Lecture 4

Prepared by: Dr. Alireza Kavianpour

How to define a clock

* For positive edge triggered :
* **IF(clock'EVENT AND clock = '1') THEN**
* **Q <= '1';**
* **----**
* For negative edge triggered :
* **IF(clock'EVENT AND clock = ‘0') THEN**
* **Q <= '1';**
* **----**

VHDL Design for D Flip-Flop

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

```
ENTITY dflipflop IS
PORT(D :IN BIT; clock :IN BIT; Q :OUT BIT);
END dflipflop;
```

```
ARCHITECTURE behavior OF dflipflop IS
BEGIN
    PROCESS(clock)
    BEGIN
        IF(clock'EVENT AND clock = '1') THEN
            IF(D = '1') THEN
                Q <= '1';
            ELSE
                Q <= '0';
            END IF;
        END IF;
    END PROCESS;
END behavior;
```

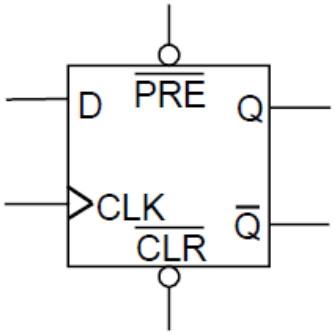
VHDL Alternative Description for D Flip-Flop

* **VHDL Alternative Description for D Flip-Flop Architecture:**
* **ARCHITECTURE behavior OF dflipflop IS**

```
    BEGIN
        PROCESS(clock)
        BEGIN
            IF(clock'EVENT AND clock = '1') THEN
                Q <= D;
            END IF;
        END PROCESS;
    END behavior;
```

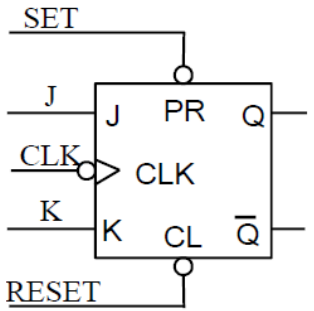
VHDL File for the 74LS74 D Flip-Flop

* **VHDL File for the 74LS74 D Flip-Flop**
* **ENTITY dflipflop IS**
  PORT(D, PRE, CLR, CLOCK :IN BIT; Q,QN :OUT BIT);
  END dflipflop;
* **ARCHITECTURE behavior OF dflipflop IS**
  BEGIN
  PROCESS(CLOCK, CLR, PRE)
  BEGIN
  IF CLR = '0' THEN
  Q <= '0'; QN <= '1';
  ELSIF PRE = '0' THEN
  Q <='1'; QN <= '0';
  ELSIF(clock'EVENT AND clock = '1') THEN
  Q <= D; QN <= NOT D;
  END IF;
  END PROCESS;
  END behavior;



JK Flip Flop

```
ENTITY jk IS
PORT( J, K, PRE, CLR, CLOCK :IN BIT; q, qn :INOUT BIT);
END jk;
ARCHITECTURE behavior OF jk is
BEGIN
PROCESS (CLOCK, CLR, PRE)
BEGIN IF CLR = '0' THEN
q <= '0'; qn <= '1'; ELSIF PRE = '0' THEN q <= '1'; qn <= '0';
ELSIF
if j = '0' and k = '0' then
q <= q; qn <= qn;
elsif j = '1' then
q <= '1'; qn <= '0';
elsif k = '1' then
q <= '0'; qn <= '1';
elsif j= '1' and k = '1' then
q <= qn; qn <= q;
end if;
end if;
end process;
end behavior;
```



JK Flip Flop

```
ENTITY jk IS
PORT( J, K, PRE, CLR, CLOCK :IN BIT; q, qn :INOUT BIT);
END jk;
ARCHITECTURE behavior OF jk is
BEGIN
PROCESS (CLOCK, CLR, PRE)
BEGIN IF CLR = '0' THEN
q <= '0'; qn <= '1'; ELSIF PRE = '0' THEN q <= '1'; qn <= '0';
ELSE
q<= j AND NOT q OR NOT k AND q
 end if;
end process;
end behavior;
```

