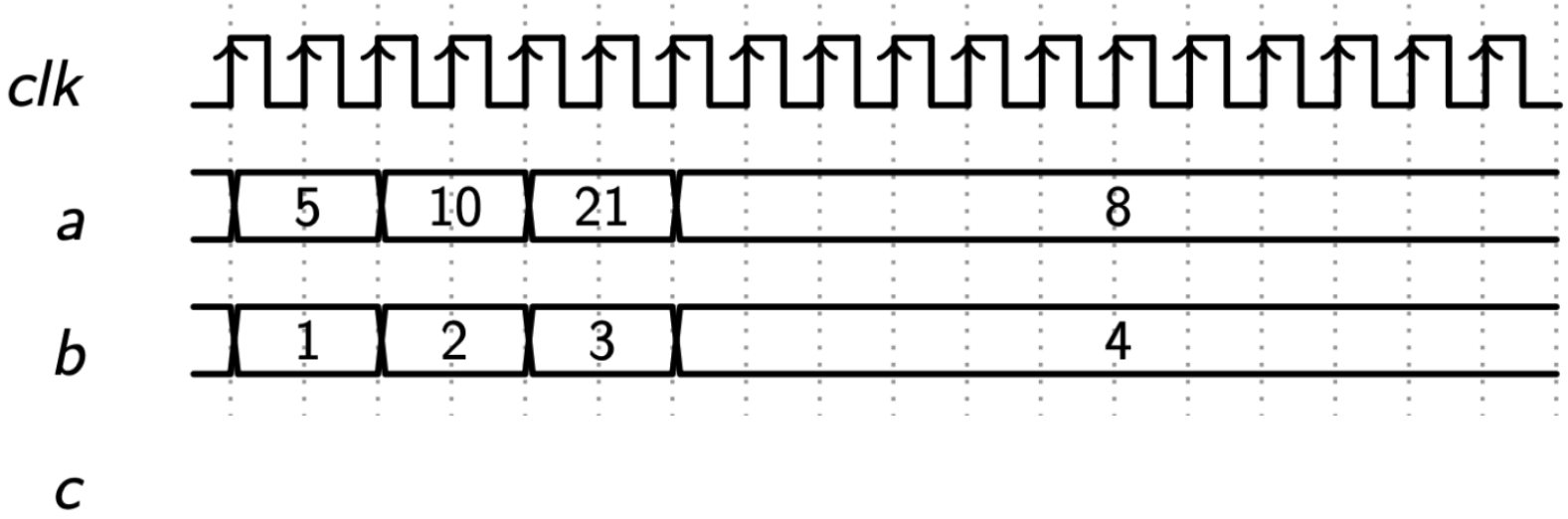
ECE 327/627
Digital Hardware Systems

Tutorial 5

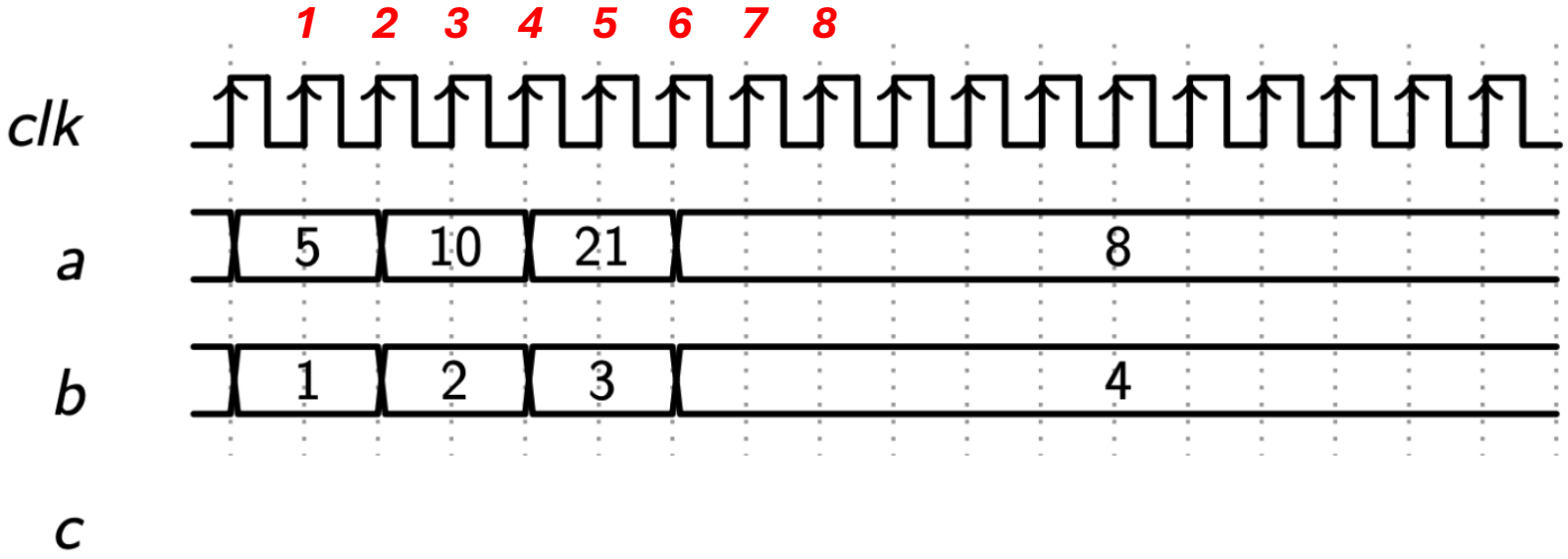
Q1

You implemented a divider circuit that computes c = a/b. Your circuit consumes inputs and produces an output every 2 cycles (i.e., throughput is 1/2 operations per cycle) and has a latency of 8 cycles. Complete the waveform given below.



