

How much testing do I need to do?

Suppose you've built a 64-bit ALU with 4 operations

(Not hypothetical, if you're Intel/AMD/ARM/NVIDIA!)

$$2^{64} \cdot 2^{64} \cdot 4 = 2^{130}$$

How long would it take to test it exhaustively?

$$2^{10} \approx 10^3$$

$$\approx 10^{39} \text{ tests}$$

$$\approx 10^{30} \text{ seconds}$$

Besides exhaustive testing, how can you gain confidence it works?

ES 4: Sequential logic in VHDL

Steven Bell

17 October 2023

Exams are graded

Median was 40.5 (of 47)

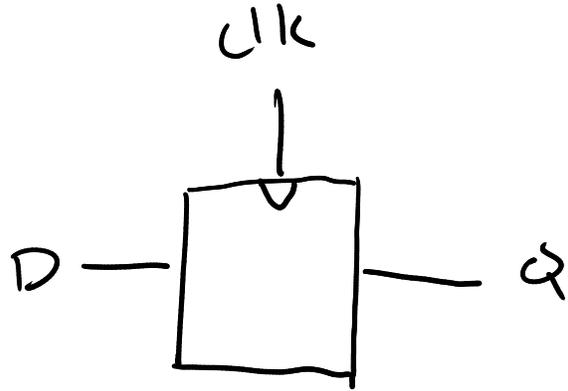
If you got below ~30, I'd like to chat and understand where you're at with the course.

By the end of class today, you should be able to:

- Create flip-flops using VHDL
- Describe the structure of a counter or shift register
- Build counters and other simple sequential circuits using VHDL

Warmup

Write a concise description of the signals (inputs/outputs) and behavior of a D flip-flop.

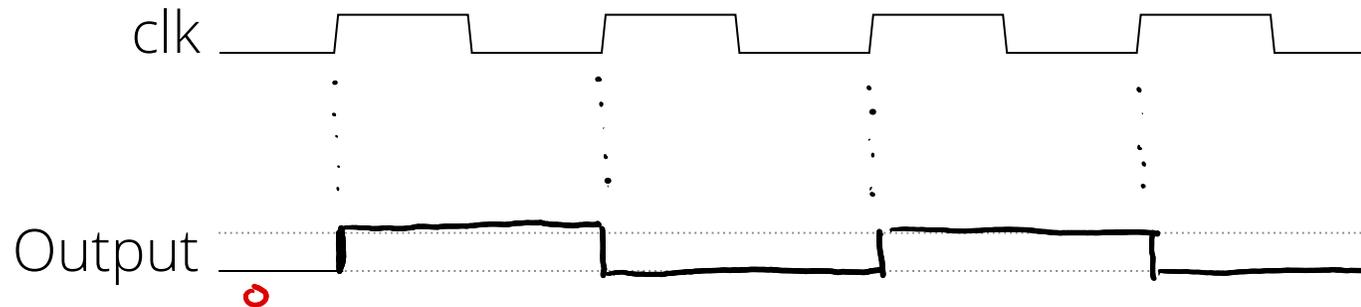
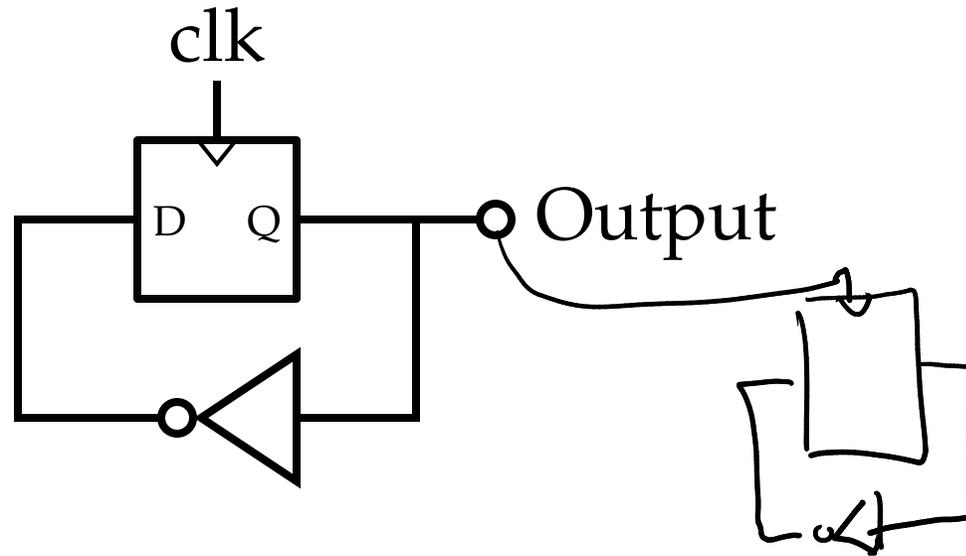