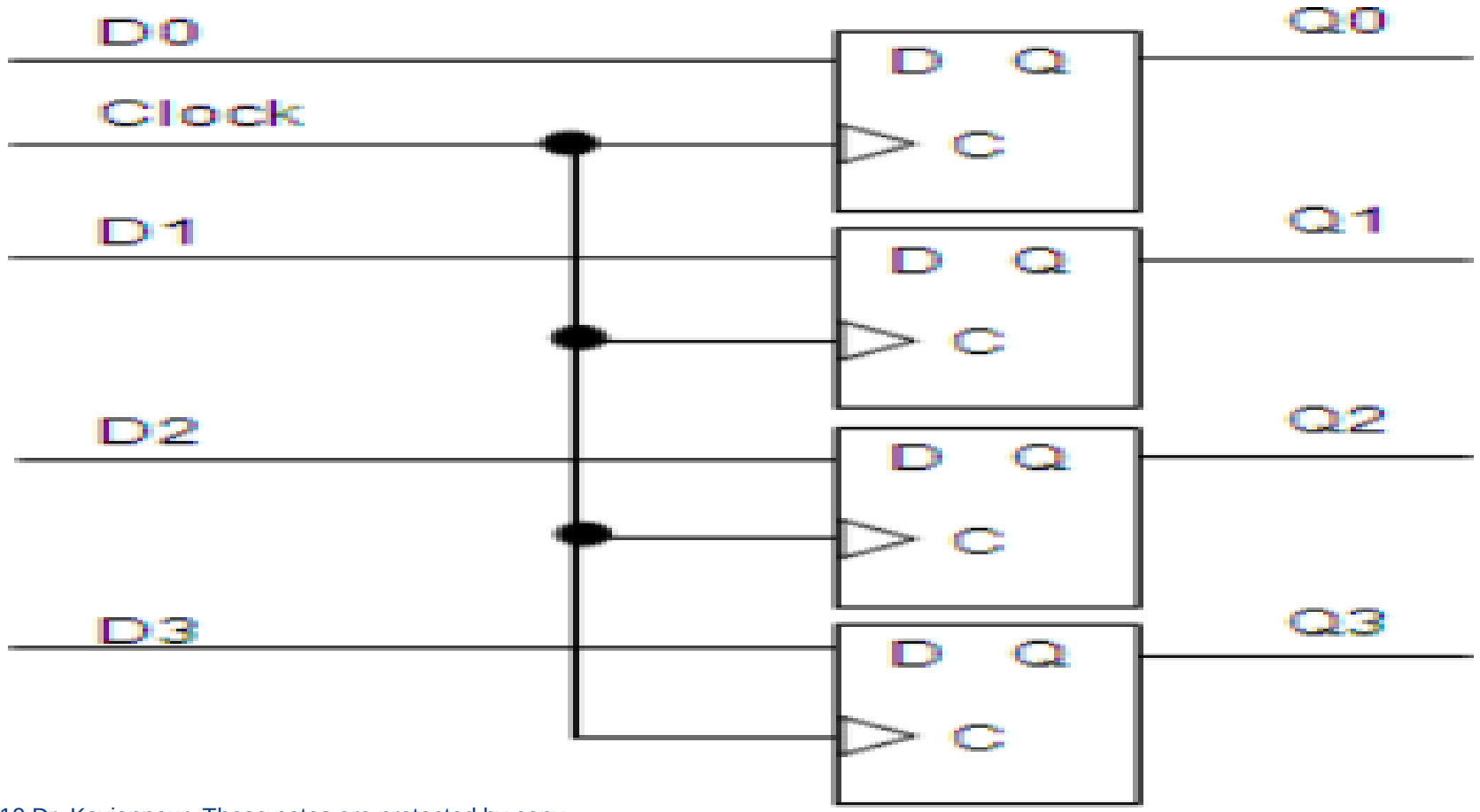
Characteristics Equations for Flip flop

* **SR Flip flop:**
  q<= S+ NOT R AND q
* **JK Flip Flop:**
* **q<= j AND NOT q OR NOT k AND q**
* **T Flip flop:**
  q<= T XOR q
* **D Flip flop:**
  q<= D

Registers

* Registers are nothing more than a group of flip-flops with a common clock.
* Used to store bits that have a relationship to one another.
* The most common use of registers is to store bits of numerical data in computers.
* A 32- or 64-bit machines this simply means that the computers deal with numbers using 32 or 64 bits.
* During computer operations, such as arithmetic, these numbers must be stored.
* This storage utilizes registers; in these cases, either 32 or 64 flip-flops with a common clock.

