Binary Multiplication

Andrew Lukefahr
Indiana University - Bloomington

Calculator.py

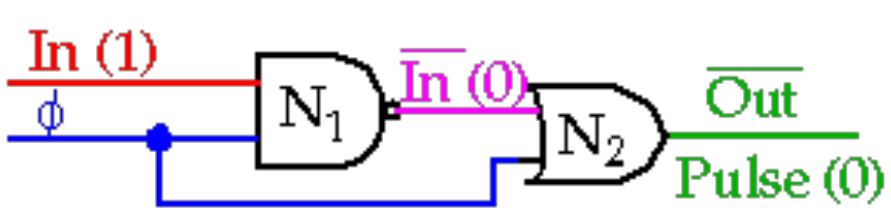
- Lets you interact with the FPGA via the keyboard
  - Translates ASCII <--> Binary for you

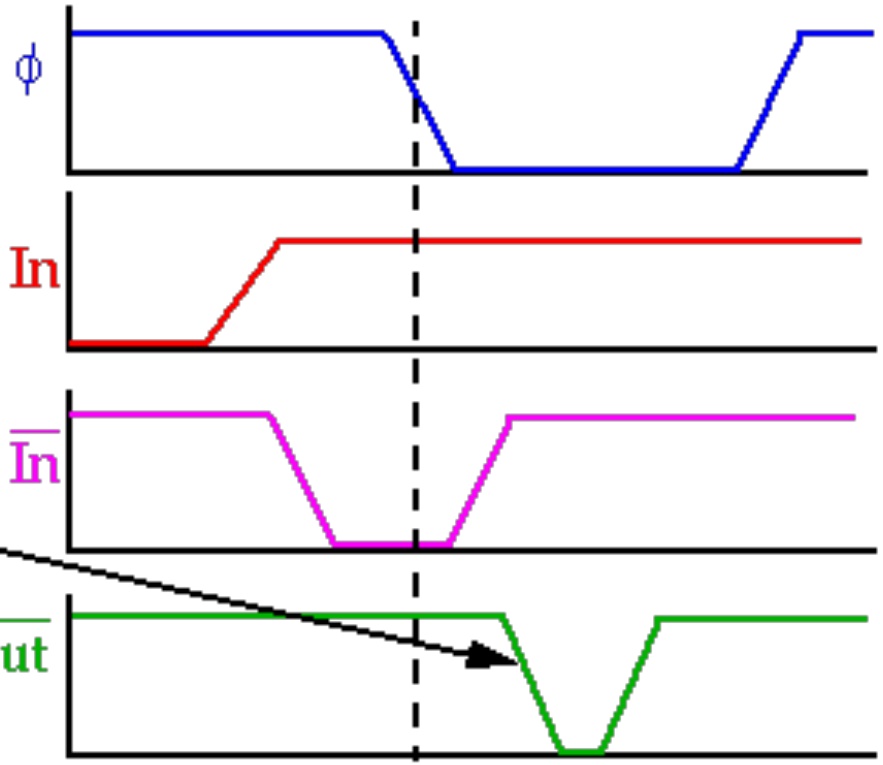
```
$ python3 calculator.py –s /dev/ttyUSB1
```

```
$ python3 calculator.py –-help
```

python3 (not “python”)

Review: Glitch





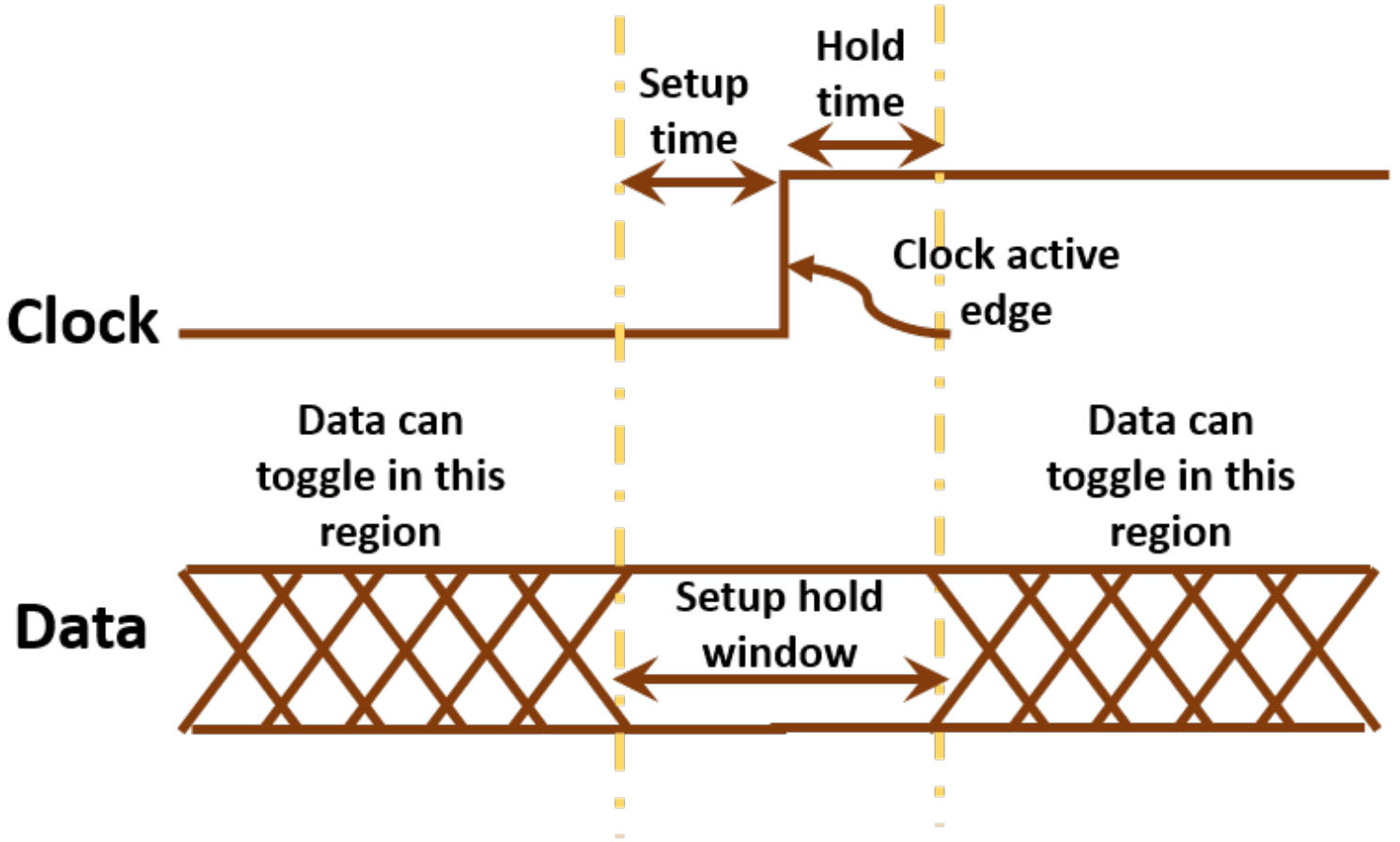
Results in a short low-going pulse at the output of N_2 with length approximately equal to the propagation delay through N_1.

