ข้อ 1

Design Code

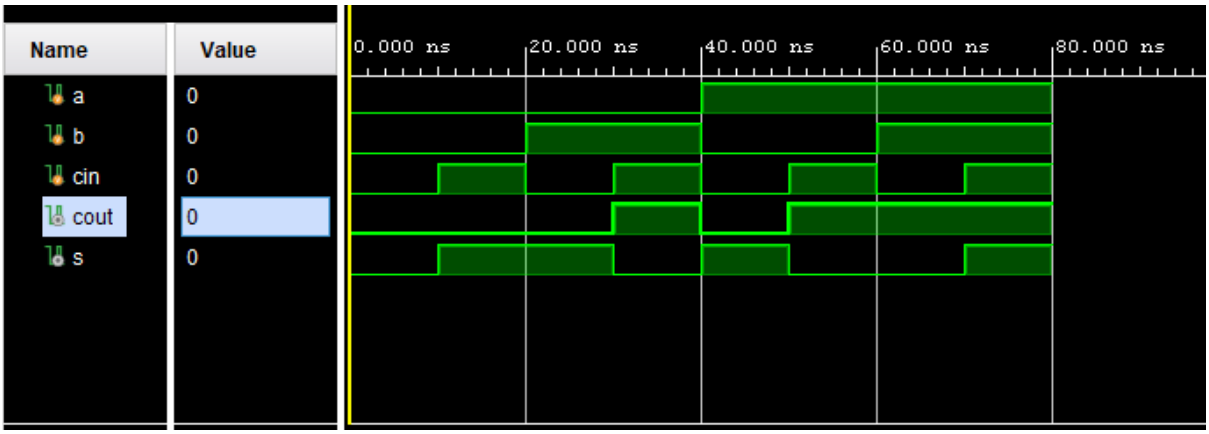
```
module fullAdder(
    output cout,
    output s,
    input a,
    input b,
    input cin
    );
    reg cout,s;
    always @(a or b or cin)
    begin
        {cout,s} = a+b+cin;
    end
endmodule
```

Simulation Code

```
module fullAdderTester(
    );
        reg a,b,cin;
        wire cout,s;
        fullAdder a1(cout,s,a,b,cin);
    initial
    begin
        //$dumpfile("time.dump");
        //$dumpvars(2,a1);
        $monitor("time %t: {%b %b} <- {%d %d %d}", $time,cout,s,a,b,cin);
        #0;
        a = 0 ;  b = 0 ; cin = 0 ;
        #10
        a = 0 ;  b = 0 ; cin = 1 ;
        #10
        a = 0 ;  b = 1 ; cin = 0 ;
        #10
        a = 0 ;  b = 1 ; cin = 1 ;
        #10
        a = 1 ;  b = 0 ; cin = 0 ;
        #10
        a = 1 ;  b = 0 ; cin = 1 ;
```

```
        #10
        a = 1 ;  b = 1 ; cin = 0 ;
        #10
        a = 1 ;  b = 1 ; cin = 1 ;
        #10
        $finish;
    end
endmodule
```

Simulation



ผลลัพธ์ s จะมาจาก a+b+cin แต่ถ้าบวกแล้วเกินจะทดส่วนที่เกินไปใน cout ก็คือเป็นผลรวมตรงตาม Concept ของ Full Adder

ข้อ 2

Design Code

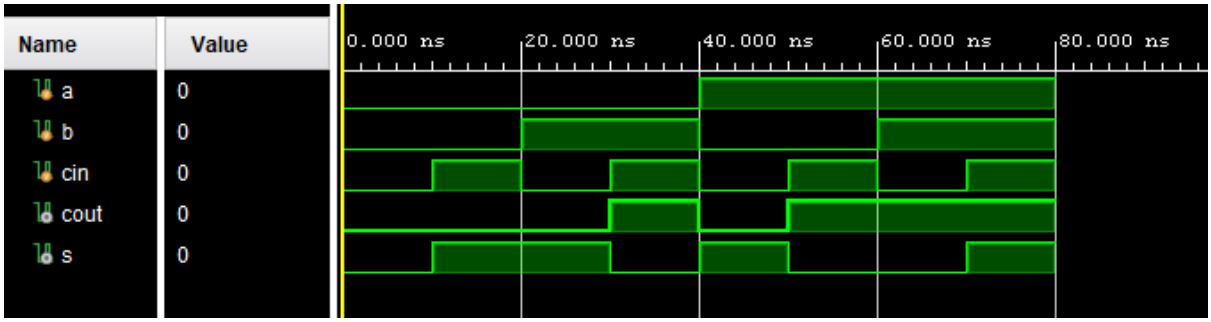
```
module fullAdder2(
    output cout,
    output s,
    input a,
    input b,
    input cin
    );
    assign {cout,s} = a + b + cin;
endmodule
```