Q gets D when (and only when) clk goes 0 to 1

Warmup 2

Draw the output of this circuit as a function of time.

Assume the output is 0 (low) at time = 0. Ignore propagation delay.



Definition: register

A "register" is just a bank of flip-flops.

Unless otherwise specified, we mean D flip-flops (possibly with a reset)

Process sensitivity

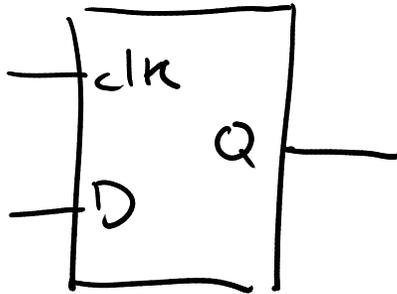
```
process (SENSITIVITY)
```

```
begin
```

```
end process;
```

Modeling a D flip-flop

```
entity dff is
  port(
    clk : in std_logic;
    d : in std_logic;
    q : out std_logic
  );
end;
```



Modeling a D flip-flop

```
entity dff is
  port(
    clk : in std_logic;
    d : in std_logic;
    q : out std_logic
  );
end;
```

```
architecture synth of dff is
begin

end;
```

Modeling a D flip-flop

```
entity dff is
  port(
    clk : in std_logic;
    d : in std_logic;
    q : out std_logic
  );
end;
```

```
architecture synth of dff is
begin
    process
    process ???
    begin
        q <= d;
    end process;
end;
```

Modeling a D flip-flop

```
entity dff is
  port(
    clk : in std_logic;
    d : in std_logic;
    q : out std_logic
  );
end;
```

```
architecture synth of dff is
begin

  process (clk)
  begin
    if rising_edge(clk) then
      q <= d;
    end if;
  end process;

end;
```

Does using a process block then immediately mean we are creating a sequential circuit?

And if so does that make all test benches a sequential circuit?

Does using a process block then immediately mean we are creating a sequential circuit?

And if so does that make all test benches a sequential circuit?

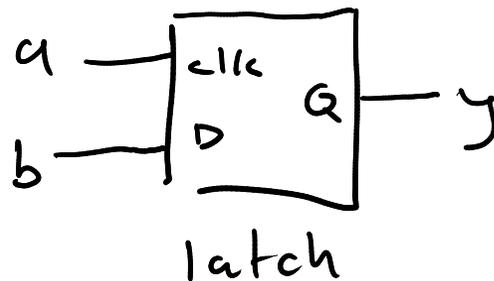
all

It depends on the sensitivity list!

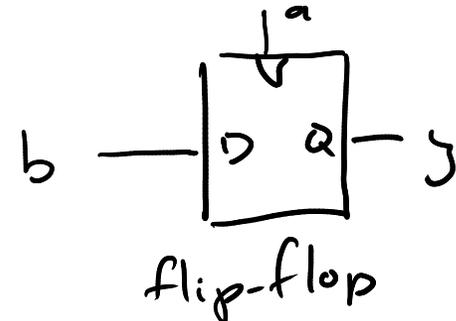
```
process (all b)
begin
  y <= b;
end process;
```



```
process (a, b)
begin
  if a = '1' then
    y <= b;
  end if;
end process;
```



```
process (a)
begin
  if rising_edge(a)
    y <= b;
  end if;
end process;
```



Does using a process block then immediately mean we are creating a sequential circuit?