Q1

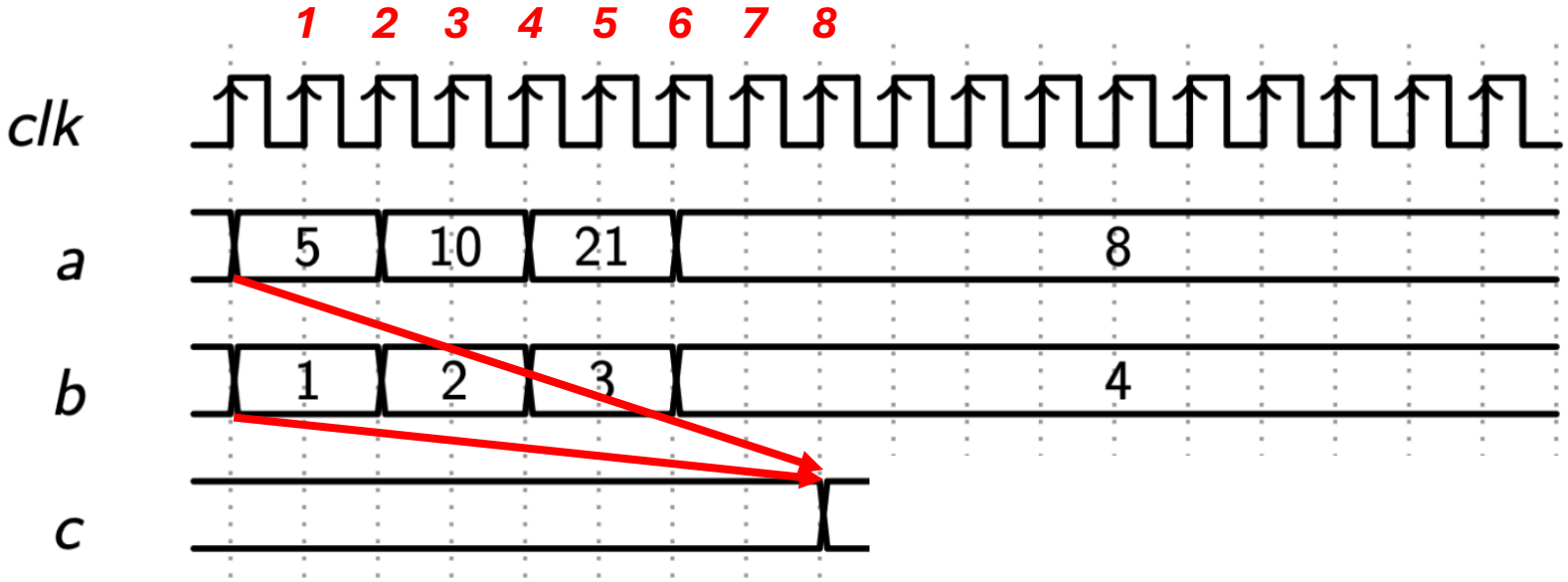
You implemented a divider circuit that computes c = a/b. Your circuit consumes inputs and produces an output every 2 cycles (i.e., throughput is 1/2 operations per cycle) and has a latency of 8 cycles. Complete the waveform given below.



For latency, count 8 positive clock edges from receiving an input. This is when the corresponding output appears on port c.

Q1

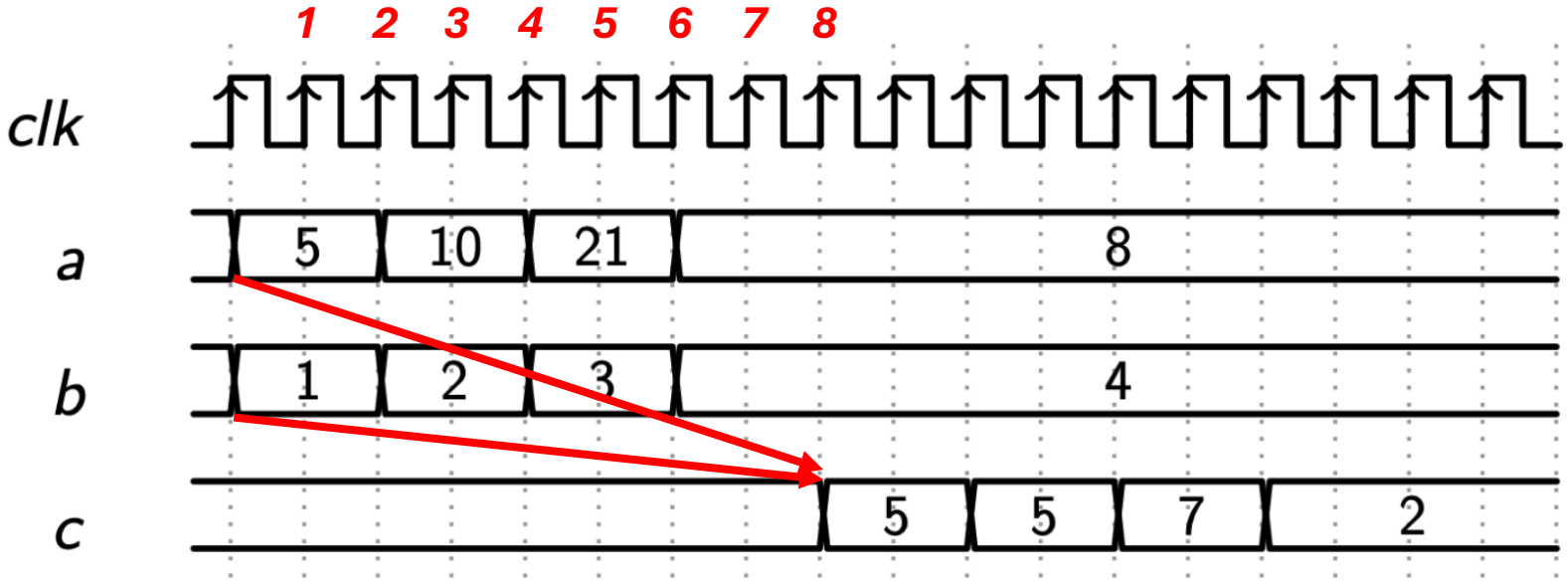
You implemented a divider circuit that computes c = a/b. Your circuit consumes inputs and produces an output every 2 cycles (i.e., throughput is 1/2 operations per cycle) and has a latency of 8 cycles. Complete the waveform given below.



For latency, count 8 positive clock edges from receiving an input. This is when the corresponding output appears on port c.

