VE270
Recitation Class for Week 11
RTL Design

HAO Zhiyu

2019.11.20.

Outline

1. RTL design
2. Lookahead adder
3. Incrementer
4. Comparator
5. Multiplier

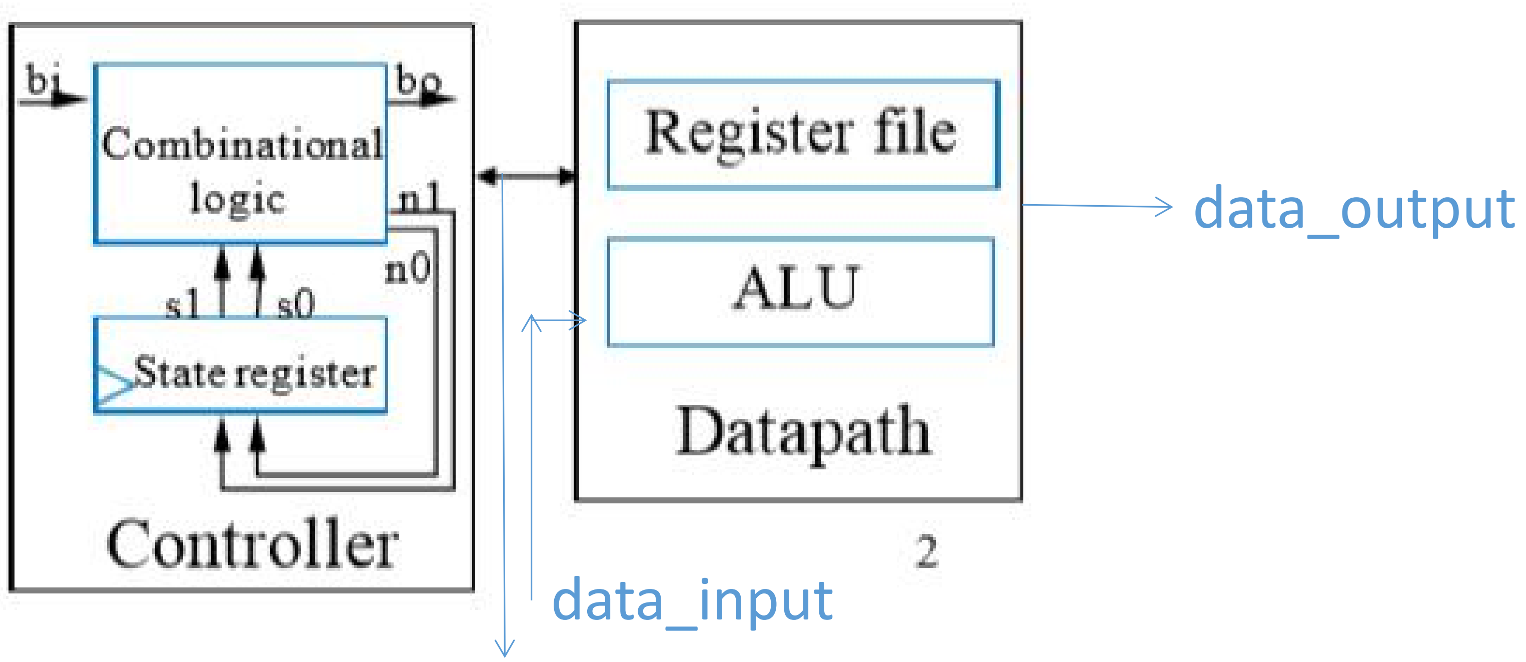
RTL design

- Controllers(FSM):

  produce simple output and control signals for datapath

- Datapath:

  Manipulate the data according to the controllers’ command



RTL steps

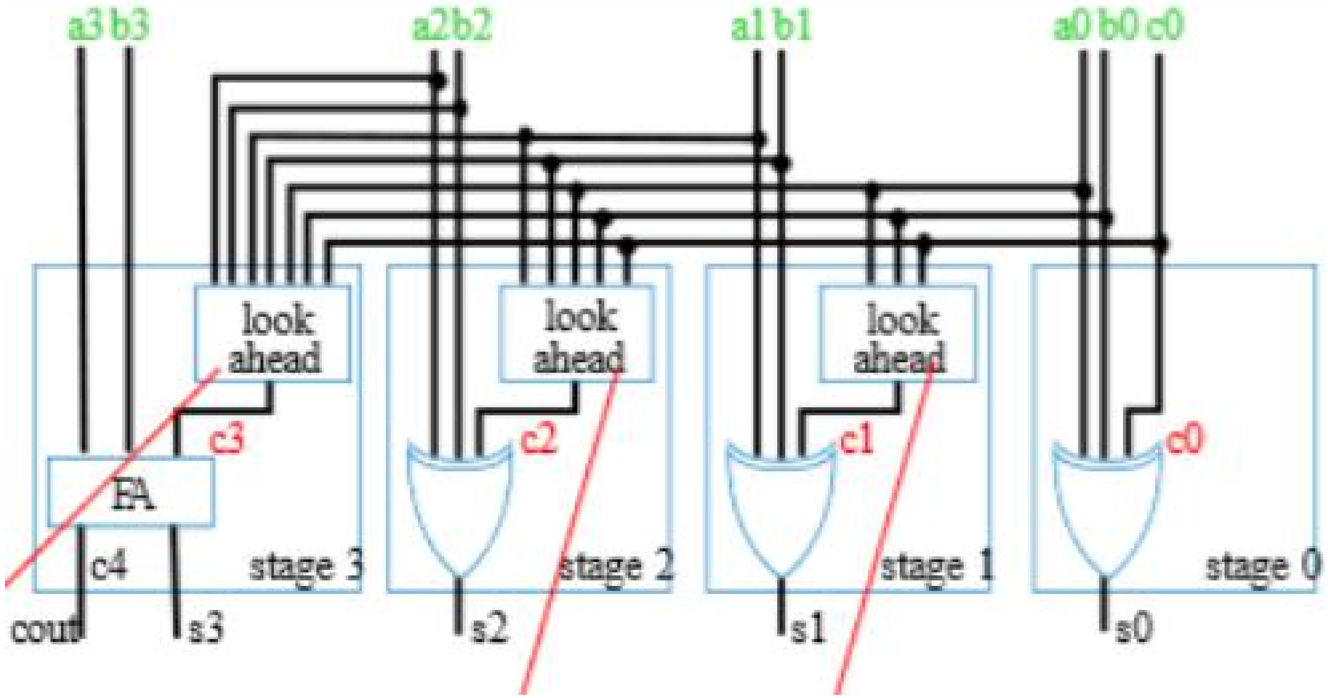
- A high level state machine
- Datapath
- Connect the datapath to a controller
- Derive the (controller's) FSM from the high level state machine

Make sure you understand
the examples provided in
the lecture notes!!!!
Vending machine,
Laser
Bus interface

Lookahead adder

- Original adder: slow, need to wait for the carry
- Lookahead logic:

  the circuit inside is too complicated, can it be better?



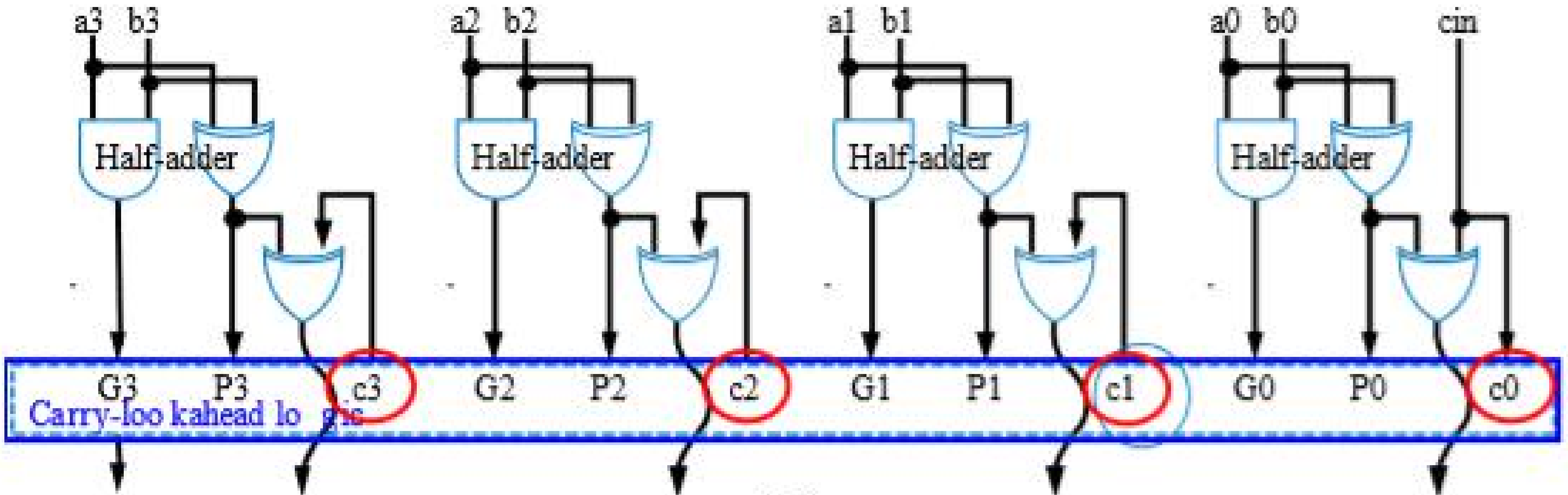
(Fast Adder)