Last Time: Setup and Hold Time

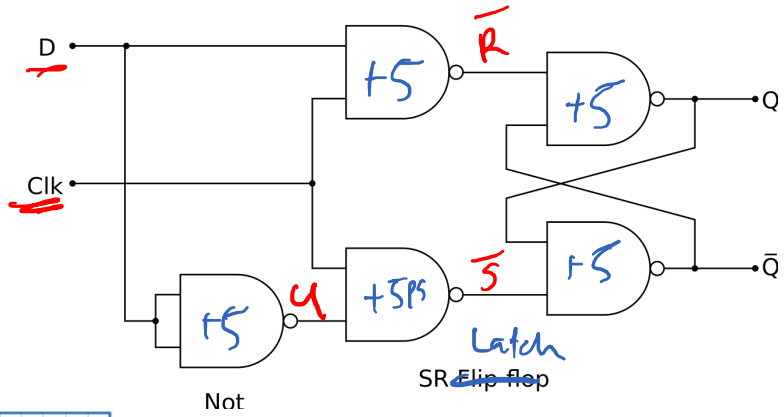
- **Setup Time**: minimum time the inputs to a flip-flop **must** be stable **before** the clock edge

- **Hold Time:** minimum time the inputs to a flip-flop **must** be stable **after** the clock edge

