CME 341: Blocking vs Non-blocking Assignments

Brian Berscheid

Department of Electrical and Computer Engineering
University of Saskatchewan

Today's agenda

1. Verilog assignment operators overview

Verilog assignment operators overview

Verilog assignment operators

- Up to this point in CME 341, we have used the = operator to perform assignments
- In fact, there are two distinct assignment operators in Verilog
  - ▷ The blocking assignment operator (=)
  - ▷ The non-blocking assignment operator (<=)
- The two operators behave differently in only one specific case:
  - ▷ Synchronous procedures with `begin ... end` wrappers
- Mixing up the assignment operators can result in synthesizing unexpected hardware
  - ▷ This is a common source of bugs in CME 341 and beyond!
- Follow the synthesizable coding guidelines presented earlier to avoid this problem!

Blocking vs nonblocking: overview example

Blocking assignments

```
always @ (posedge clk)
  begin
    reg_1 = in_1 ^ in_2;
    reg_2 = reg_1 & in_3;
  end
```

"At each positive clock edge, load `reg_1` with the XOR of `in_1` and `in_2`. Then, compute the AND of that result and `in_3`. Load the result into `reg_2`."

Non-blocking assignments

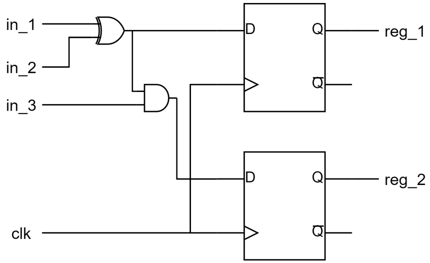
```
always @ (posedge clk)
  begin
    reg_1 <= in_1 ^ in_2;
    reg_2 <= reg_1 & in_3;
  end
```

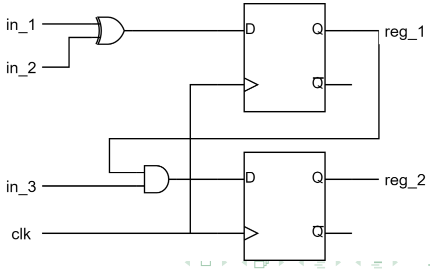
"At each positive clock edge, compute the XOR of `in_1` and `in_2`. Then, compute the AND of the previous value of `reg_1` (prior to the clock arriving) and `in_3`. Assign the results to `reg_1` and `reg_2`."

Blocking vs nonblocking: overview example (synthesized hardware)

```
always @ (posedge clk)
  begin
    reg_1 = in_1 ^ in_2;
    reg_2 = reg_1 & in_3;
  end
```

```
always @ (posedge clk)
  begin
    reg_1 <= in_1 ^ in_2;
    reg_2 <= reg_1 & in_3;
  end
```





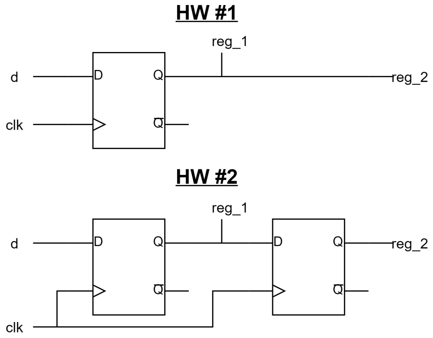
Blocking vs nonblocking assignment operators

- When used within a `begin ... end` block of code:
  - ▷ Blocking operator (=): as soon as an assignment is seen, evaluate the RHS and assign to LHS before moving to next line of code in the block
    - Similarity to a sequential programming language; updates "as-if" they are updated in order listed in code
    - Don't forget that all FFs are updated simultaneously in parallel!
  - ▷ Non-blocking operator (<=): when an assignment is seen, evaluate the RHS but don't assign the result to the LHS until the end of the code block
    - All assignments take effect at end of code block

- For combinational logic, the synthesized hardware will be identical

- For synchronous logic, the synthesized hardware could be different
  - ▷ Specifically, if a LHS is reused in a RHS in the same code block