PG Lookahead

– *Propagate*: $P = a \oplus b$
– *Generate*: $G = ab$

$Cout = G + Pc$

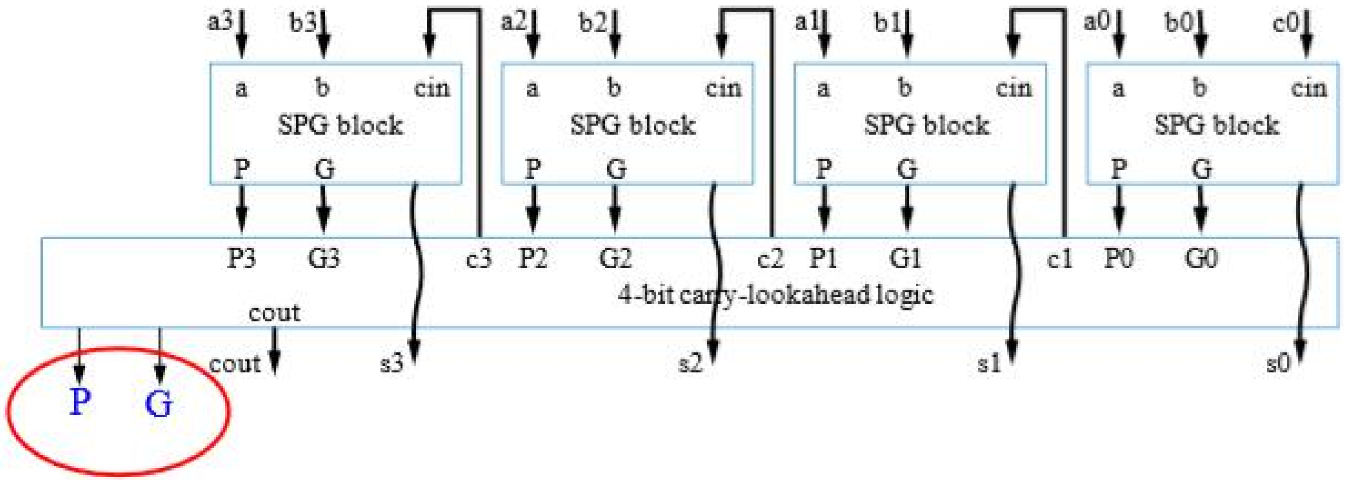
- $c1 = a0b0 + (a0 \oplus b0)c0 = G0 + P0c0$
- $c2 = a1b1 + (a1 \oplus b1)c1 = G1 + P1c1$
- $c3 = a2b2 + (a2 \oplus b2)c2 = G2 + P2c2$



(b)