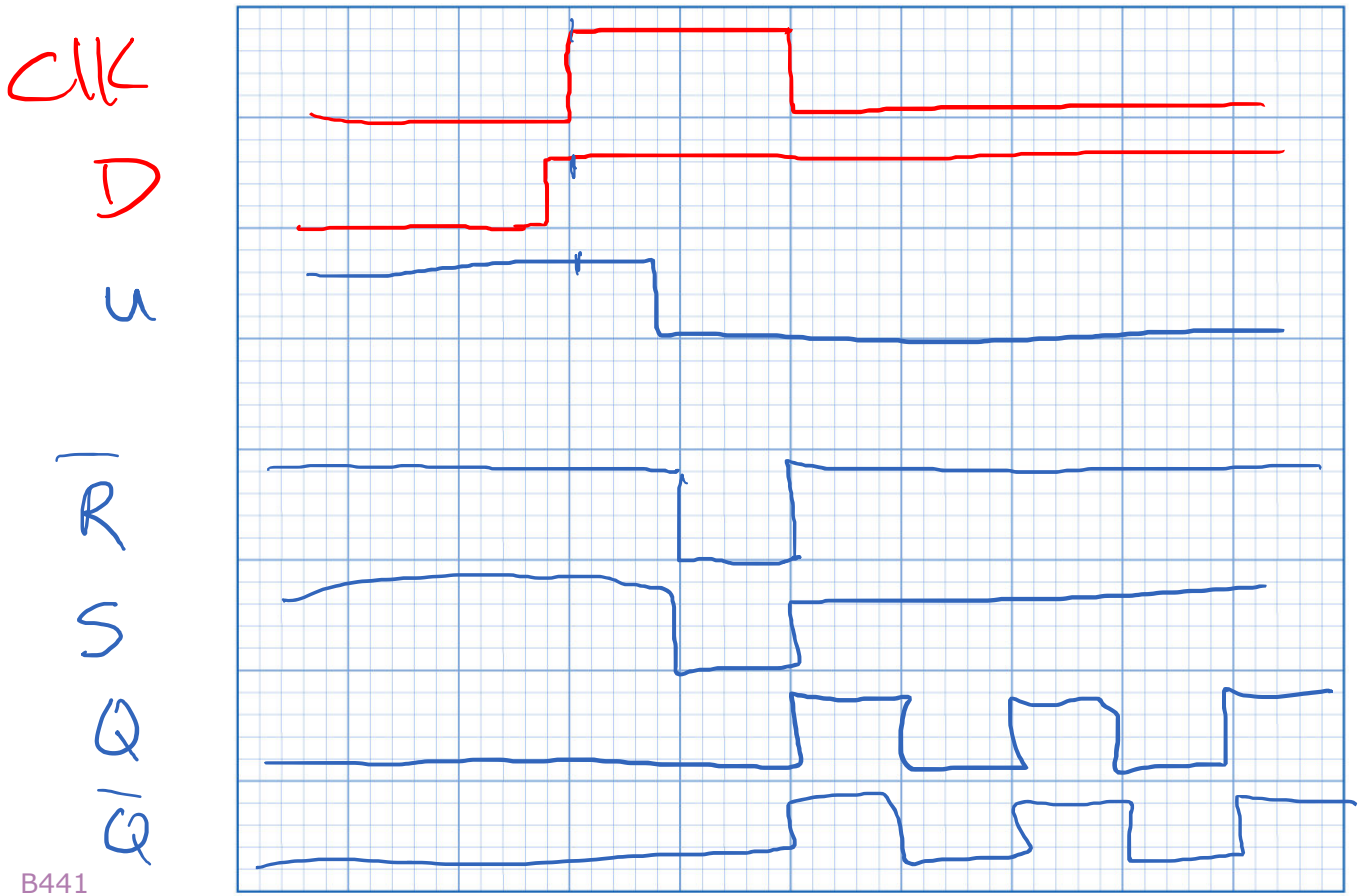
Setup and Hold Time



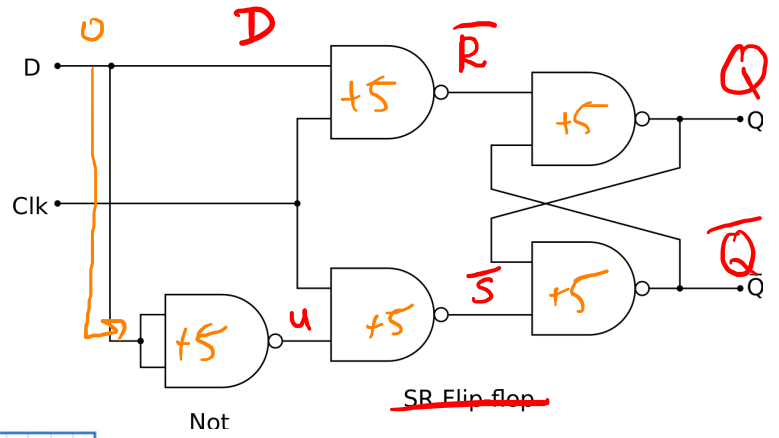
Setup Time Problems



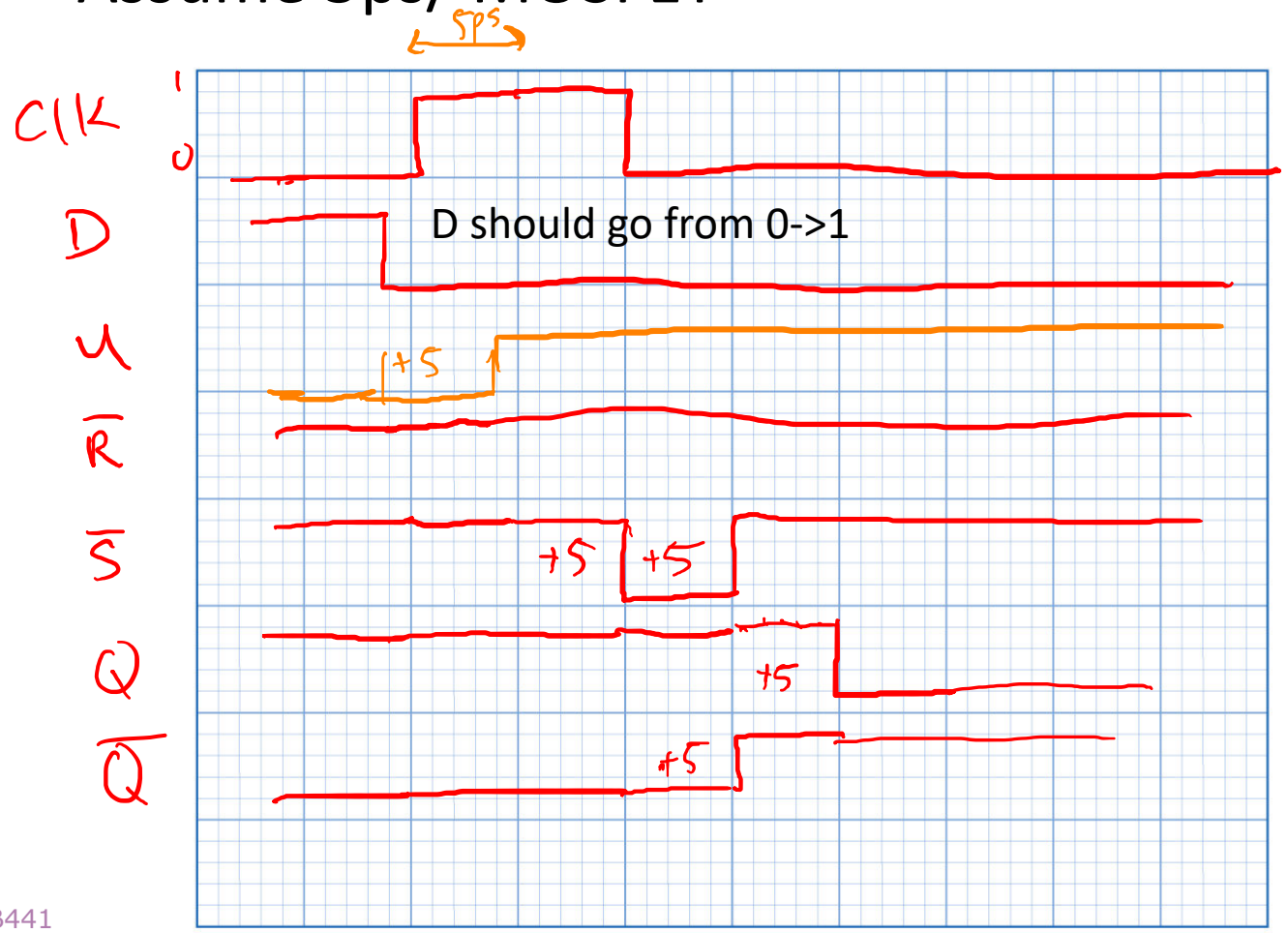
- Assume 5ps/ MOSFET



Setup Time Problems



- Assume 5ps/ MOSFET



screwed this up redo next time...