Q1

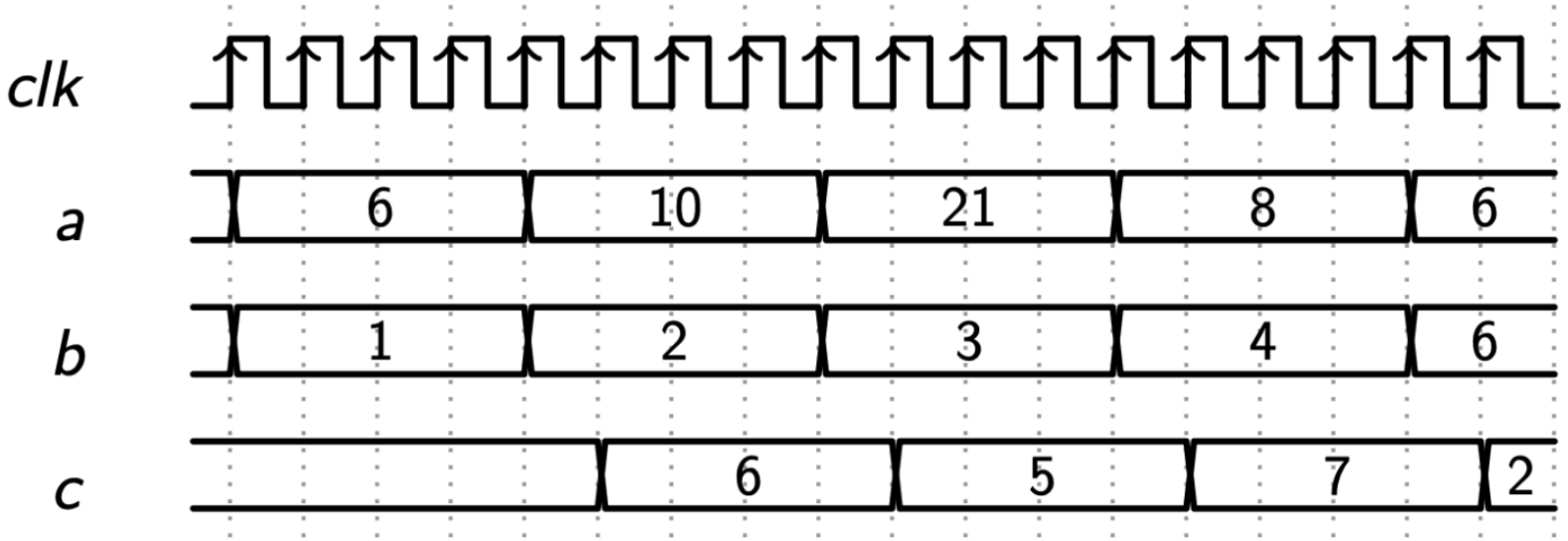
You implemented a divider circuit that computes c = a/b. Your circuit consumes inputs and produces an output every 2 cycles (i.e., throughput is 1/2 operations per cycle) and has a latency of 8 cycles. Complete the waveform given below.



Since inputs are consumed and an output is produced every 2 cycles. The output is fixed for 2 cycles, and its value is "a" divided by "b"

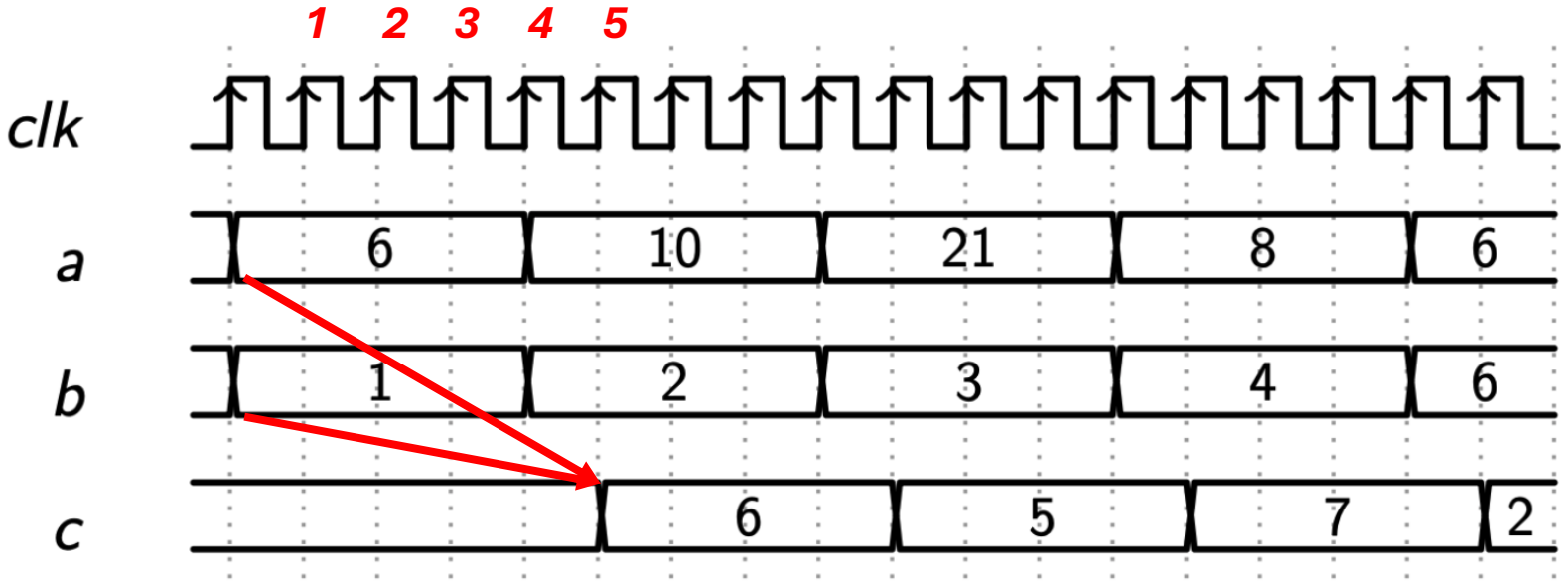
Q2

You then implemented a different divider from what you had in Q1. When simulating the new circuit, it generated the waveform given below. What is the throughput (in operations per cycle) and latency (in cycles) of this circuit?



Q2

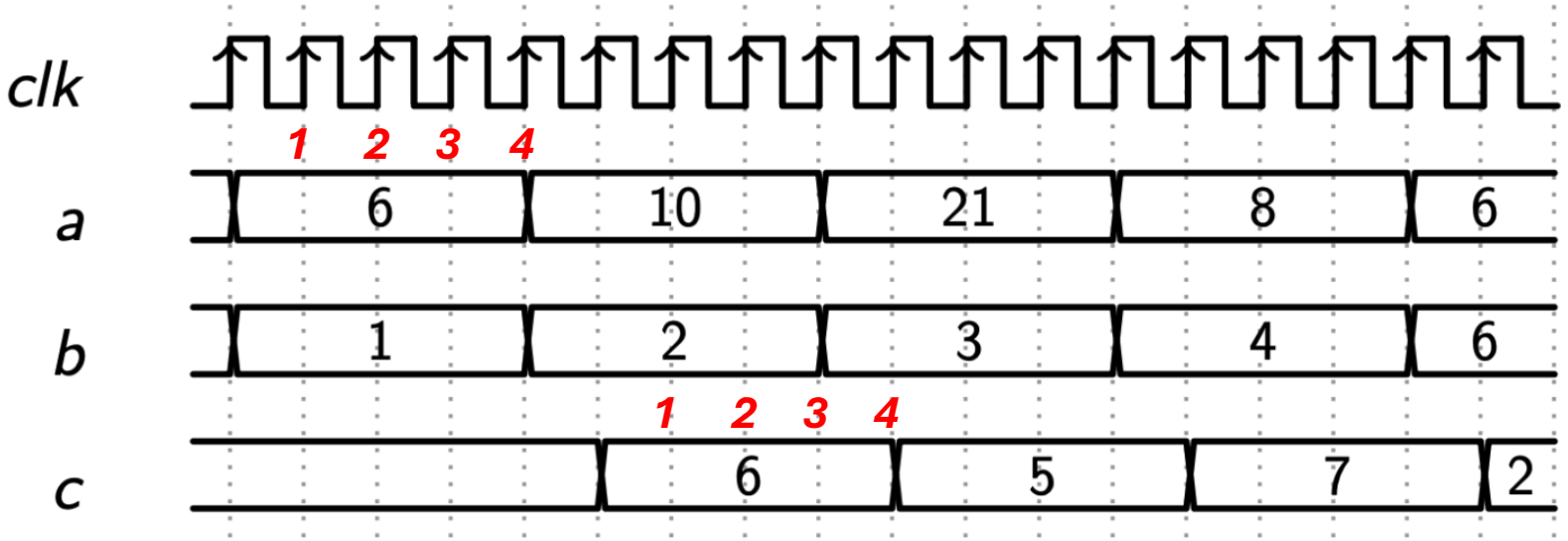
You then implemented a different divider from what you had in Q1. When simulating the new circuit, it generated the waveform given below. What is the throughput (in operations per cycle) and latency (in cycles) of this circuit?