Registers



Registers

* Common examples of registers inside CPU are:
* PC: program counter
* SP: stack pointer
* CCR: condition code register
* ACC: Accumulator
* MAR: memory address register
* MDR: memory data register

VHDL: 4-Bit Register

```
ENTITY register4 IS
PORT(
        CLK             :IN BIT;
        D               :IN BIT_VECTOR(3 DOWNTO 0);
        Q               :OUT BIT_VECTOR(3 DOWNTO 0));
END register4;

ARCHITECTURE  behavior OF register4 is
BEGIN
PROCESS(CLK)
        BEGIN
                IF(CLK'EVENT AND CLK = '1') THEN
                        Q <= D;
                END IF;
         END PROCESS;
END behavior;
```

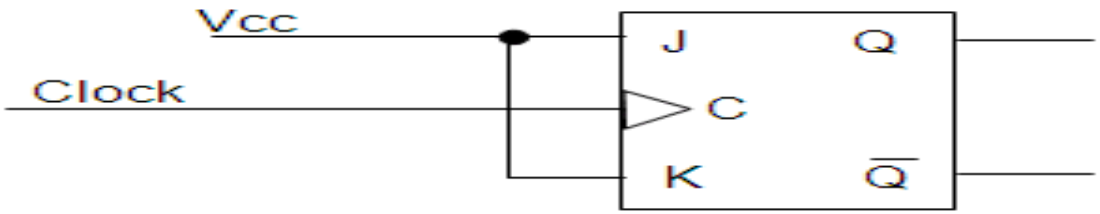
Counter Overview

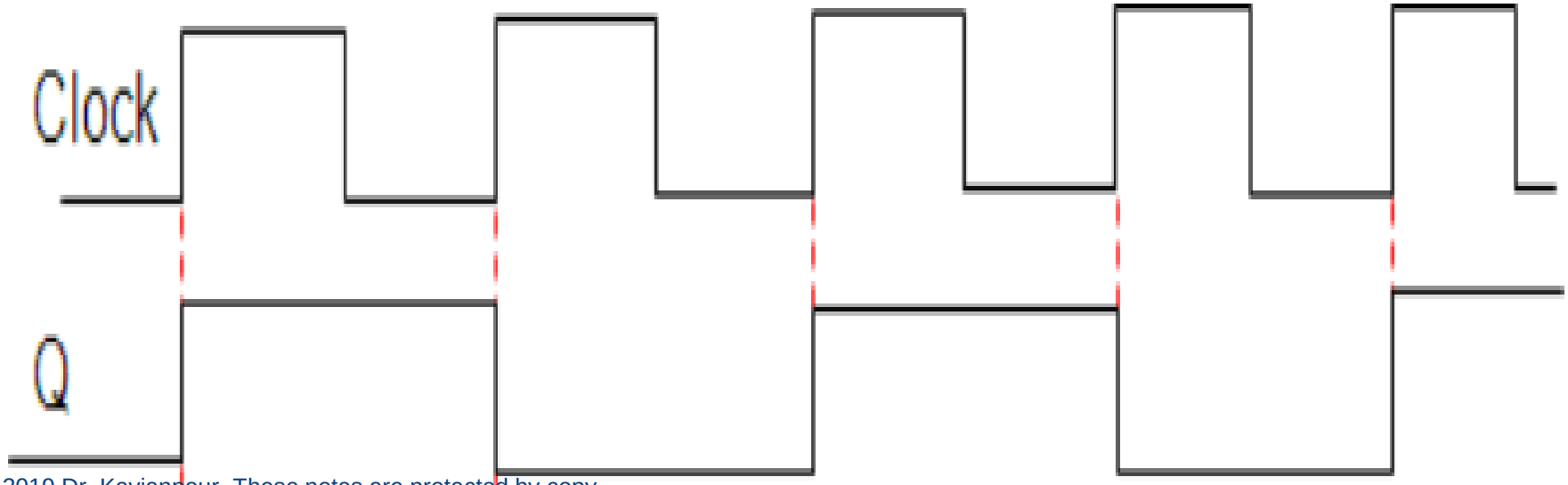
* A common example of a MOD 2^N counter can be found in standard clocks and watches.
* Today, most timekeeping devices are digitally driven with a crystal oscillator running at 32.768 kHz.
* The reason this value is chosen is because this is an inexpensive configuration for the oscillator and because the value is 1 Hz x 2^5. This means we need a 15-bit binary counter to produce a 1-second output.

