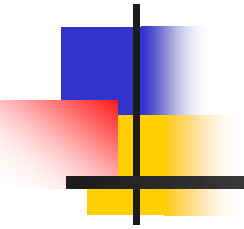
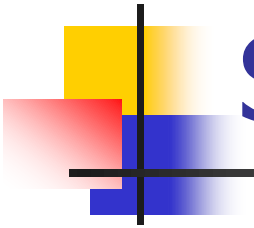


Digital Design: An Embedded Systems Approach Using Verilog

Chapter 4
Sequential Basics

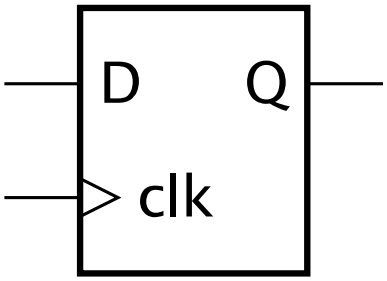


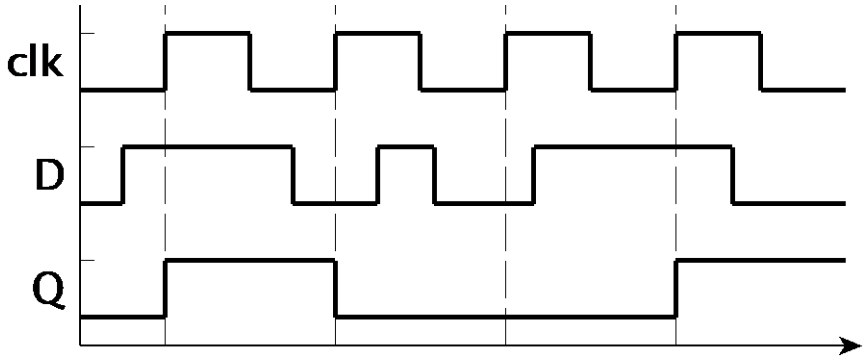
Sequential Basics

- Sequential circuits
  - Outputs depend on current inputs and previous inputs
  - Store *state*: an abstraction of the history of inputs
- Usually governed by a periodic clock signal

D-Flipflops

- 1-bit storage element
  - We will treat it as a basic component





- Other kinds of flipflops
  - SR (set/reset), JK, T (toggle)

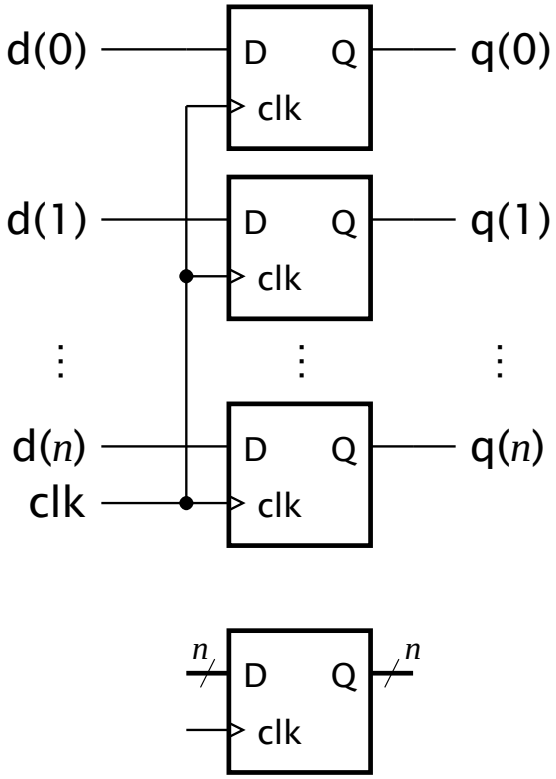
Registers

- Store a multi-bit encoded value
  - One D-flipflop per bit
  - Stores a new value on each clock cycle

```
wire [n:0] d;
reg  [n:0] q;
...

always @(posedge clk)
  q <= d;
```