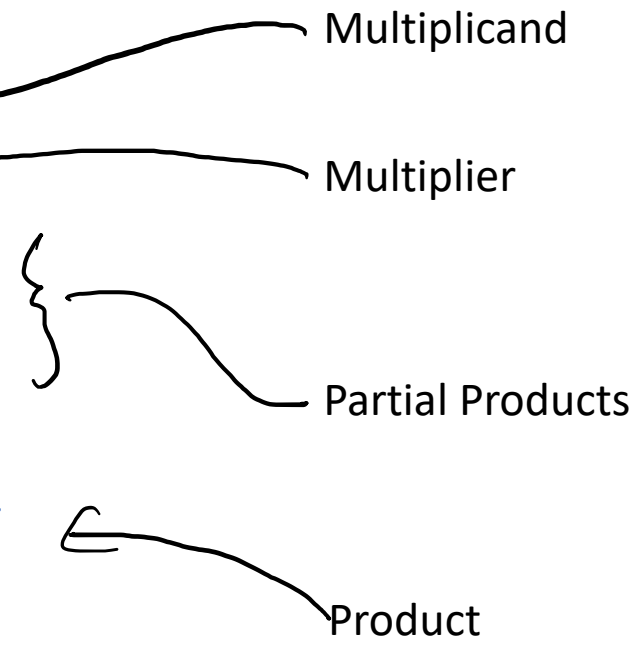
Decimal Multiplication

$$\begin{array}{r} 6 \\ \times\ 3 \\ \hline 18 \end{array}$$

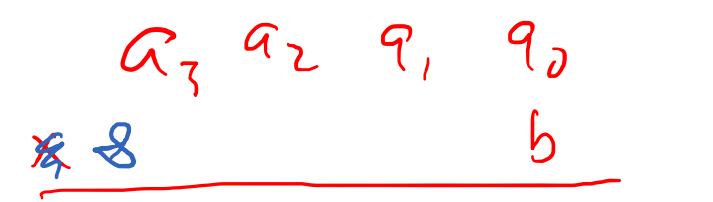
$$\begin{array}{r} 62 \\ \times\ 3 \\ \hline 186 \end{array}$$

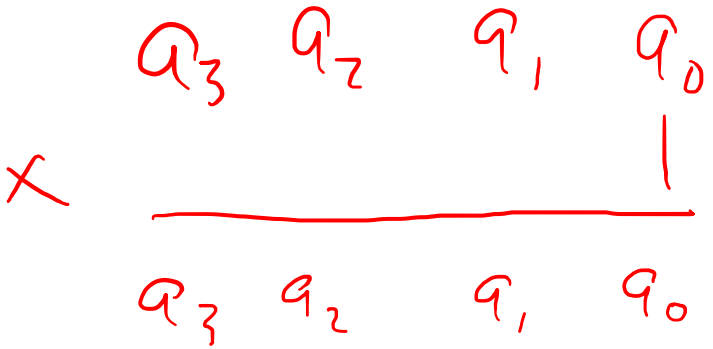
$$\begin{array}{r} 62 \\ \times\ 31 \\ \hline 62 \\ +1860 \\ \hline 1922 \end{array}$$

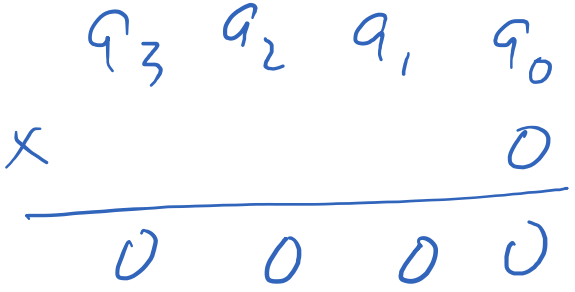
$$\begin{array}{r} 624 \\ \times\ 312 \\ \hline 1248 \\ 6240 \\ +187200 \\ \hline 194688 \end{array}$$



Binary Multiplication








```
     6
×    3
-----
    18
```

```
      0110
×     0011
-----------
      0110
     0110
    0000
+  0000
-----------
  00100110
```