Simulation Code

```
module fullAdderTester(
    );
        reg a,b,cin;
        wire cout,s;
```

```
    fullAdder2 a2(cout,s,a,b,cin);
  initial
  begin
    //$dumpfile("time.dump");
    //$dumpvars(2,a1);
    $monitor("time %t: {%b %b} <- {%d %d %d}", $time,cout,s,a,b,cin);
    #0;
    a = 0 ;  b = 0 ; cin = 0 ;
    #10
    a = 0 ;  b = 0 ; cin = 1 ;
    #10
    a = 0 ;  b = 1 ; cin = 0 ;
    #10
    a = 0 ;  b = 1 ; cin = 1 ;
    #10
    a = 1 ;  b = 0 ; cin = 0 ;
    #10
    a = 1 ;  b = 0 ; cin = 1 ;
    #10
    a = 1 ;  b = 1 ; cin = 0 ;
    #10
    a = 1 ;  b = 1 ; cin = 1 ;
    #10
    $finish;
  end
```

Simulation



ผลลัพธ์เช่นเดียวกับข้อ 1 เนื่องจากว่าความหมายเหมือนกันแต่เปลี่ยนแค่วิธีการเขียนเท่านั้น

ข้อ 3

Design Code

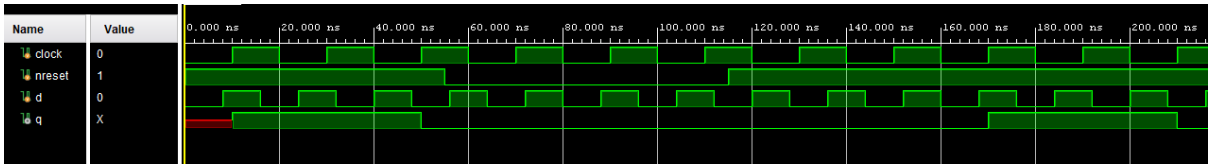
```
module DFlipFlop(
```

```
    output q,
    input clock,
    input nreset,
    input d
    );
    reg q;
    always @(posedge clock or negedge nreset)
        begin
            if (nreset==1)
                q=d;
            else
                q=0;
        end
endmodule
```

Simulation Code

```
module DFlipFlopTester(
    );
    reg clock, nreset, d;
    DFlipFlop D1(q,clock,nreset,d);
    always
        #10 clock=~clock;
    initial
        begin
            //$dumpfile("testDFlipFlop.dump");
            //$dumpvars(1,D1);
            #0
            d = 0 ; clock = 0 ; nreset = 1 ;
            #55
            nreset = 0;
            #10
            d = 0 ; clock = 0 ;
            #50
            nreset=1;
            #1000
            $finish;
        end
             always
                          #8 d=~d;
endmodule
```

Simulation



ผลลัพธ์มีผลเหมือน D-flipflop positive edge clockที่มี Asynchronus reset เมื่อพิจารณา nreset เป็น 0 เมื่อไรผลลัพธ์ q จะเป็น 0 ทันทีโดยไม่ต้องรอ clock ส่วนค่า q=d ต้องมี nreset=1 และเมื่อ clock เปลี่ยนจาก 0 เป็น 1

ข้อ 4

Design Code

```
module ShiftA(q,clock, d);
    output [1:0] q;
    input clock,d;
    reg [1:0] q;
    always @(posedge clock)
    begin
        q[0]=d;
        q[1]=q[0];
    end
endmodule
module ShiftB(q,clock,d);
    output [1:0] q;
    input clock,d;
    reg [1:0] q;
    always @(posedge clock)
    begin
        q[0]<=d;
        q[1]<=q[0];
    end
endmodule
```