event list

nonblocking asignment

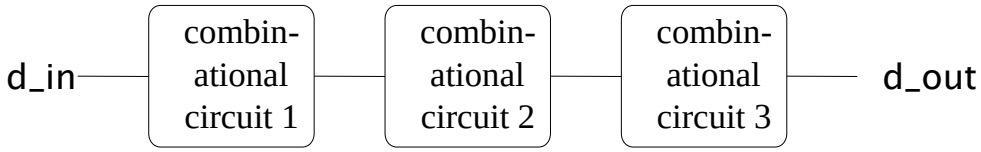


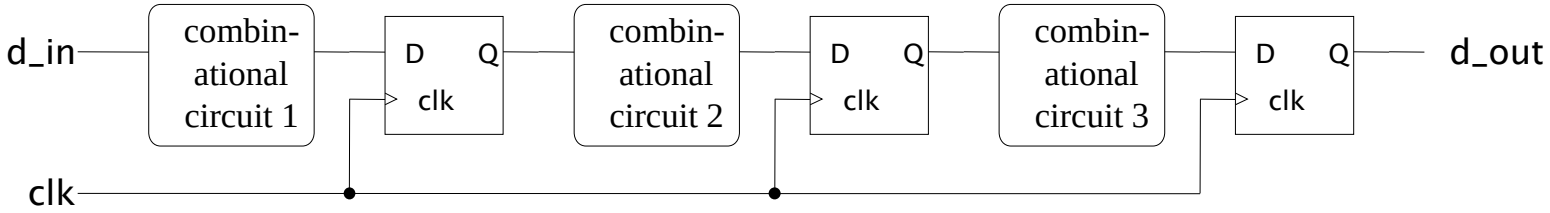
Pipelines Using Registers

Total delay = $Delay_1 + Delay_2 + Delay_3$

Clock period = $Delay_1 + Delay_2 + Delay_3$

Interval between outputs > Total delay

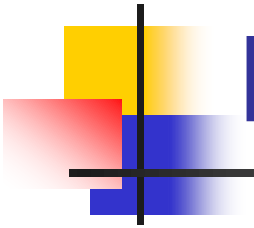




Clock period = max($Delay_1$, $Delay_2$, $Delay_3$)

Total delay = 3 × clock period

Interval between outputs = 1 clock period

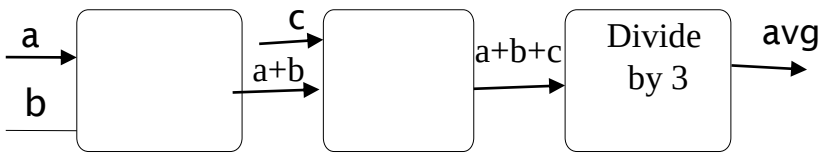


Example: Average Calculator

- Design a module that calculates the average of 3 inputs using 2-input Adder and one divider.
- First design the module in combinational style, assuming you receive one set of inputs.
- Next Pipeline the design to receive stream of inputs.