Counter Overview

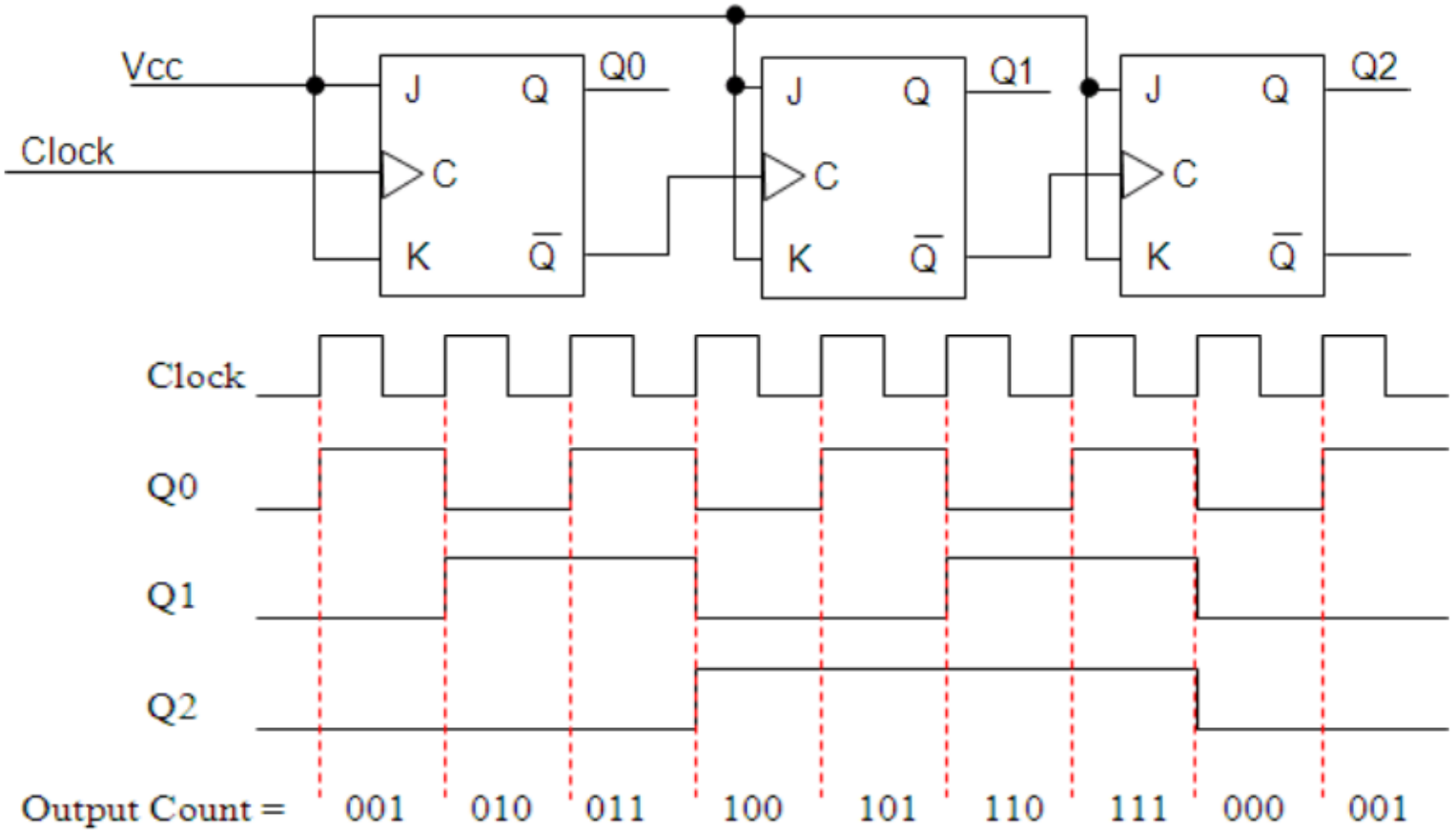
- .
- The clock example points out the limitation of using 2^N counters;
- namely, that the number of seconds in a minute, minutes in an hour, and hours in a day are not 2^N values. AMOD 60 counter for minutes from seconds, a second MOD 60 counter to get hours from minutes, and a MOD 24 counter to get days from hours.
- There is another class of digital counters that do not give sequential counts. One example is a **Gray code counter** .