Simulation Code

```
module ShiftTester(
    );
    wire[1:0] q1, q2;
    reg clock, d;
    ShiftA s1(q1,clock,d);
    ShiftB s2(q2,clock,d);
    always
```

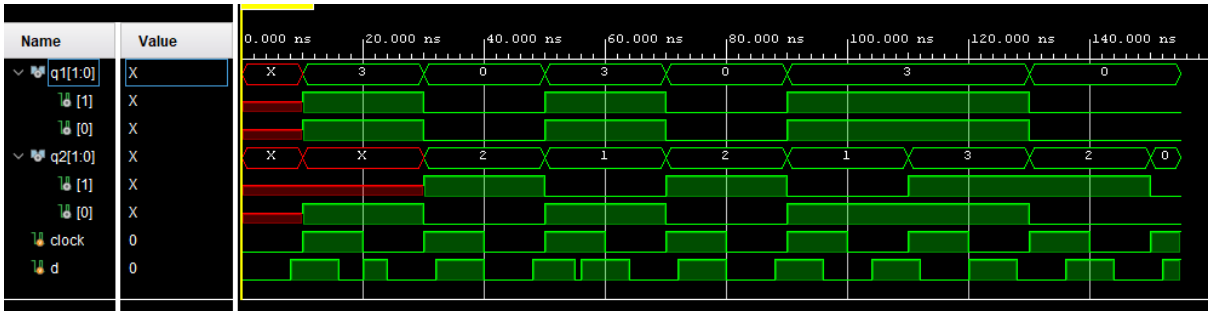
```
        #10
        clock=~clock;
    initial
    begin
        #0
        clock=0;
        d=0;

        #20
        d=1;

        #35
        d=0;

        #100
        $finish;
    end
    always
        #8 d=~d;
endmodule
```