To get the circuit latency, count the number of positive clock edges from receiving an input until its corresponding output appears on port c → ***Latency = 5 cycles***

Q2

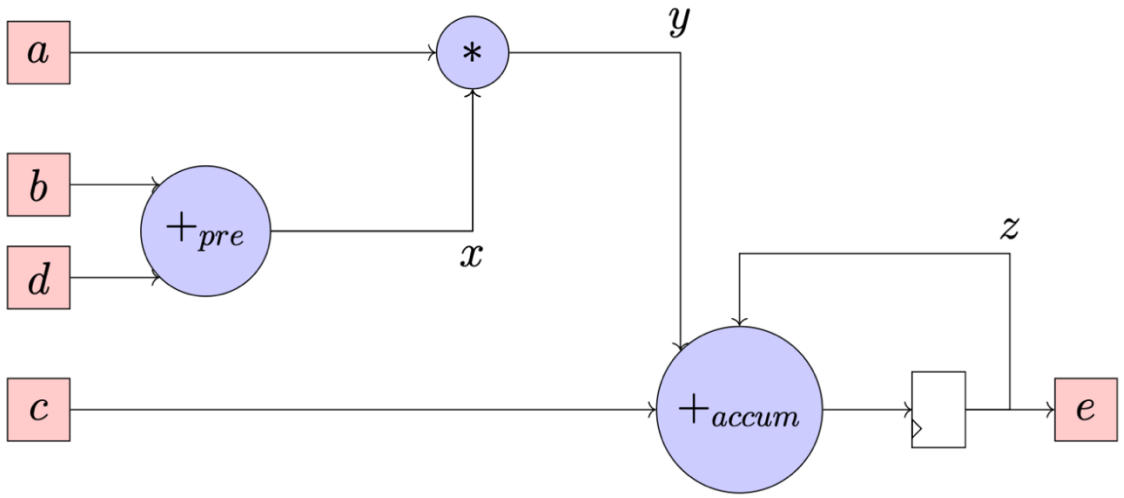
You then implemented a different divider from what you had in Q1. When simulating the new circuit, it generated the waveform given below. What is the throughput (in operations per cycle) and latency (in cycles) of this circuit?



To get the throughput, count the number of cycles between consuming/producing two consecutive inputs/outputs → ***Throughput = ¼ operation per cycle***

Q3

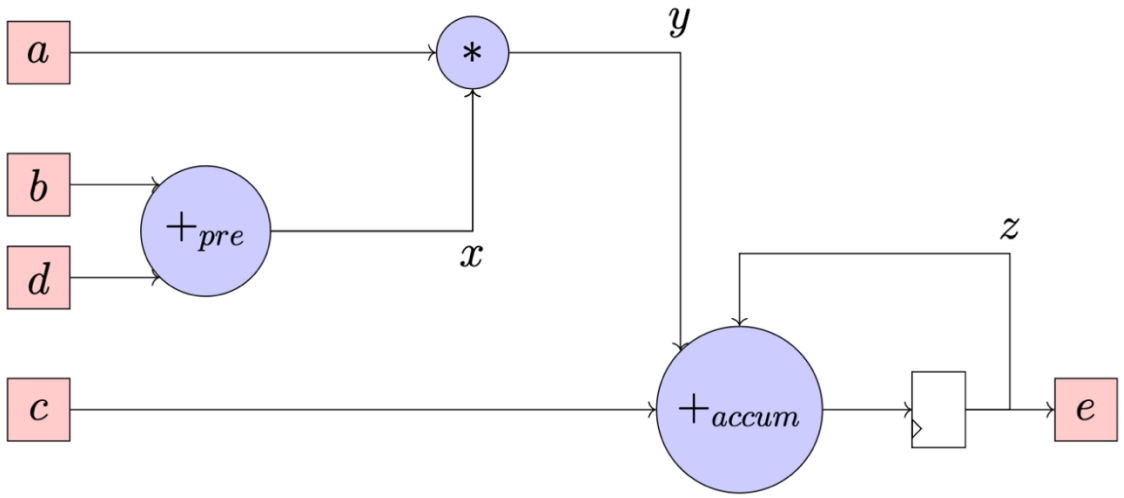
The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.



(a) Inputs a, b, c, and d are all 8-bit signed numbers. Output e is a 32-bit signed number. Calculate the number of bits required to represent the data on wires x and y.

Q3

The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.

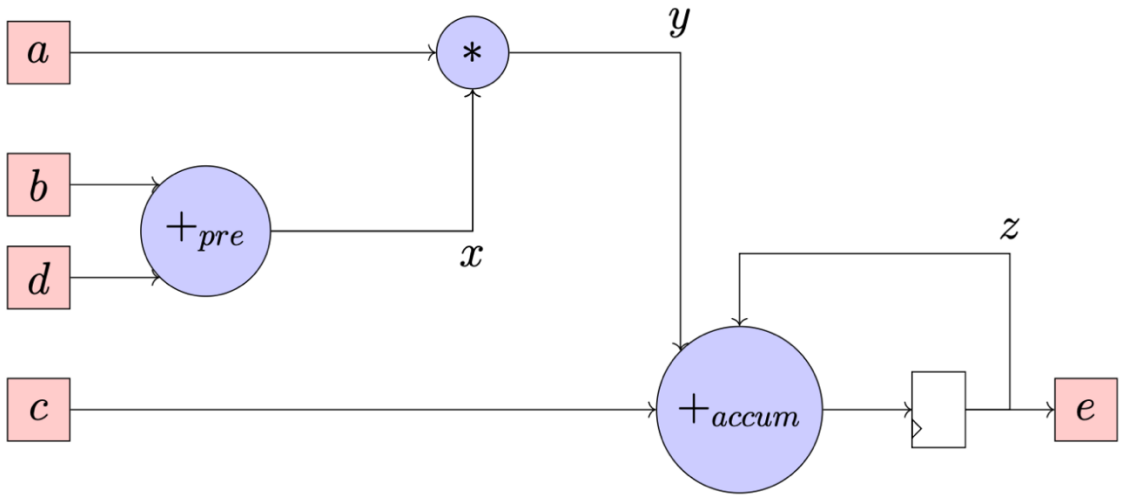


(a) Inputs a, b, c, and d are all 8-bit signed numbers. Output e is a 32-bit signed number. Calculate the number of bits required to represent the data on wires x and y.