Ripple Counters

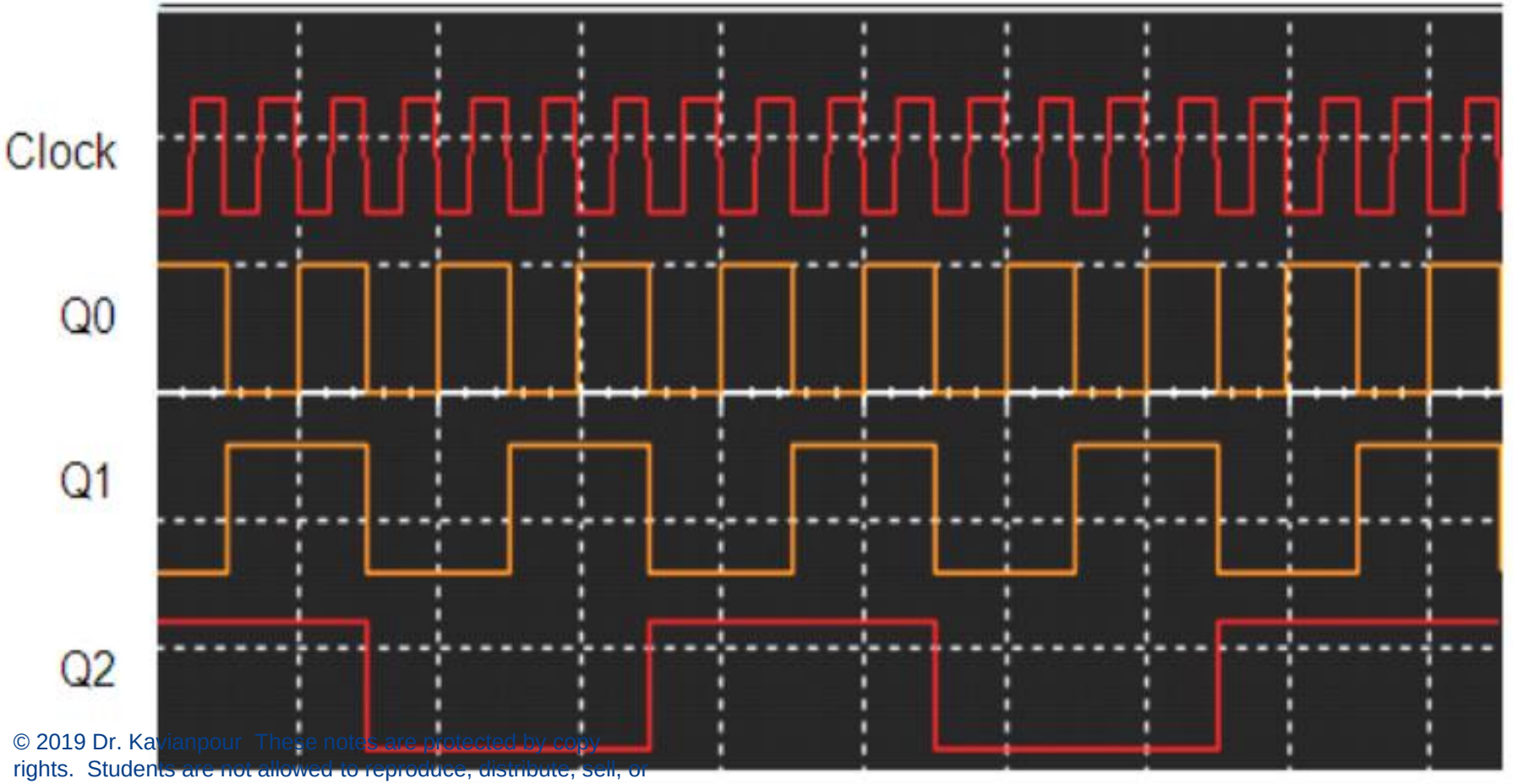




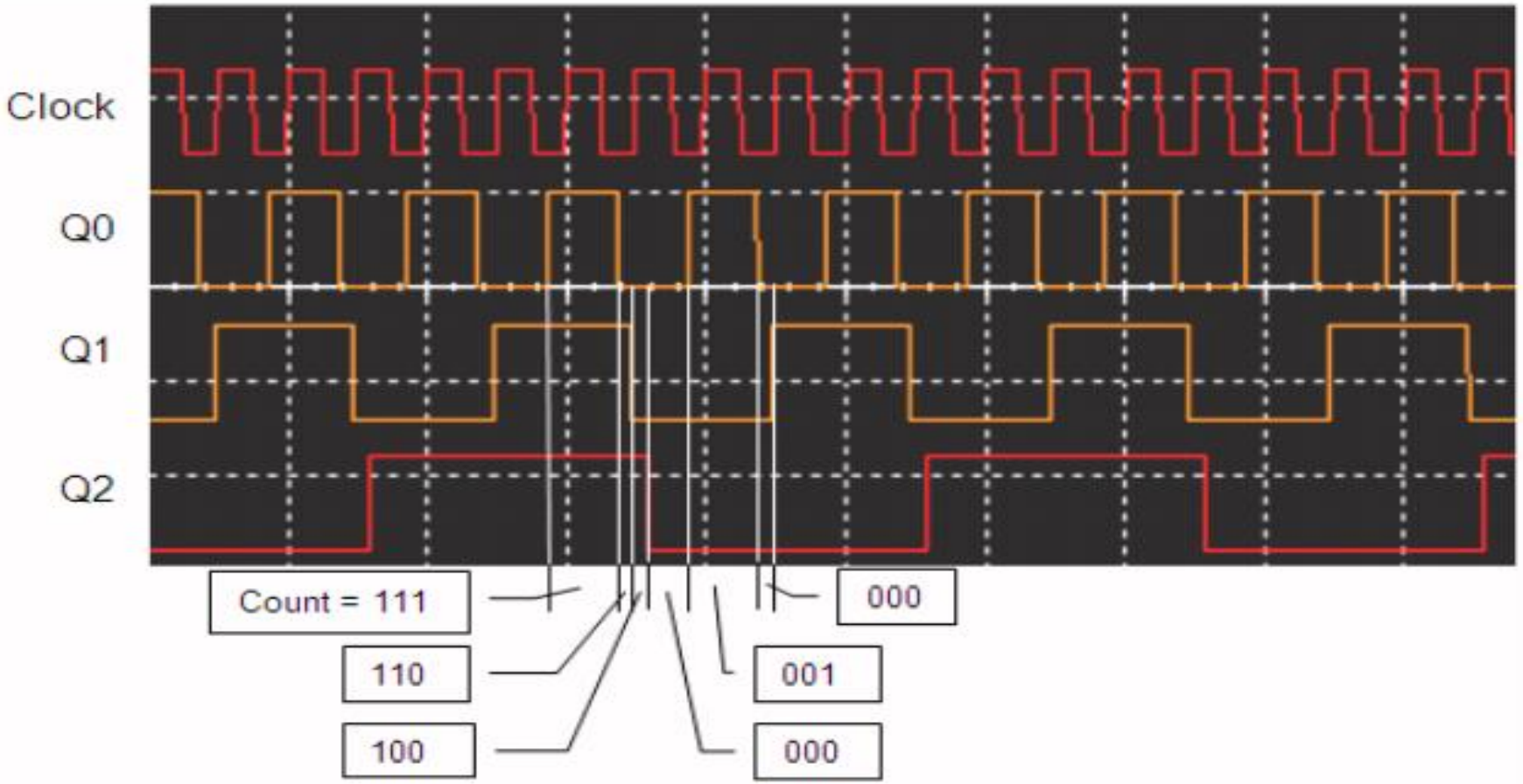
Three-Bit Ripple Counter(Asynchronous)



Ripple Counter at 10 kHz



Ripple Counter at 10 MHz



Ripple Count

* As you can see, the ripple counter does not produce a reliable count sequence at higher frequencies.
* As shown in the simulation, the counter increments from a count of 111 to a count of 000, but has transition counts of 110 and 100.
* In a similar fashion, the counter increments from 001 to 010 with a transition count of 000.
* These transition states at higher frequencies limit the use of ripple counters to applications running at relatively slow speeds, such as time-of-day clocks and watches.