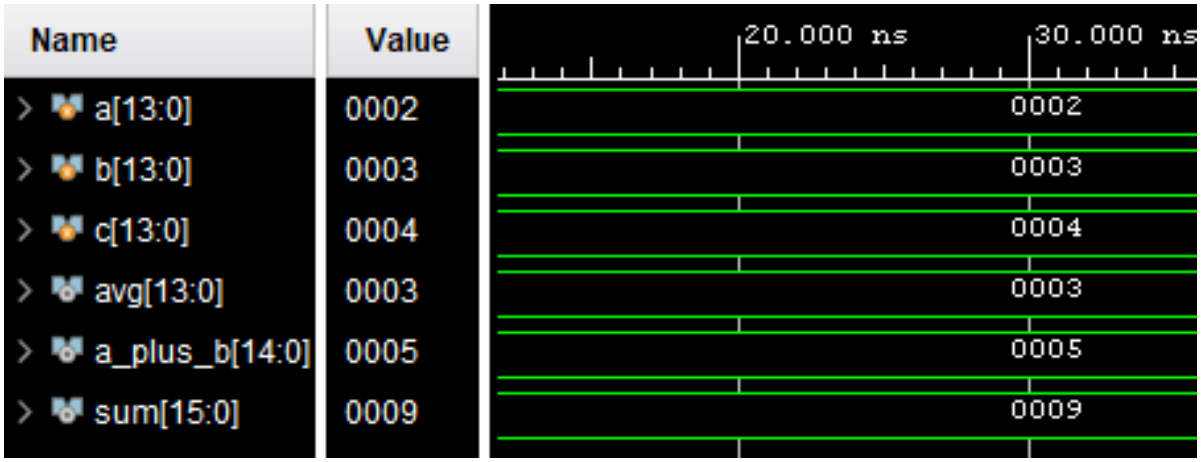
Example Combinational Module for Average

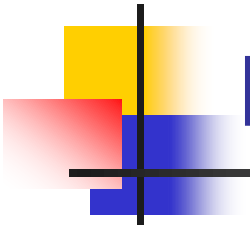
```
module average(output signed [0:13] avg,
               input signed [0:13] a,b,c);

wire signed [14:0] a_plus_b;
wire signed [15:0] sum;
assign a_plus_b=a+b;
assign sum = a_plus_b + c;
assign avg = sum / 3;
endmodule
```



Testbench and results

```
module average_tb();
reg signed [13:0] a,b,c;
wire signed [13:0] avg;

average uut(.a(a),
            .b(b),
            .c(c),
            .avg(avg));
initial begin
a=2;
b=3;
c=4;
end
endmodule
```

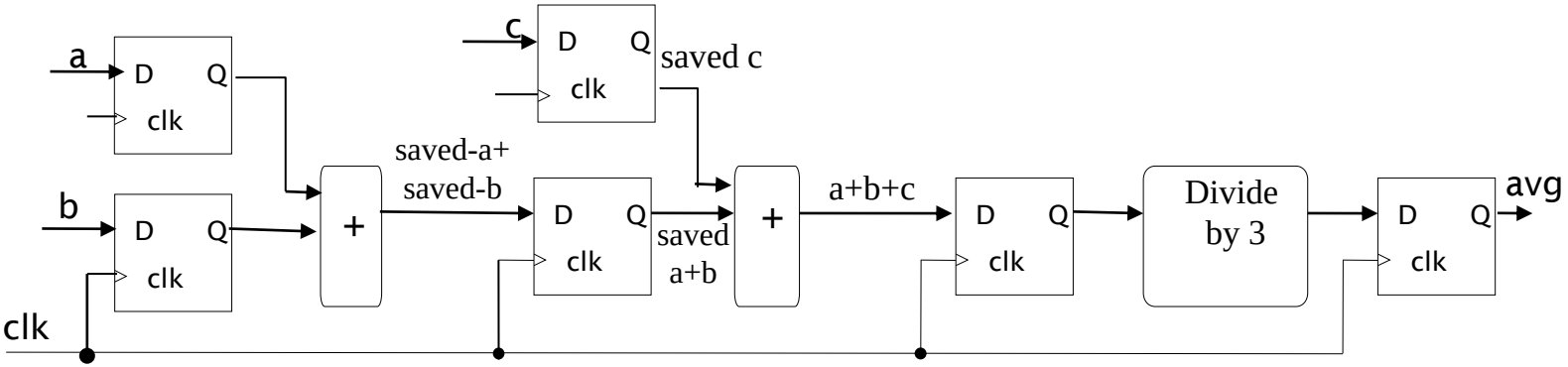




Pipeline Example of Average Module

- Compute the average of corresponding numbers in three input streams
  - New values arrive on each clock edge