An 8-bit signed number has a range from -128 to 127 so:
x has a range from -256 to 254 → ***needs 9 bits to represent***
y has a range from -32512 to 32768 → ***needs 17 bits to represent***

Q3

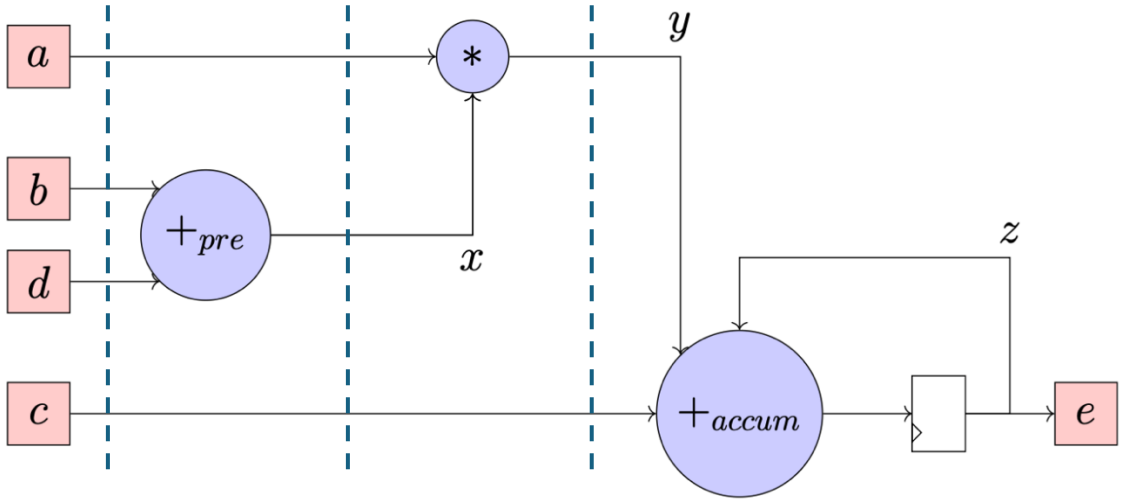
The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.



(b) You are developing a new high performance DSP block for a new FPGA chip. If you are allowed to fully pipeline the DSP block, where will you place the registers? How many extra FFs do you need? Assume that wire delay is negligible and all operators have equal delay.

Q3

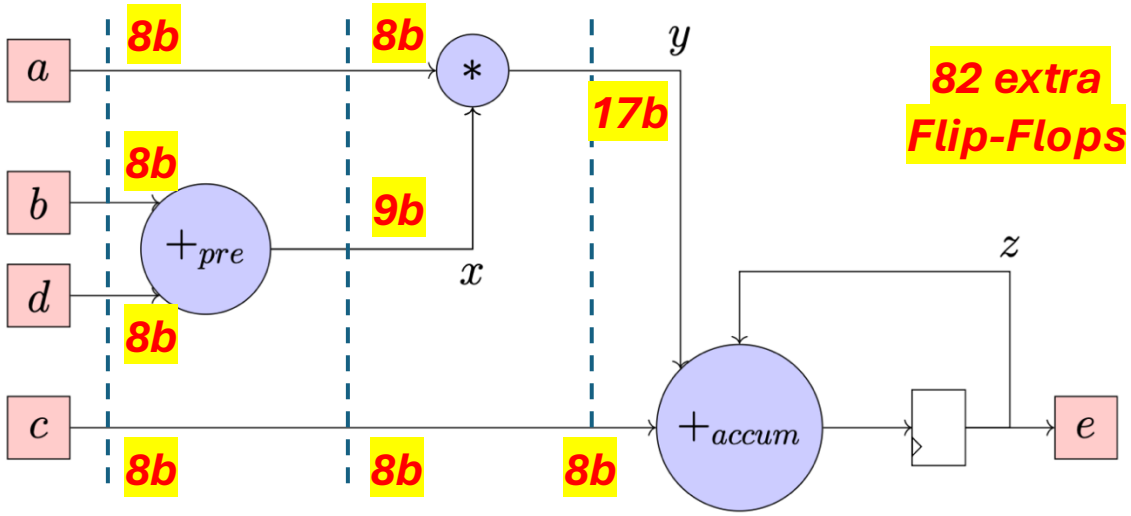
The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.



(b) You are developing a new high performance DSP block for a new FPGA chip. If you are allowed to fully pipeline the DSP block, where will you place the registers? How many extra FFs do you need? Assume that wire delay is negligible and all operators have equal delay.

Q3

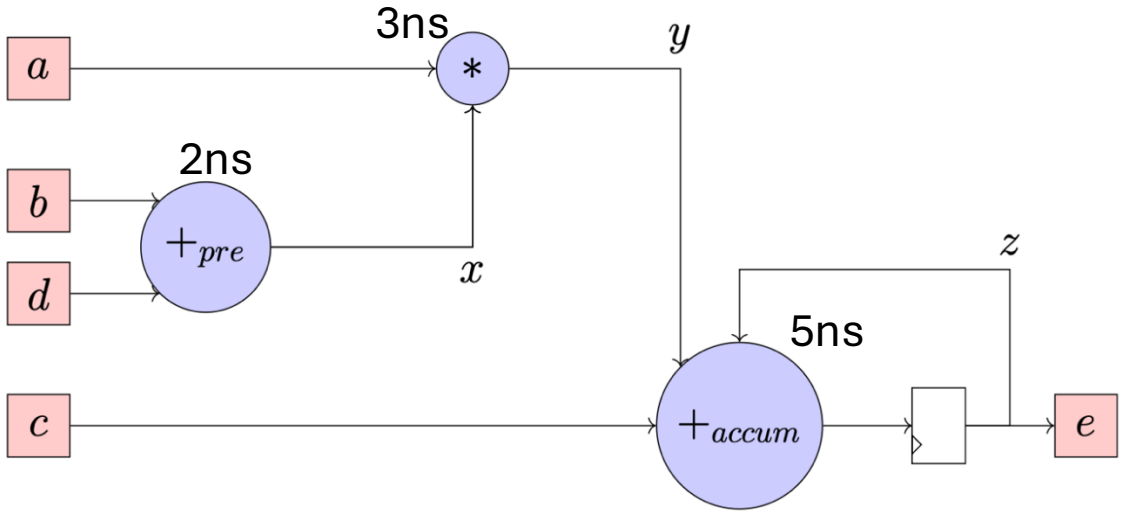
The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.