16 + 2 = 18

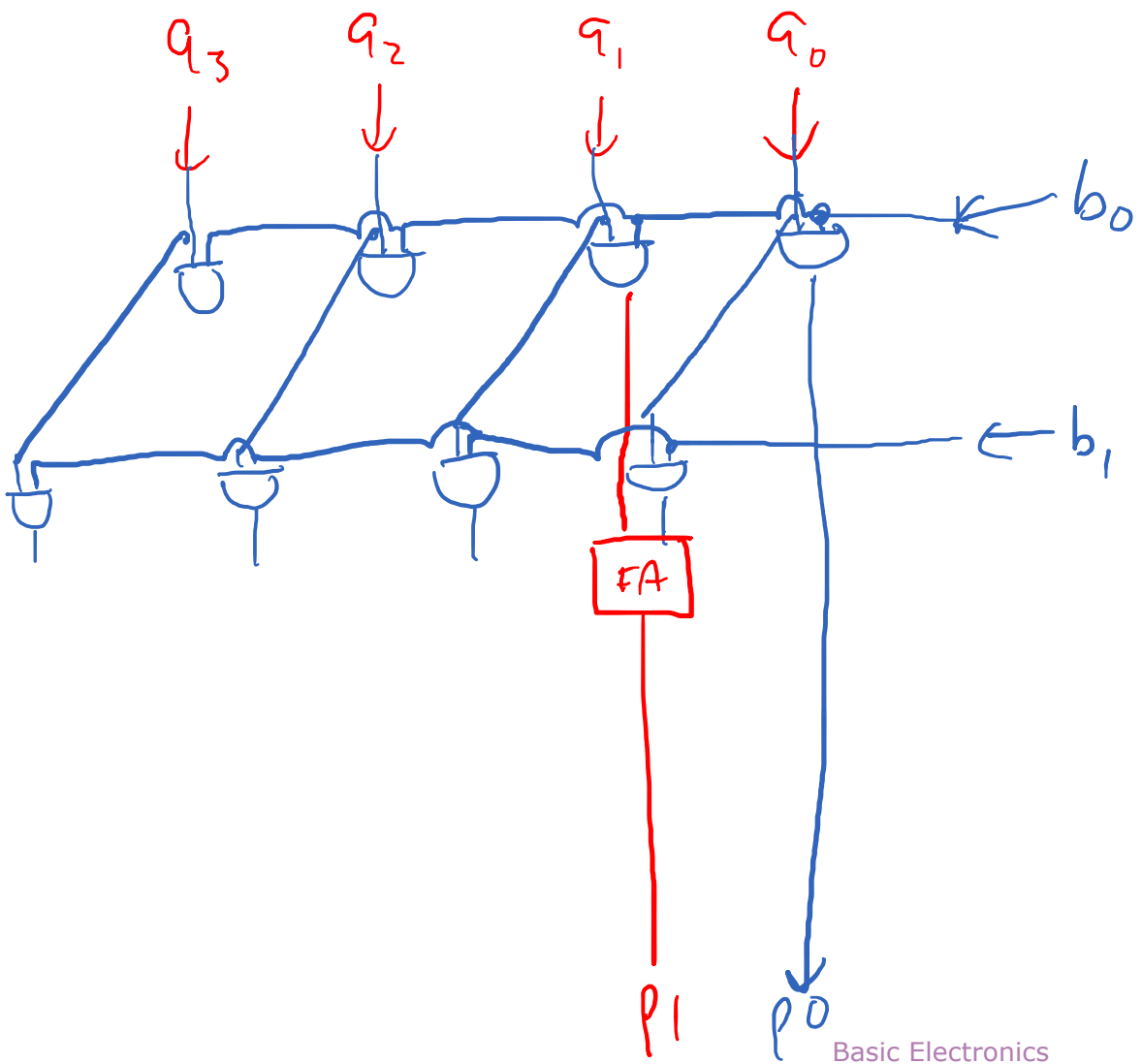
Binary Multiplication

```
    6
×   3
```

```
    0110
×   0011
```

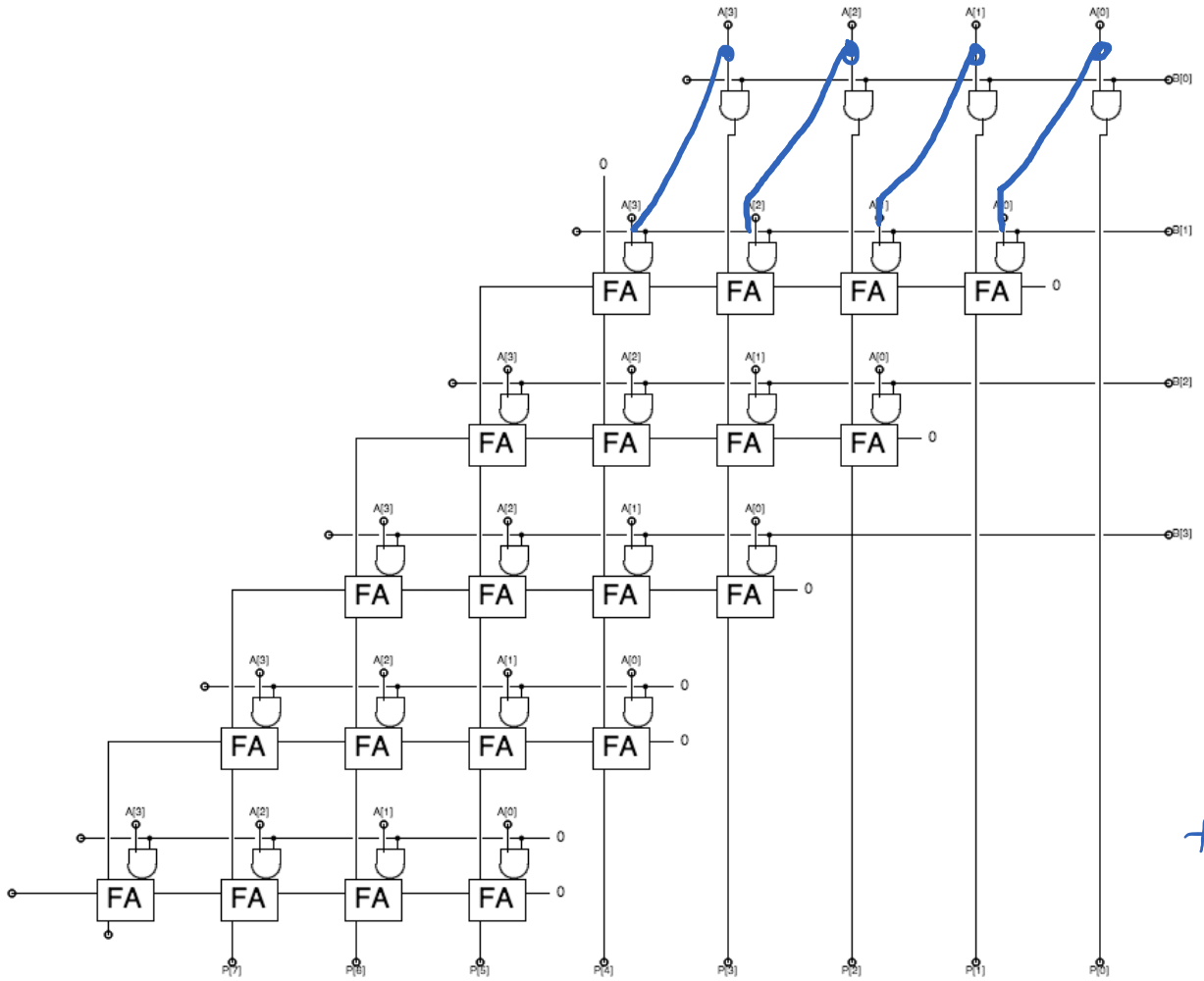
```
                                  a3  a2  a1  a0
                          ×       b3  b2  b1  b0
                        ---------------------------
                            b0a3 b0a2 b0a1 b0a0
                  +     b1a3 b1a2 b1a1 b1a0    0
                  +  b2a3 b2a2 b2a1 b2a0    0    0
                  + b3a3 b3a2 b3a1 b3a0    0    0    0
                  ---------------------------------------
                  p7   p6   p5   p4   p3   p2   p1   p0
```