Block diagram of the Pipeline Example



Verilog of the Pipeline Example

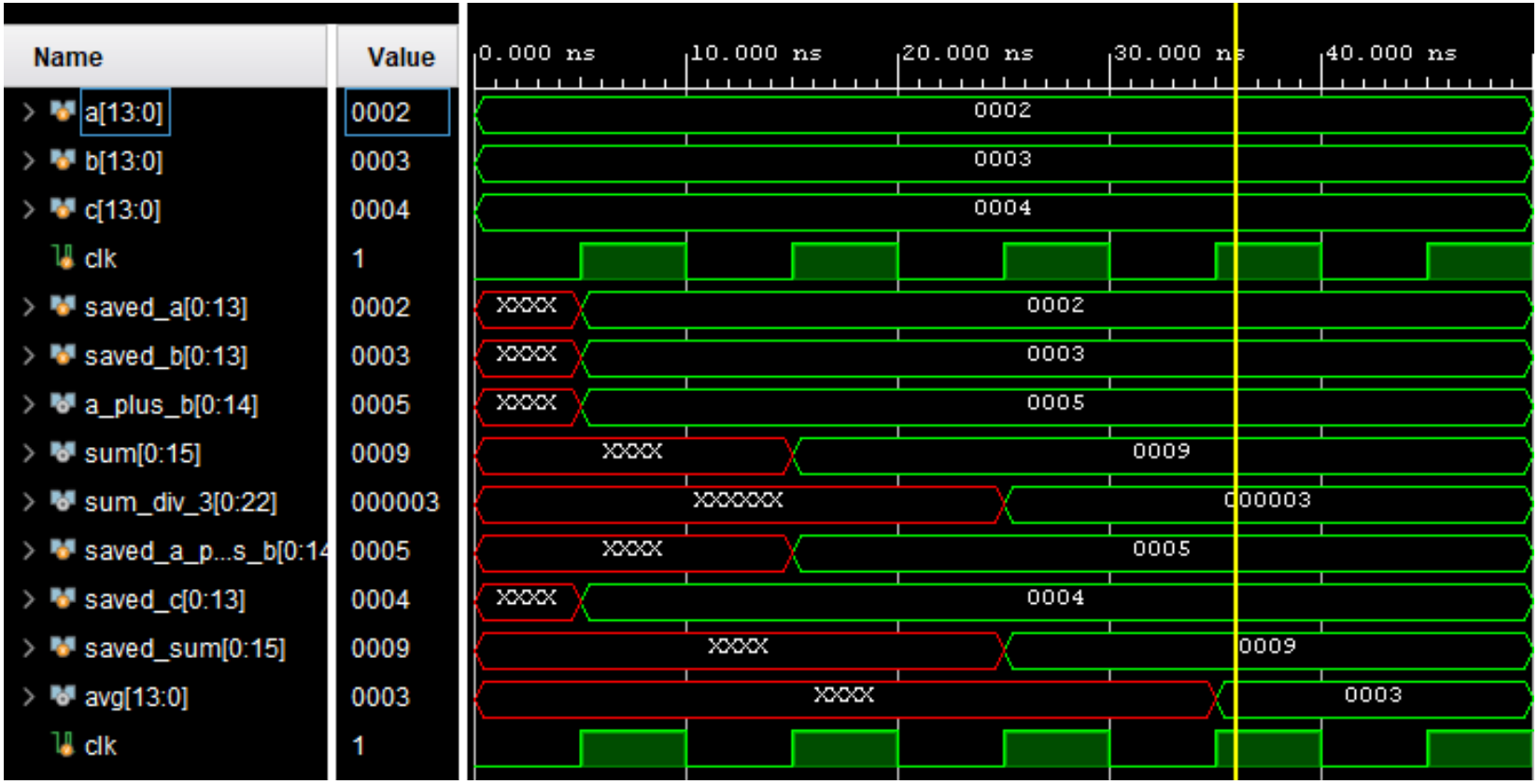
```
module pipeline(output reg signed
[0:13] avg,
                input signed [0:13] a,b,c,
                input clk);

reg signed [0:13] saved_a,saved_b;
wire signed [0:14] a_plus_b;
wire signed [0:15] sum;
wire signed [0:22] sum_div_3;
reg signed [0:14] saved_a_plus_b;
reg signed [0:13] saved_c;
reg [0:15] saved_sum;
```

```
always @(posedge clk)begin
    saved_a <= a;
    saved_b <= b;
end
assign a_plus_b = saved_a + saved_b;

always @(posedge clk)begin
    saved_a_plus_b <= a_plus_b;
    saved_c <= c;
end
assign sum = saved_a_plus_b + saved_c;
always @(posedge clk)begin
    saved_sum <= sum;
end
assign sum_div_3 = saved_sum /3;
always @(posedge clk)
    avg <= sum_div_3;
endmodule
```