Synchronous Counters

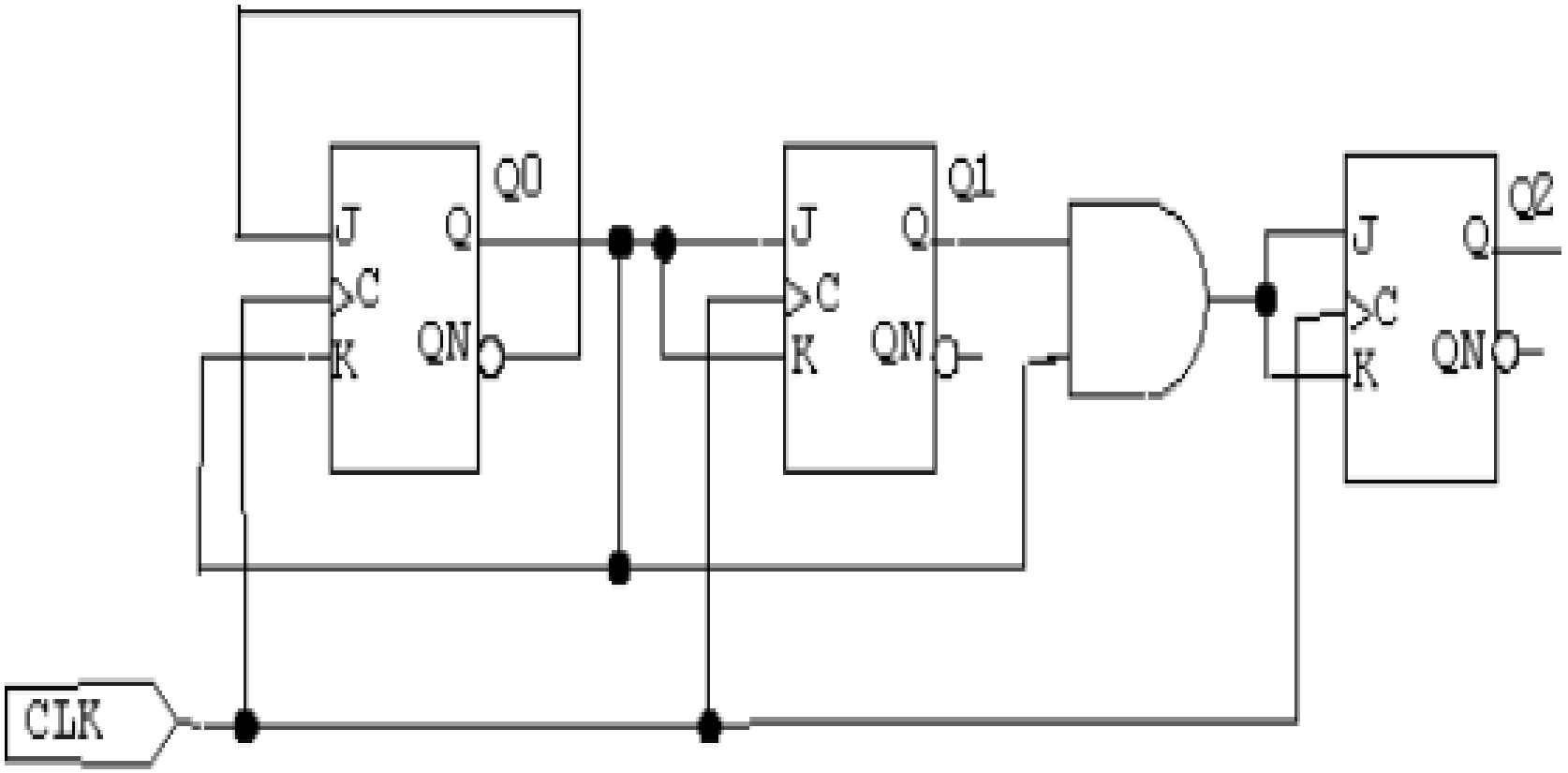
Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

Changes each time

Change when Q0 = 1

Change when Q0 = Q1 = 1

Synchronous Counter



Synchronous Mod-8 Up Counter

```
ENTITY counter3 IS
PORT(
    CLK :IN BIT;
    Q :BUFFER BIT_VECTOR(2 downto 0));
END counter3;

ARCHITECTURE behavior OF counter3 is
BEGIN
    PROCESS(CLK)
        BEGIN
            IF (CLK'EVENT AND CLK = '1') THEN
                IF Q = "000" THEN Q <= "001";
                ELSIF Q = "001" THEN Q <= "010";
                ELSIF Q = "010" THEN Q <= "011";
                ELSIF Q = "011" THEN Q <= "100";
                ELSIF Q = "100" THEN Q <= "101";
                ELSIF Q = "101" THEN Q <= "110";
                ELSIF Q = "110" THEN Q <= "111";
                ELS Q <= "000";
                END IF;
            END IF;
    END PROCESS;
END behavior;
```