Order of statements matters in a blocking assignment

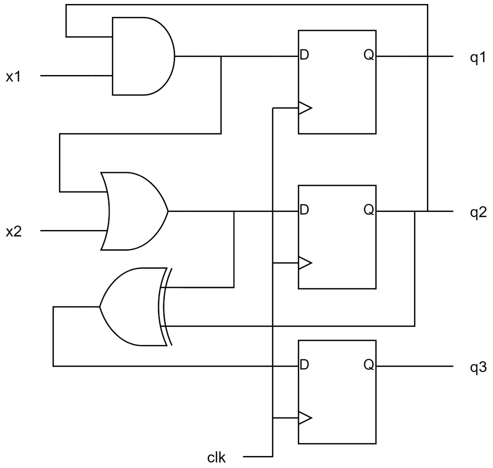


```
always @ (posedge clk)
  begin
    reg_1 = d;
    reg_2 = reg_1;
  end // builds HW #1
```

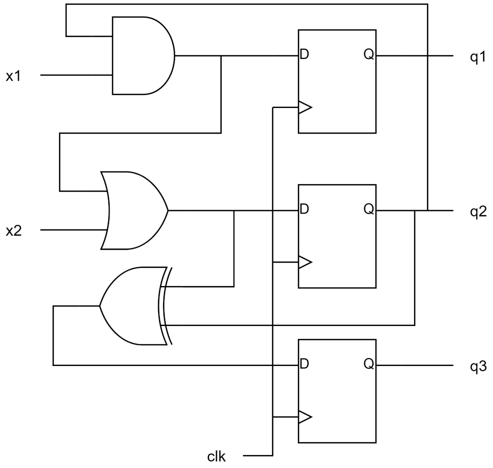
```
always @ (posedge clk)
  begin
    reg_2 = reg_1;
    reg_1 = d;
  end  // builds HW #2
always @ (posedge clk)
  begin
    reg_1 <= d;
    reg_2 <= reg_1;
  end // builds HW #2
always @ (posedge clk)
  begin
    reg_2 <= d;
    reg_1 <= reg_1;
  end // builds HW #2
```

Hardware to code example

Write both blocking and non-blocking code to implement this hardware

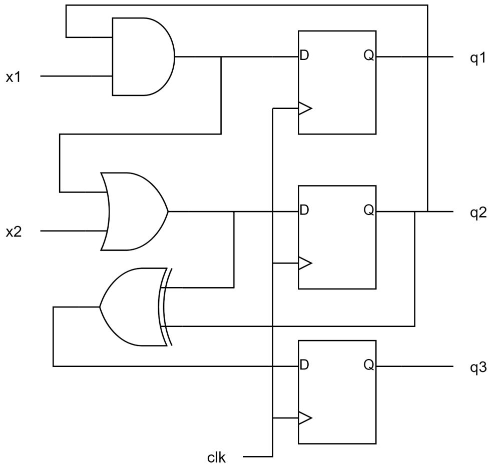


Hardware to code example: non-blocking code



```
always @ (posedge clk)
  begin
    q1 <= x1 & q2;
    q2 <= (x1 & q2) | x2;
    q3 <= ((x1 & q2) | x2) ^ q2;
  end
```

Hardware to code example: blocking code



```
always @ (posedge clk)
  begin
    q1 = x1 & q2;
    temp = q2;
    q2 = q1 | x2;
    q3 = temp ^ q2;
  end
// OR

always @ (posedge clk)
  begin
    q1 = x1 & q2;
    q3 = (q1 | x2) ^ q2
    q2 = q1 | x2;
  end
```

Final thoughts

- Blocking assigments function more like sequential programming (but still build parallel hardware)
- Non-blocking assignments perhaps more accurately reflect to the actual hardware that is being constructed
- More examples are available in the Chapter 3 part 2 notes (handwritten)
- Overall, don't stress too much about this topic. You won't run into problems as long as you follow the CME 341 coding style recommendations:
  - ▷ Avoid `begin ... end` wrappers
  - ▷ Assign one variable per `always` block
- If `begin ... end` is used in (synthesizable code), the non-blocking operator is recommended
  - ▷ Note continuous assignments (`assign a = b`), MUST use the blocking operator.

Thank you!
Have a great day!