And if so does that make all test benches a sequential circuit?

It depends on the sensitivity list!

```
process (a, b)  
begin  
  y <= b;  
end process;
```

```
process (a, b)  
begin  
  if a = '1' then  
    y <= b;  
  end if;  
end process;
```

```
process (a)  
begin  
  if rising_edge(a)  
    y <= b;  
  end if;  
end process;
```

Process sensitivity

```
process (SENSITIVITY)
```

```
begin
```

```
end process;
```

If sensitivity includes:

`all` \updownarrow \longrightarrow Combinational logic

`clk` \uparrow \longrightarrow Flip-flop / register

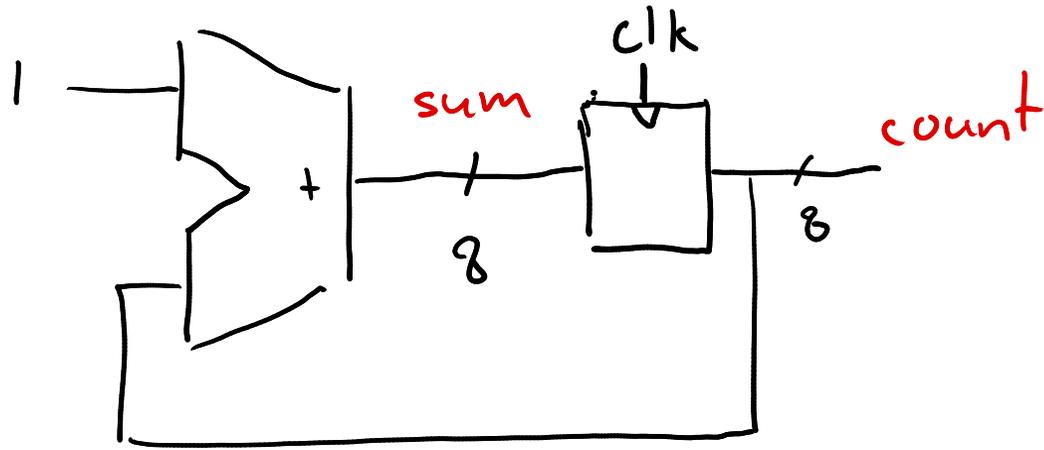
`clk` \uparrow + `data` \updownarrow \longrightarrow Latch

Nothing \longrightarrow Testbench (continuous evaluation)

Something else \longrightarrow Bad things you probably didn't want.

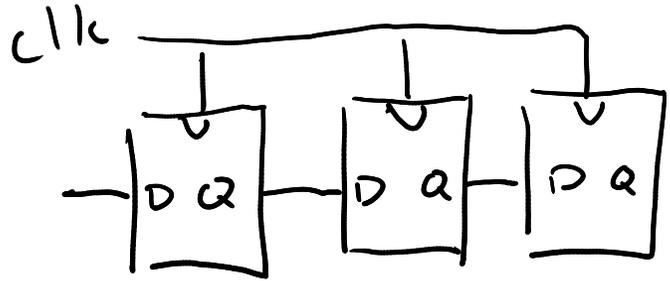
Counter in VHDL

vhdlweb.com/problem/synth_playground



Shift register

On each clock cycle, shift the input into the lowest register



Reset signals!

For Thursday

1. No new reading!!
2. Continue with lab 5, report due next week

Bonus material

Blocking vs non-blocking

Real hardware is always non-blocking!

So signals/outputs must be assigned with non-blocking assignments.

Intermediate variables can have blocking assignments.

```
process(all) is
  variable ab : std_logic;
  variable notbc : std_logic;
begin
  ab := a and b;
  notbc := (not b) and c;
  y <= ab or notbc;
end
```