VHDL Counter Description Using INTERER Type

```
ENTITY counter3 IS
PORT(
    CLK :IN BIT;
    Q :BUFFER INTEGER RANGE 0 TO 7);
END counter3;
ARCHITECTURE behavior OF counter3 is
BEGIN
PROCESS(CLK)
    BEGIN
        IF (CLK'EVENT AND CLK = '1') THEN
            Q <= Q + 1";
        END IF;
    END PROCESS;
END behavior;
```

MOD-8 Up-Counter With Enable and Clear

```
ENTITY counter8 IS
PORT(
    CLK :IN BIT;
    CLR,EBL :IN BIT;
Q :BUFFER INTEGER RANGE 0 TO 7);
END counter3;

ARCHITECTURE behavior OF counter8 is
BEGIN
PROCESS(CLK,CLR)
    BEGIN
        IF CLR = '0' THEN
            Q <= 0;
        ELSIF EBL = '0' THEN
            Q <= Q;
        ELSIF (CLK'EVENT AND CLK = '1';
            Q <= Q + 1;
        END IF;
    END PROCESS;
END behavior;
```

non-2^N binary counter

* Creation of the non-2^N binary counter requires a test of the output state of the counter, followed by a reset to zero, if necessary. In a MOD-3 counter, for example, Qout takes values 0, 1, 2, 0, 1, 2,..
* The code is written to reset the counter to zero whenever the counter has an output value of 2:

```
Qout <= Qout +1;
IF Qout = 2 THEN
        Qout <= 0;
```

Mod 14 Up Counter

```
ENTITY count4up IS
    PORT(CLK:  IN BIT;
         CLR:  IN BIT;
         QOUT: BUFFER INTEGER RANGE 0 to 15);
      END count4up;
ARCHITECTURE circuit OF count4up IS
      BEGIN
    PROCESS (CLK, CLR)
    BEGIN
         IF (CLR = '0') THEN QOUT <= 0;
         ELSIF (CLK'EVENT and CLK = '1') THEN
         QOUT <= QOUT + 1;
         IF QOUT = 13 THEN
         QOUT <= 0;
         END IF;
    END PROCESS;
      END circuit;
```

Test bench for counter

```
UUT : MOD32COUNTER
      PORT MAP(CLK => CLK, CLR => CLR, EBL =>EBL, DIR => DIR,
        QOUT => QOUT );
        process
        begin
           wait for 20ns;
           if CLK ='0' then
              CLK <= '1';
           else
              CLK <= '0';
        end if;
       end process;
      EBL <= '0' AFTER 600ns, '1' AFTER 900ns;
end Behavioral;
```

MOD-5 Binary Up-Counter With Enable and Clear

```
ENTITY counter5 IS
PORT(
    CLK :IN BIT;
    Q :BUFFER INTEGER RANGE 0 TO 7);
END counter5;

ARCHITECTURE behavior OF counter5 is
BEGIN
PROCESS(CLK,CLR)
    BEGIN
        IF (CLK'EVENT AND CLK = '1') THEN
            IF Q = 4 THEN
                Q <= 0;
            ELSE Q <= Q +1;
            END IF;
    END PROCESS;
END behavior;
```

MOD 5 Binary Up-Counter State Diagram

* Unused states (counts 101, 110, and 111) feed the initial count (000). This is necessary so that if a counting error occurs, such as during power-on or in a noisy environment, the counter returns to the designed counting sequence.
* One traditional method of building this counter would be to design a MOD 8 binary up-counter and use the decode of count 101 to drive the RESET input of the three counter flip-flops.
* This means that the state 101 exists for a very small time, tens of nanoseconds, before resetting the counter back to 000.
* A cleaner method for designing the MOD 5 counter is an approach that only has the five required states. This can be easily accomplished using VHDL.

Cascading Counters

* There are some requirements in which counters must be used in stages. In many digital circuits, multiple clock frequencies are used.
* This is a consequence of the fact that the power consumed by digital devices, particularly CMOS circuits, is directly proportional to the clock speed.
* In a computer, for example, the central processing unit (CPU) usually runs in the GHz range, but communication channels can run in the MHz, or even kHz ranges.
* Consider the arrangement shown below.

Cascaded Counters