Simulation

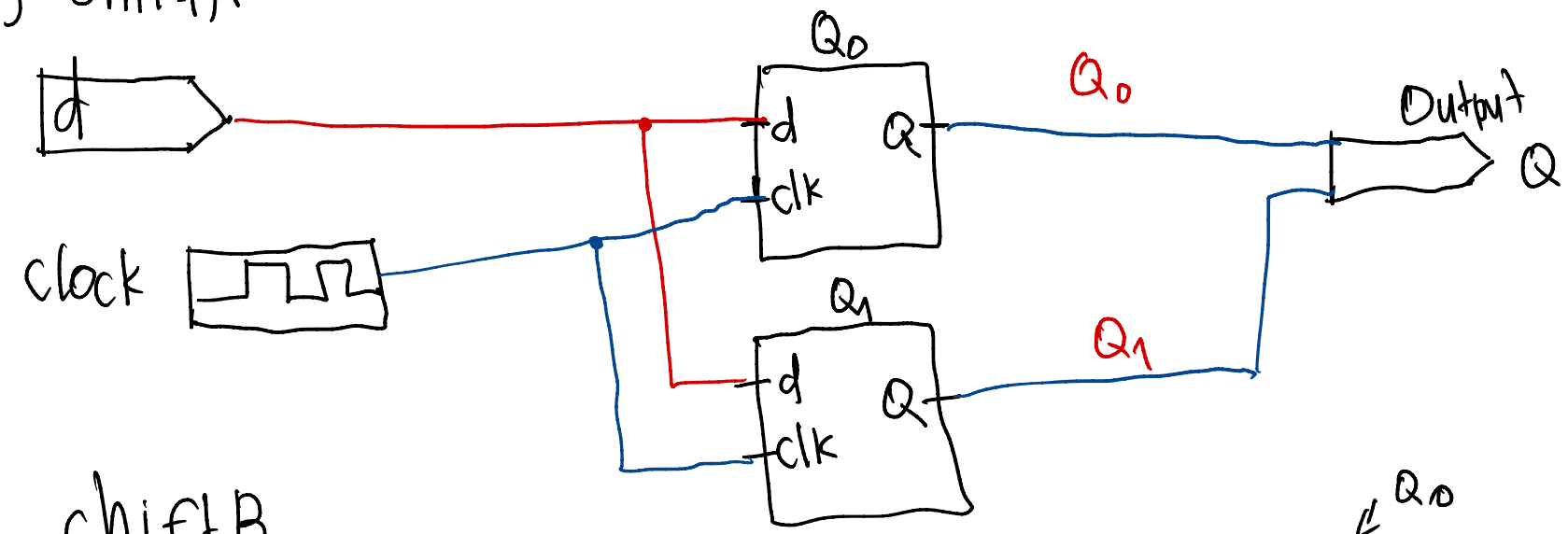


q1 คือ ShiftA q2 คือ ShiftB ส่วน ShiftA เป็นแบบ Blocking ค่าทั้งสองบิตจะเท่าๆกัน คือจะรับค่า d มาตาม clock positive edge ส่วน ShiftB เป็นแบบ Non-Blocking เสมือนการเลื่อนบิตไปตัวถัดไปเรื่อยๆ คือค่าclock ก่อนหน้าของบิต 0 จะถูกส่งต่อไปยังบิต 1 และบิต 0 จะรับค่า d ตาม clock positive edge

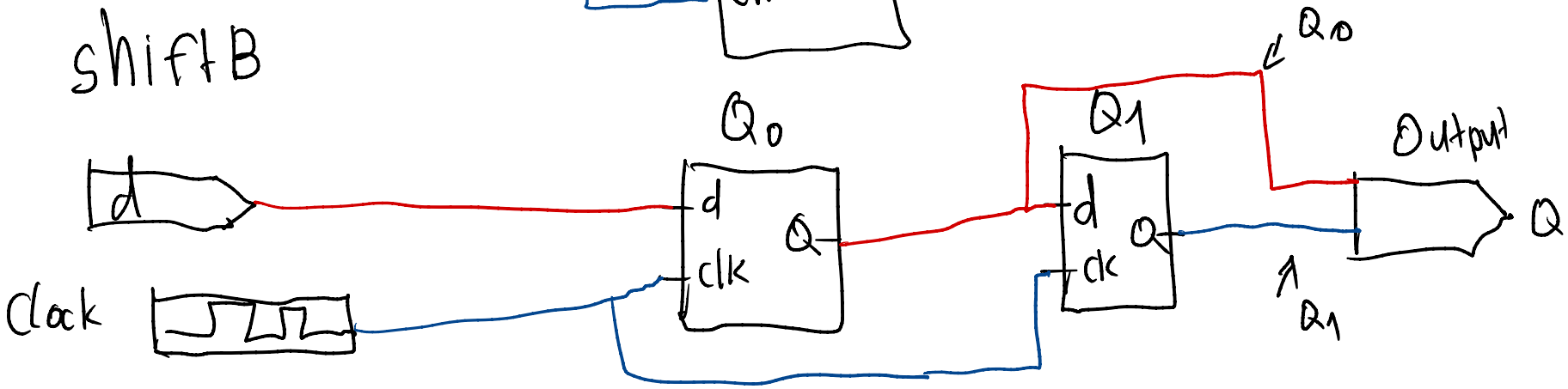
5. Please answer the following questions and submit (in PDF format) to CourseVille on Friday before 23:59 (midnight).
    1. Please draw a schematic representing the logical blocks of both shiftA and shiftB in exercise 4.
    2. What is the difference between blocking and non-blocking assignments?

3. Is it possible to apply parameters to the design in exercise 4 to create shiftRegister with any number of bits? If Yes, please explain how.

1) shiftA



shiftB



2) Blocking แต่ละตัวจะได้รับข้อมูลพร้อมๆกัน

Non-Blocking ข้อมูลที่ใส่ใหม่จะได้รับในตัวแรกเพียงตัวเดียว
ส่วนตัวถัดไปจะได้รับข้อมูลจากตัวก่อนหน้าใน
Clock cycle ก่อนหน้ามา

3)

```
module shiftA(q,clock,d);
output [n:0] q;        [n:0]
input clock,d;

reg [n:0] q;

always @(posedge clock)
begin
      q[0]=d;
      q[1]=q[0];    q[2]=q[1]; ... q[n]=q[n-1];
end
endmodule

module shiftB(q,clock,d);
output [n:0] q;        [n:0]
input clock,d;

reg [n:0] q;

always @(posedge clock)
begin
      q[0]<=d;
      q[1]<=q[0];   q[2]<=q[1]; ... q[n]<=q[n-1]
end
endmodule
```

เมื่อ n คือจำนวนบิต

ทำตามที่แก้โค้ดให้