(b) You are developing a new high performance DSP block for a new FPGA chip. If you are allowed to fully pipeline the DSP block, where will you place the registers? How many extra FFs do you need? Assume that wire delay is negligible and all operators have equal delay.

Q3

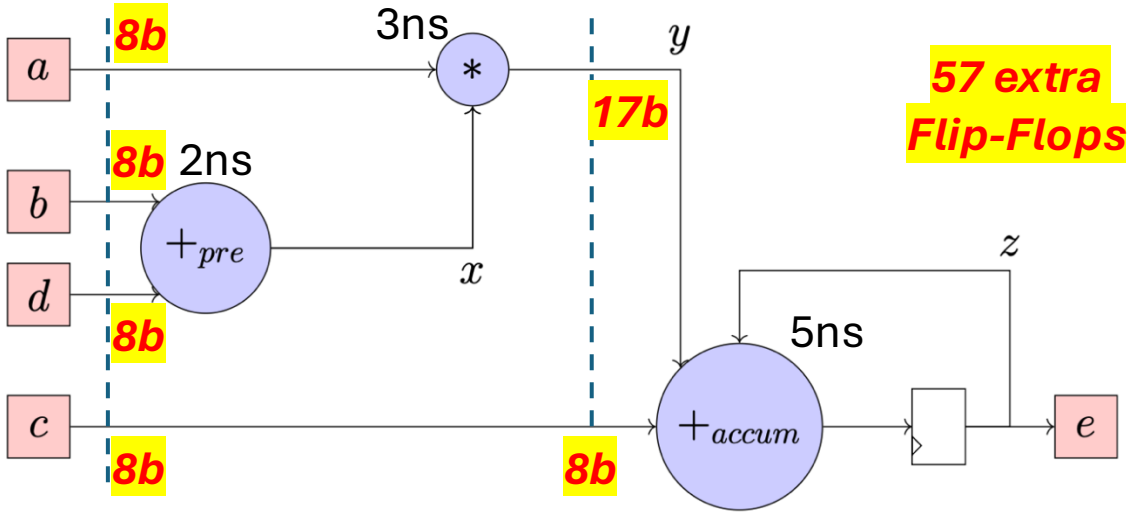
The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.



(c) Upon closer inspection, you observe that the delays of different operators are not the same (annotated on the figure). In this scenario, how would you change the pipelining of the circuit?

Q3

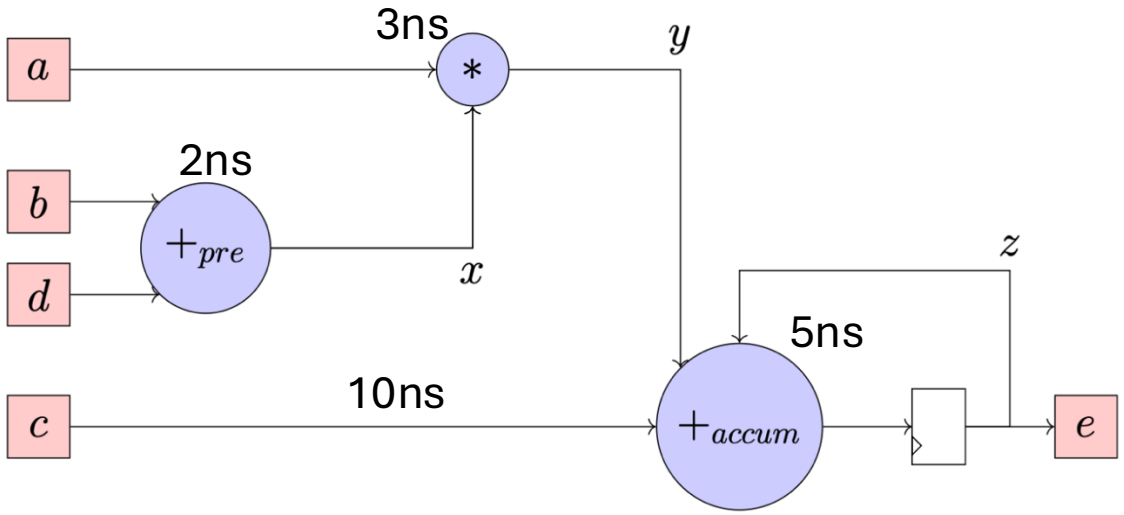
The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.



(c) Upon closer inspection, you observe that the delays of different operators are not the same (annotated on the figure). In this scenario, how would you change the pipelining of the circuit?

Q3

The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.

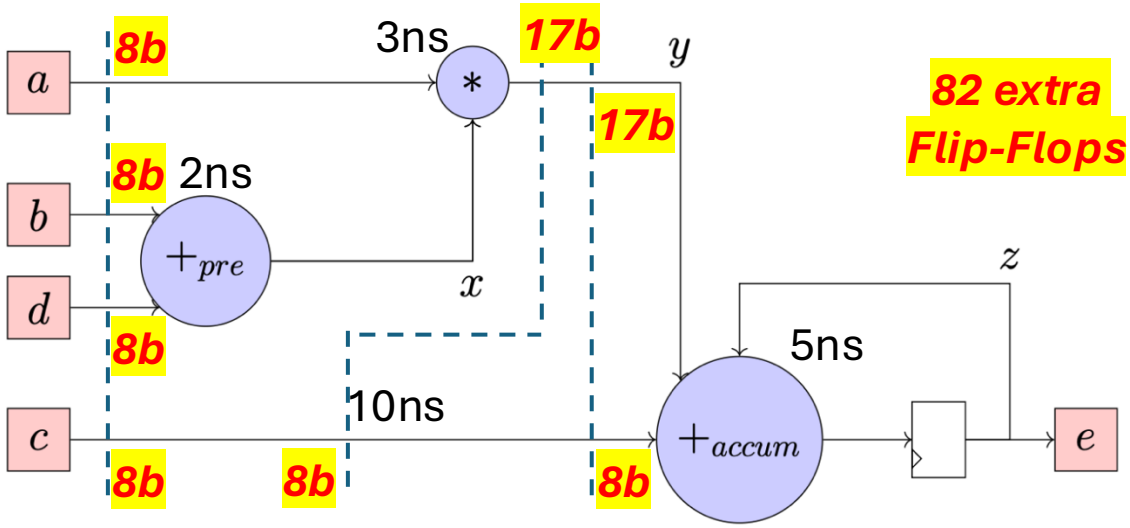


(d) After placement and routing, you notice that the wire connecting “c” to the “+accum” operator has a delay of 10 ns. In this scenario, how would you change the pipelining of the circuit?

Q3

The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.