Combinational Multiplication



					a3	a2	a1	a0
				×	b3	b2	b1	b0
				b0a3	b0a2	b0a1	b0a0
+			b1a3	b1a2	b1a1	b1a0	0
+		b2a3	b2a2	b2a1	b2a0	0	0
+	b3a3	b3a2	b3a1	b3a0	0	0	0
p7	p6	p5	p4	p3	p2	p1	p0

Combinational Multiplication



					a3	a2	a1	a0
				×	b3	b2	b1	b0
				b0a3	b0a2	b0a1	b0a0	
+			b1a3	b1a2	b1a1	b1a0	0	
+		b2a3	b2a2	b2a1	b2a0	0	0	
+	b3a3	b3a2	b3a1	b3a0	0	0	0	
p7	p6	p5	p4	p3	p2	p1	p0	

b0a3 b0a2 b0a1 b0a0

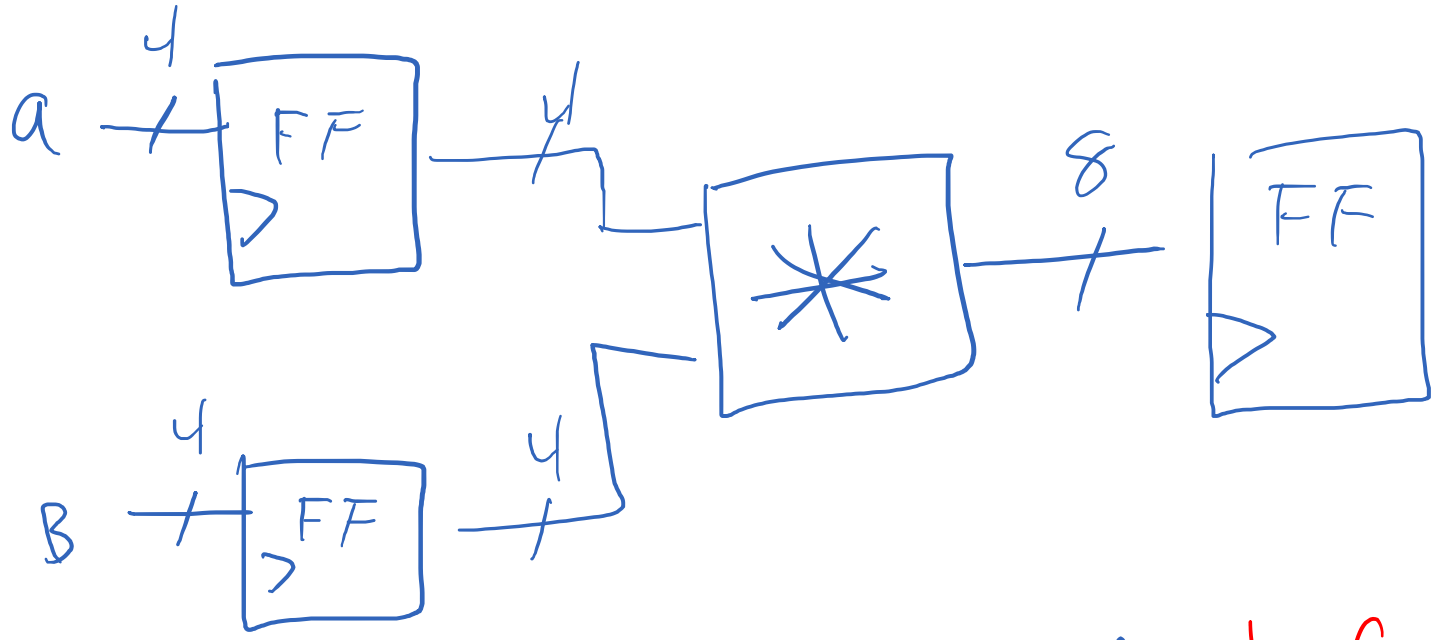
\+ b1a3 b1a2 b1a1 b1a0 0

q_4 q_3 q_2 q_1 q_0

\+ b2a3 b2a2 b2a1 b2a0

r_5 r_4 r_3 r_2 r_1 r_0

Problems with Combinational Multiplication

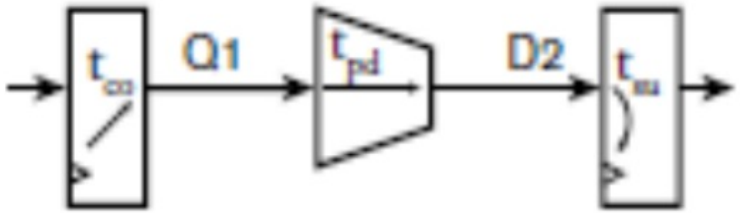