Testbench for one input set

```
module pipeline_tb();
reg clk;
reg signed [13:0] a,b,c;
wire signed [13:0] avg;
pipeline uut(.clk(clk),
             .a(a),
             .b(b),
             .c(c),
             .avg(avg));

always #5 clk = ~clk;
initial begin
clk = 0;
a=2;
b=3;
c=4;
end
endmodule
```

Simulation Results

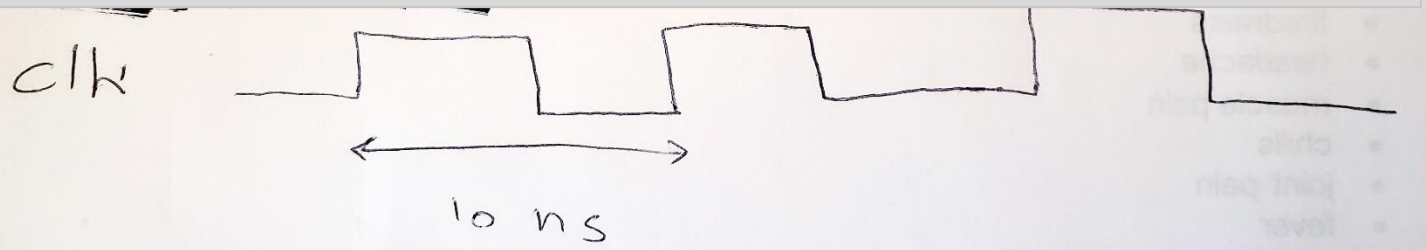


Testbench for streams of input sets

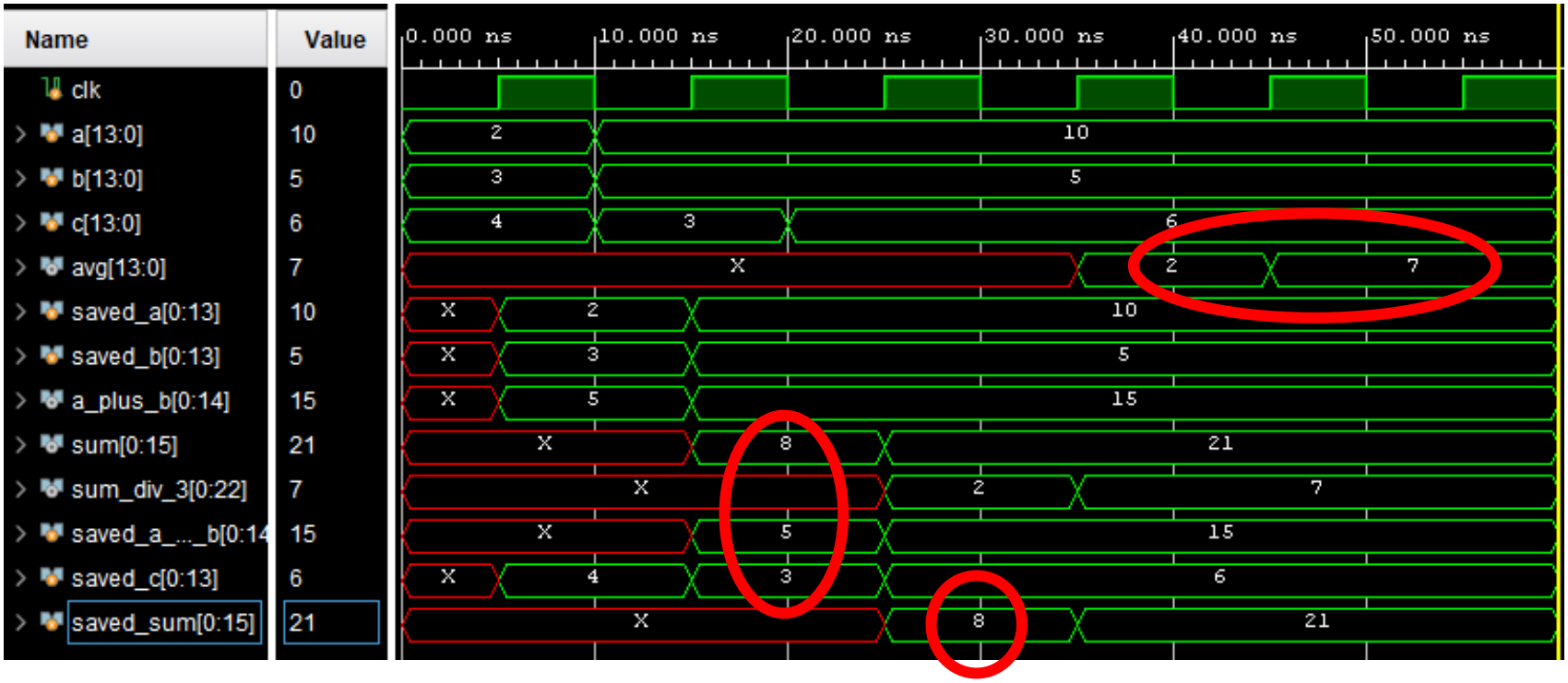
```
module pipeline_tb();
reg clk;
reg signed [13:0] a,b,c;
wire signed [13:0] avg;
pipeline uut(.clk(clk),
             .a(a),
             .b(b),
             .c(c),
             .avg(avg));

always #5 clk = ~clk;
initial begin
clk = 0;
a=2;
b=3;
c=4;

#10
a=10;
b=5;
c=3;

#10
a=10;
b=5;
c=6;
end
endmodule
```

Expected Results

	cycle 1	cycle 2	cycle 3
a:	2	10	10
b:	3	5	5
c:	4	3	6
sum:	9	18	21
Avg(sum/3):	3	6	7



Testbench Results with existing code



Wrong results! How to solve it?

Solution: Delay C one more cycle

```
module pipeline(output reg signed [0:13] avg,
                input signed [0:13] a,b,c,
                input clk);

reg signed [0:13] saved_a,saved_b;
wire signed [0:14] a_plus_b;
wire signed [0:15] sum;
wire signed [0:22] sum_div_3;
reg signed [0:14] saved_a_plus_b;
reg signed [0:13] saved_c, saved_saved_c;
reg [0:15] saved_sum;


always @(posedge clk)begin
    saved_a <= a;
    saved_b <= b;
end

assign a_plus_b = saved_a + saved_b;

always @(posedge clk)begin
    saved_a_plus_b <= a_plus_b;
    saved_c <= c;
   saved_saved_c <= saved_c;
end

assign sum = saved_a_plus_b + saved_saved_c;

always @(posedge clk)begin
    saved_sum <= sum;
end

assign sum_div_3 = saved_sum /3;

always @(posedge clk)
    avg <= sum_div_3;

endmodule
```

Testbench Results with corrected code