Solution 1:

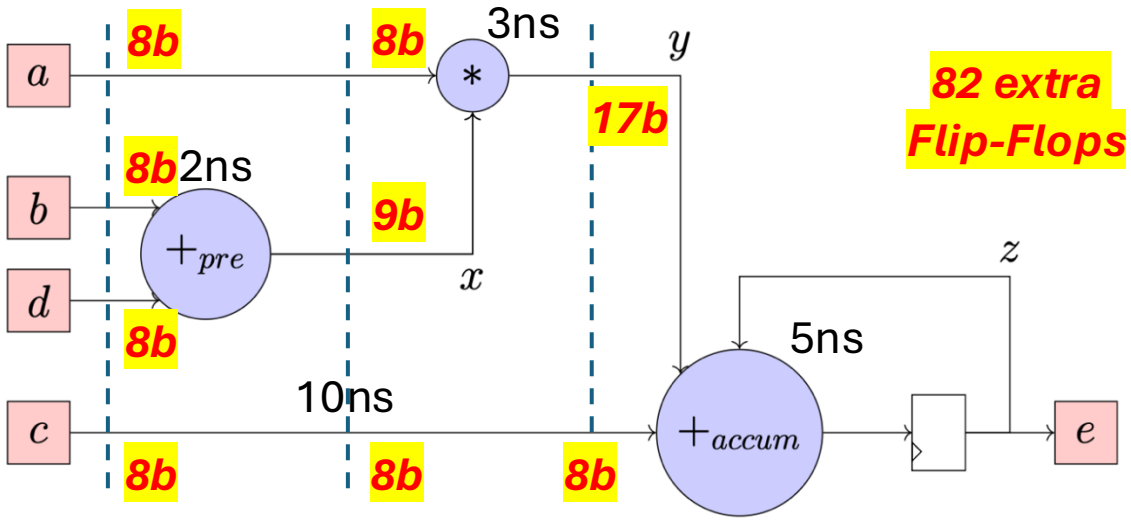


(d) After placement and routing, you notice that the wire connecting "c" to the "+accum" operator has a delay of 10 ns. In this scenario, how would you change the pipelining of the circuit?

Q3

The figure below is a simplified representation of an FPGA Digital Signal Processing (DSP) block. DSP blocks are widely used inside an FPGA to perform arithmetic operations.

Solution 2:



(d) After placement and routing, you notice that the wire connecting "c" to the "+accum" operator has a delay of 10 ns. In this scenario, how would you change the pipelining of the circuit?

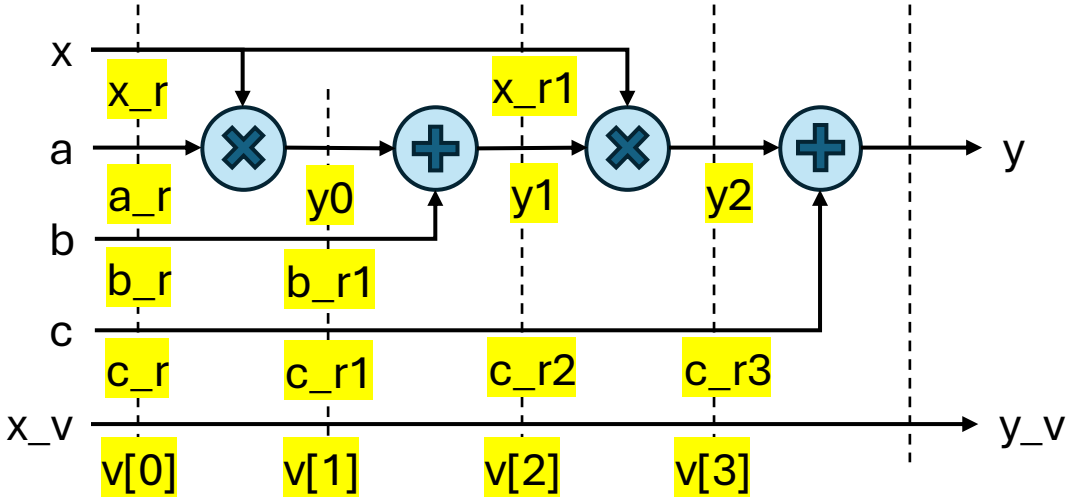
Q4

Below is an implementation of the polynomial circuit that calculates $y = ax^2 + bx + c$. What is the latency of this implementation? Every how many cycles new inputs can be provided to this circuit to be functionally correct?

```
module q4(
  input logic clk,
  input logic rst,
  input logic [7:0] a,
  input logic [7:0] b,
  input logic [7:0] c,
  input logic [7:0] x,
  input logic x_v,
  output logic [23:0] y,
  output logic y_v
);

logic [15:0] y0, y1; logic [23:0] y2;
logic [7:0] a_r;
logic [7:0] x_r, x_r1;
logic [7:0] b_r, b_r1;
logic [7:0] c_r, c_r1, c_r2, c_r3;
logic [3:0] v;

always_ff @(posedge clk) begin
  if (rst) begin
    y0 <= 'd0; y1 <= 'd0; y2 <= 'd0;
    y <= 'd0; x_r1 <= 'd0; b_r1 <= 'd0;
    c_r1 <= 'd0; c_r2 <= 'd0; c_r3 <= 'd0;
  end else begin
    // stage 1
    y0 <= a_r * x_r;
    b_r1 <= b_r;
    c_r1 <= c_r;
    // stage 2
    y1 <= y0 + b_r1;
    x_r1 <= x_r;
    c_r2 <= c_r1;
    // stage 3
    y2 <= y1 * x_r1;
    c_r3 <= c_r2;
    // stage 4
    y <= y2 + c_r3;
  end
end

always_ff @(posedge clk) begin
  if (rst) begin
    v <= {4{1'b0}};
    y_v <= 1'b0;
  end else begin
    v[0] <= x_v;
    v[1] <= v[0];
    v[2] <= v[1];
    v[3] <= v[2];
    y_v <= v[3];
  end
end

always_ff @(posedge clk) begin
  if (rst) begin
    a_r <= {8{1'b0}};
    b_r <= {8{1'b0}};
    c_r <= {8{1'b0}};
    x_r <= {8{1'b0}};
  end else begin
    a_r <= a;
    b_r <= b;
    c_r <= c;
    x_r <= x;
  end
end

endmodule
```