not ready before clock edge

⇒ Setup time violation

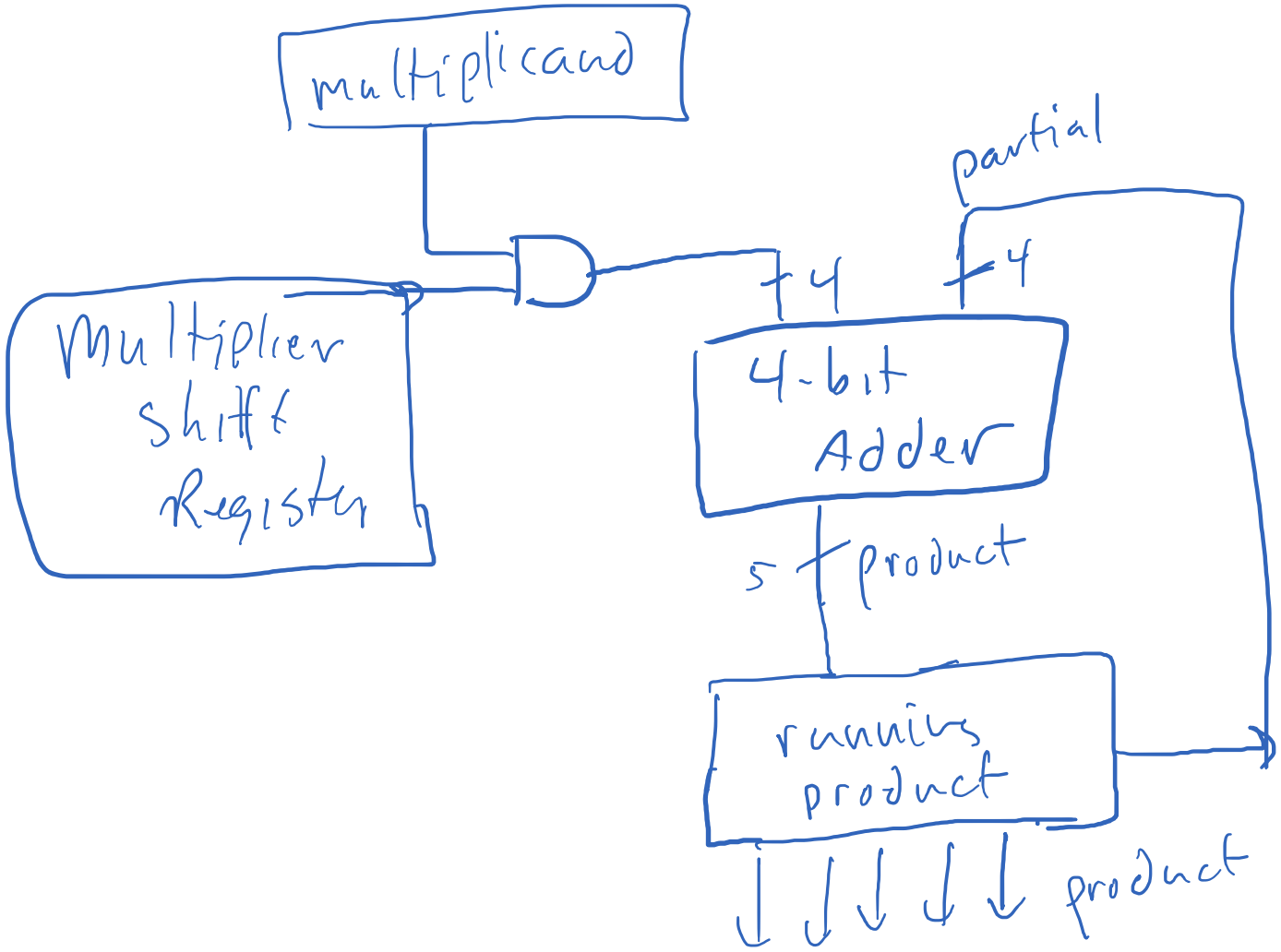
Problems with Combinational Multiplication

- Setup Time Violations!



Sequential Multiplication

sedo Demo

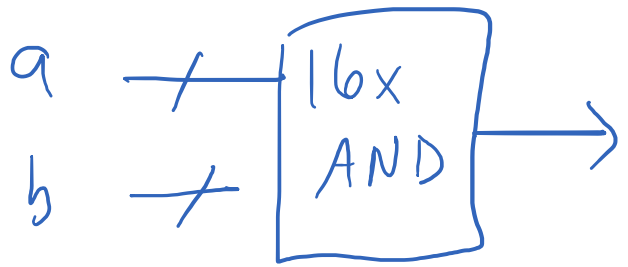


					a3	a2	a1	a0
				×	b3	b2	b1	b0
				b0a3	b0a2	b0a1	b0a0	
+			b1a3	b1a2	b1a1	b1a0	0	
+		b2a3	b2a2	b2a1	b2a0	0	0	
+	b3a3	b3a2	b3a1	b3a0	0	0	0	
p7	p6	p5	p4	p3	p2	p1	p0	

clk 1

	b0a3	b0a2	b0a1	b0a0
+	0	0	0	0

Hybrid Multiplication



"terms" → "shift" →

					a3	a2	a1	a0
				×	b3	b2	b1	b0
				b0a3	b0a2	b0a1	b0a0
+			b1a3	b1a2	b1a1	b1a0	0
+		b2a3	b2a2	b2a1	b2a0	0	0
+	b3a3	b3a2	b3a1	b3a0	0	0	0
p7	p6	p5	p4	p3	p2	p1	p0

