change its output Q? [1 mark]

- ☐ **A** At the rising edge of the clock, when Q takes the value of D
- ☐ **B** Whenever D changes
- ☐ **C** Whenever the clock is 1
- ☐ **D** At the falling edge of the clock, when Q becomes NOT D

Answer: A. It samples D at the rising edge and holds that value at every other time.

2. What is a D-type flip-flop used for? [1 mark]

- ☐ **A** Storing one bit
- ☐ **B** Adding two bits
- ☐ **C** Generating the clock signal
- ☐ **D** Inverting a signal

Answer: A. It is a memory unit for one bit; eight side by side with a shared clock make an 8-bit register.

3. What does this program print? [1 mark]

```
clock = [0, 1, 0, 1, 1, 0, 1]
d = [1, 1, 0, 0, 1, 0, 0]
q, last = 0, 0
for i in range(len(clock)):
    if last == 0 and clock[i] == 1:
        q = d[i]
    last = clock[i]
print(q)
```

Answer:

```
0
```

Rising edges are at steps 1, 3 and 6, storing 1, then 0, then 0; the changes in D at steps 4 and 5 happen with no rising edge.

4. A flip-flop has $\overline{Q}$ wired back to D. The clock runs at 1000 Hz. What is the frequency of Q, in Hz? [1 mark]

> **Answer:** 500. Q toggles once per clock cycle, so one full cycle of Q takes two clock cycles: half the frequency.

5. What is the difference between combinational and sequential logic? [1 mark]

☐ **A** Sequential logic's output also depends on what it has stored; combinational logic's depends only on its current inputs

☐ **B** Combinational logic uses a clock and sequential logic does not

☐ **C** Sequential logic uses only NAND gates

☐ **D** There is no difference

> **Answer:** A. Sequential circuits, like flip-flops, use feedback to hold a state; combinational circuits, like adders, do not.

The task: a D-type flip-flop

Write `rising_edge(previous, now)`, which takes two clock values (each 0 or 1) and returns `True` if the clock has just gone from 0 to 1, otherwise `False`. **Part 1.** Simulate a D-type flip-flop over the 14 steps in the lists `CLOCK` and `D`. Q starts at 0 and the clock value before step 0 counts as 0. At each step, if `rising_edge` is true for the previous and current clock values, Q becomes `D` at that step; otherwise Q keeps its value. Record Q after every step, and print all 14 values on one line, separated by spaces, in the form `Q: 0 0 0 ...`. **Part 2.** Build a divide-by-two counter: a flip-flop whose D is always NOT Q. Q starts at 0. Simulate 8 rising edges. After each edge, show Q on the LED (`led(0, 255, 0)` when Q is 1, `led(0, 0, 0)` when it is 0) and `wait(0.25)`. Then print the 8 values of Q on one line, in the form `divider: 1 0 ...`.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

CLOCK = [0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1]
D = [0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1]

def rising_edge(previous, now):
    return False
```

The hint students can ask for: Keep the clock's previous value in a variable so you can spot the moment it goes from 0 to 1. Q only takes the value of D at that moment and holds it at every other step. For the divider, wire NOT Q back into D, so each edge stores the opposite of what was there.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

CLOCK = [0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1]
D = [0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1]

def rising_edge(previous, now):
    return previous == 0 and now == 1

q = 0
previous = 0
trace = []
for step in range(len(CLOCK)):
    if rising_edge(previous, CLOCK[step]):
        q = D[step]
    previous = CLOCK[step]
    trace.append(str(q))
print("Q: " + " ".join(trace))

q = 0
out = []
for edge in range(8):
    q = 1 - q
    out.append(str(q))
    if q == 1:
        led(0, 255, 0)
    else:
        led(0, 0, 0)
```

```
    wait(0.25)
print("divider: " + " ".join(out))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.