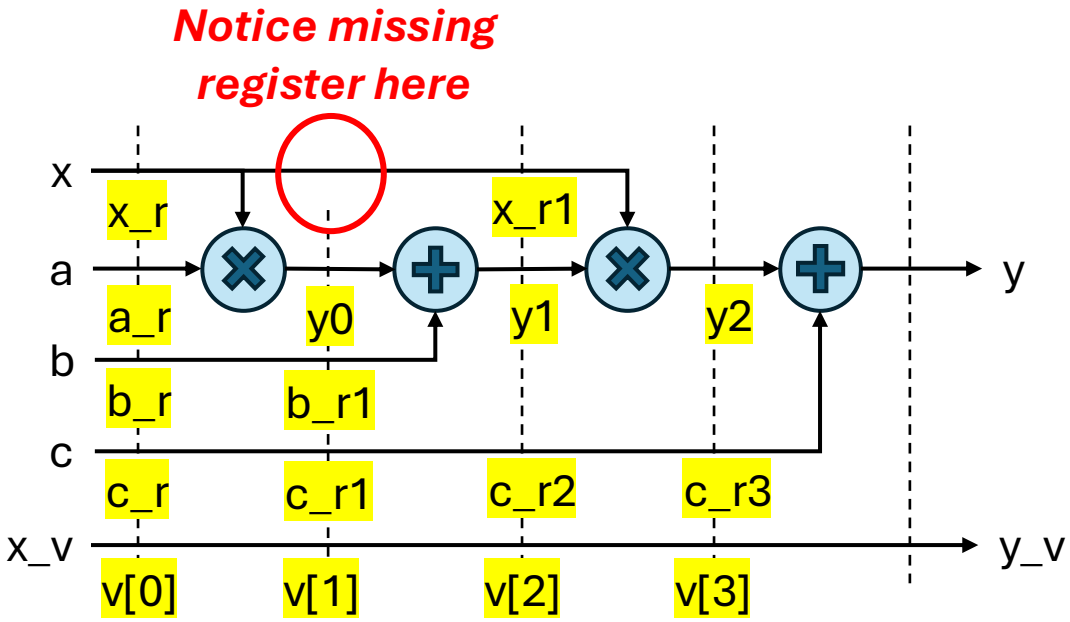
Q4

Below is an implementation of the polynomial circuit that calculates $y = ax^2 + bx + c$. What is the latency of this implementation? Every how many cycles new inputs can be provided to this circuit to be functionally correct?



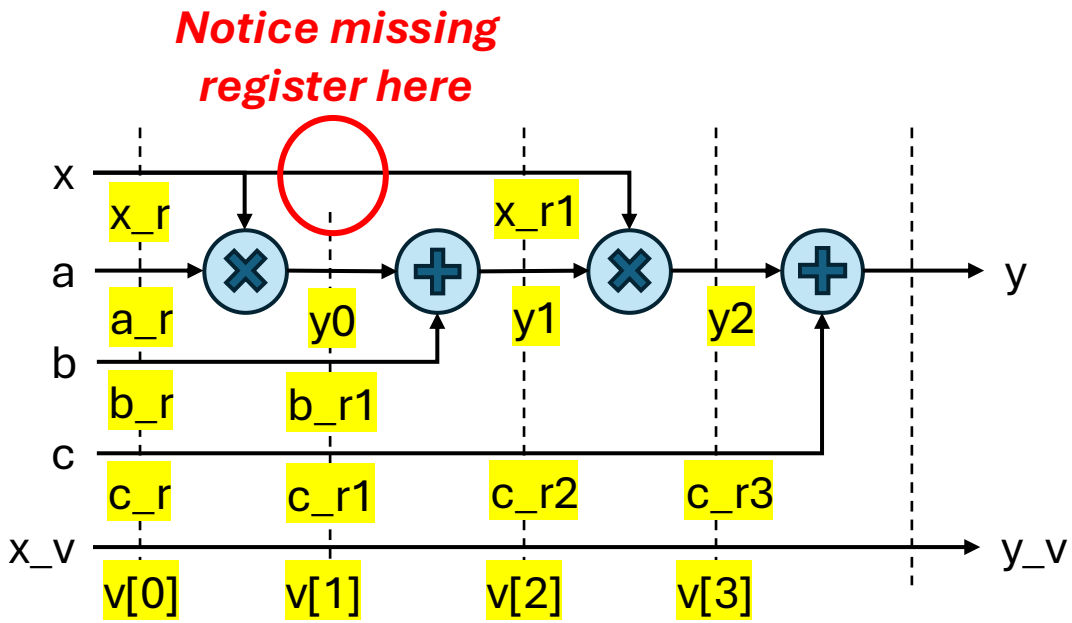
Q4

Below is an implementation of the polynomial circuit that calculates $y = ax^2 + bx + c$. What is the latency of this implementation? Every how many cycles new inputs can be provided to this circuit to be functionally correct?



Q4

Below is an implementation of the polynomial circuit that calculates $y = ax^2 + bx + c$. What is the latency of this implementation? Every how many cycles new inputs can be provided to this circuit to be functionally correct?



Latency = time from inputs to outputs = ***5 x clock period (5 cycles)***

Register x_r1 needs to hold its value for 2 cycles for the circuit to function correctly →

provide inputs every other cycle = 4 ops / 2 cycles = ***2 ops/cycle***

Q4

How can you fix it such that you can provide new inputs every cycle (i.e., full throughput)?

Q4

How can you fix it such that you can provide new inputs every cycle (i.e., full throughput)?

```
module q4(
  input logic clk,
  input logic rst,
  input logic [7:0] a,
  input logic [7:0] b,
  input logic [7:0] c,
  input logic [7:0] x,
  input logic x_v,
  output logic [23:0] y,
  output logic y_v
);

logic [15:0] y0, y1; logic [23:0] y2;
logic [7:0] a_r;
logic [7:0] x_r, x_r1, x_extra;
logic [7:0] b_r, b_r1;
logic [7:0] c_r, c_r1, c_r2, c_r3;
logic [3:0] v;

always_ff @(posedge clk) begin
  if (rst) begin
    y0 <= 'd0; y1 <= 'd0; y2 <= 'd0;
    y <= 'd0; x_r1 <= 'd0; b_r1 <= 'd0;
    c_r1 <= 'd0; c_r2 <= 'd0; c_r3 <= 'd0;
    x_extra <= 'd0;
  end else begin
    // stage 1
    y0 <= a_r * x_r;
    b_r1 <= b_r;
    c_r1 <= c_r;
    x_extra <= x_r;
    // stage 2
    y1 <= y0 + b_r1;
    x_r1 <= x_extra;
    c_r2 <= c_r1;
    // stage 3
    y2 <= y1 * x_r1;
    c_r3 <= c_r2;
    // stage 4
    y <= y2 + c_r3;
  end
end

always_ff @(posedge clk) begin
  if (rst) begin
    v <= {4{1'b0}};
    y_v <= 1'b0;
  end else begin
    v[0] <= x_v;
    v[1] <= v[0];
    v[2] <= v[1];
    v[3] <= v[2];
    y_v <= v[3];
  end
end

always_ff @(posedge clk) begin
  if (rst) begin
    a_r <= {8{1'b0}};
    b_r <= {8{1'b0}};
    c_r <= {8{1'b0}};
    x_r <= {8{1'b0}};
  end else begin
    a_r <= a;
    b_r <= b;
    c_r <= c;
    x_r <= x;
  end
end

endmodule
```

Q4

How can you fix it such that you can provide new inputs every cycle (i.e., full throughput)?