partial products

partial products

partial products

clock 1

clk 2

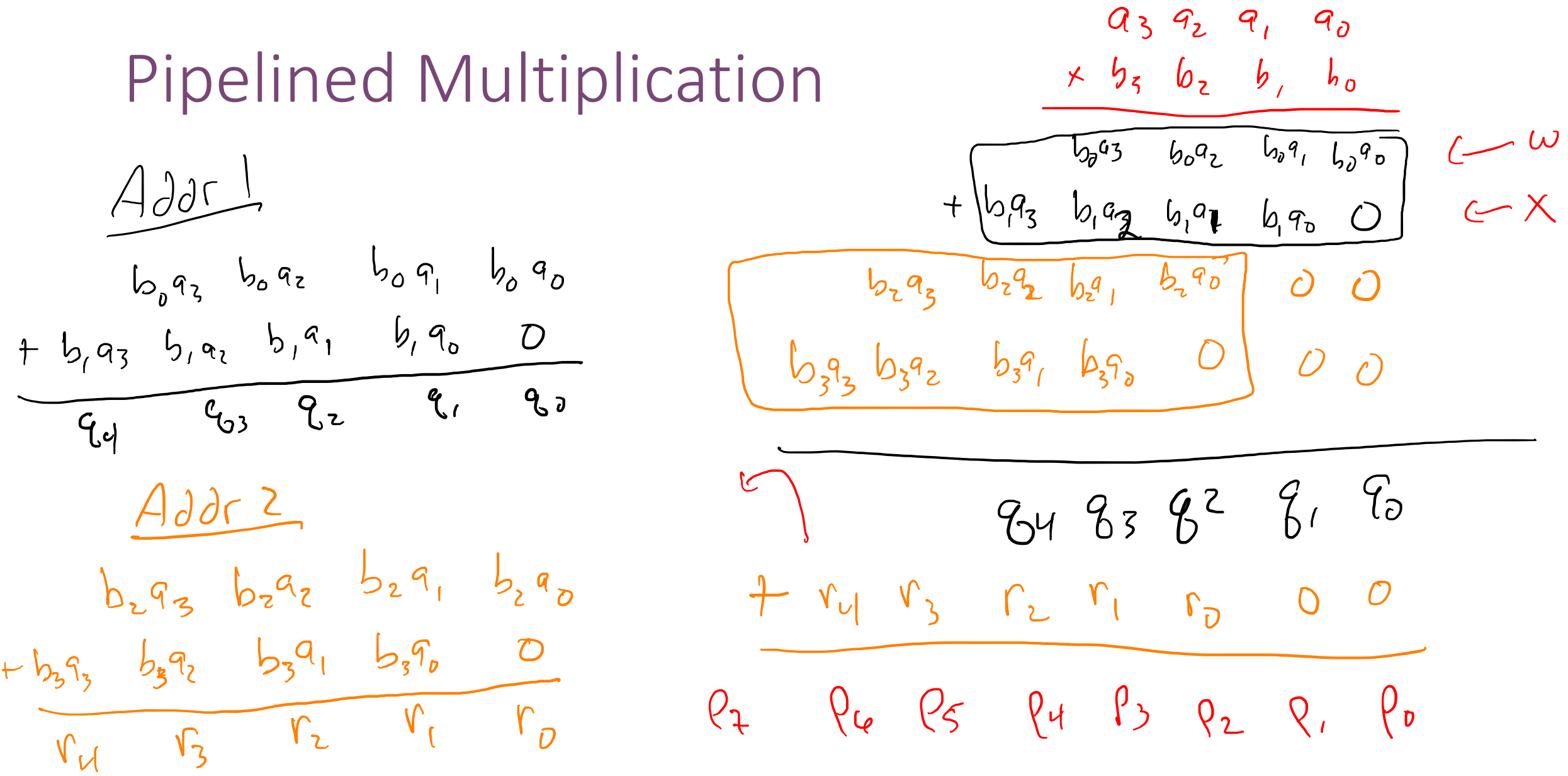
clk 3

w + x = q

y + z = r

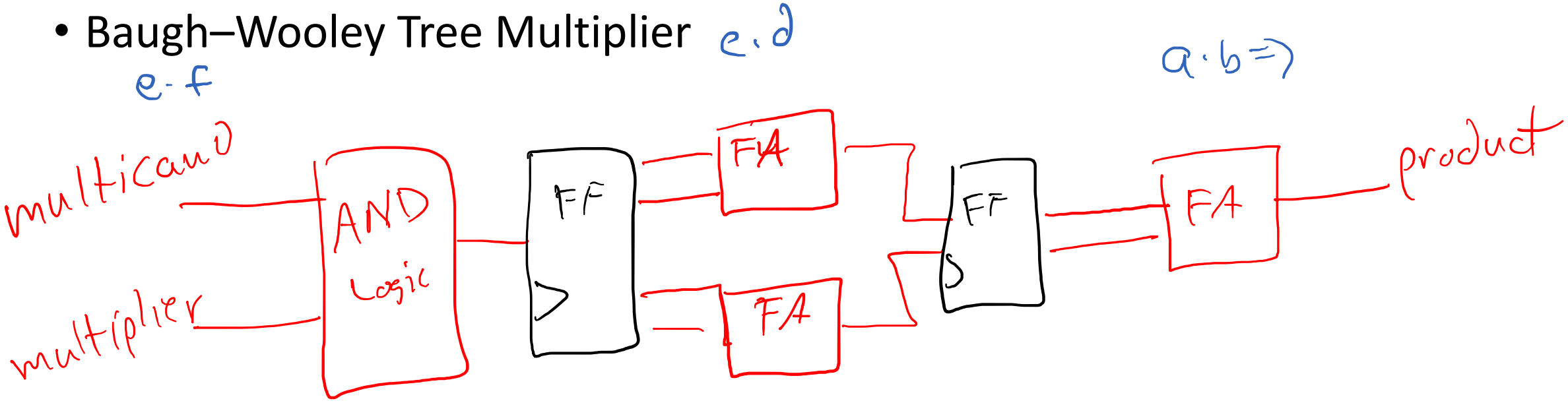
q + r = product

Pipelined Multiplication



Even More Multipliers

- Wallace Tree Multiplier
- Dadda Tree Multiplier
- Baugh–Wooley Tree Multiplier



Next Time

- Binary Division
- More Pipelining