Carry-Lookahead Adder

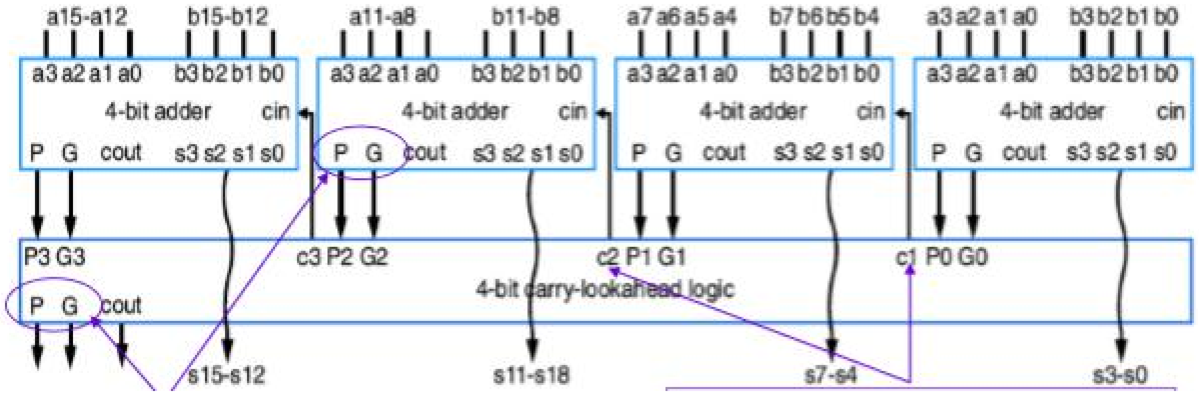
- 4 gate delay to compute result
- 3 to produce cout and blue PG
- High level design:



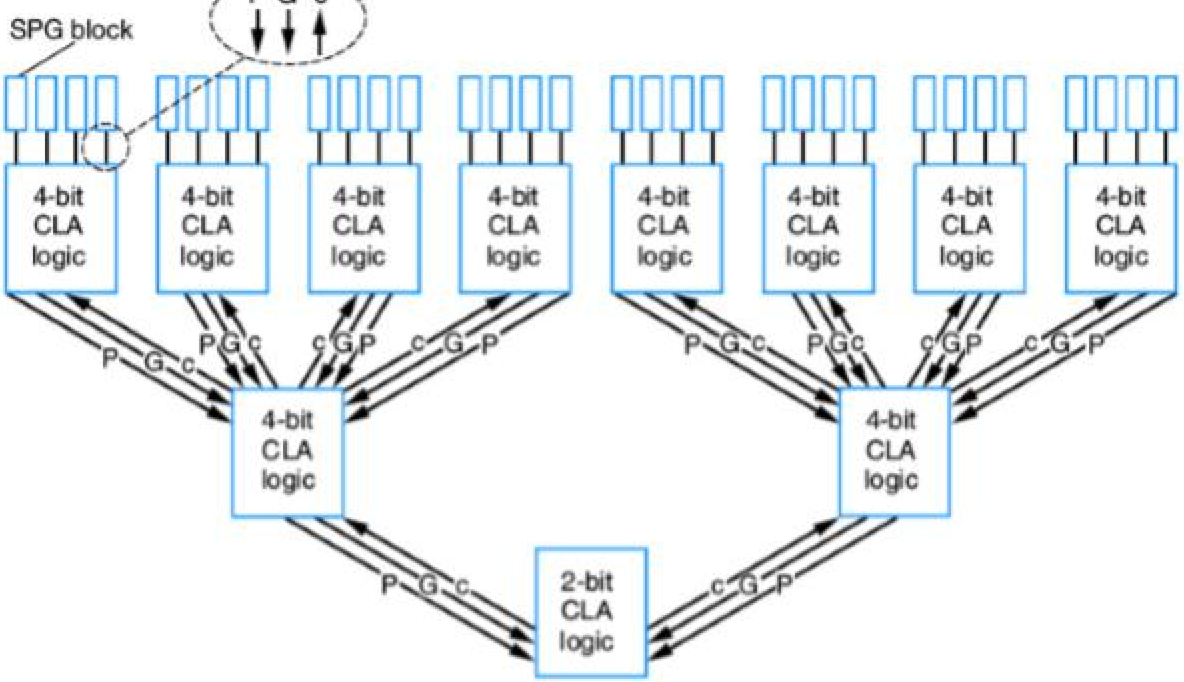
P = P3P2P1P0

G = G3+P3G2+P3P2G1+P3P2P1G0

High level View



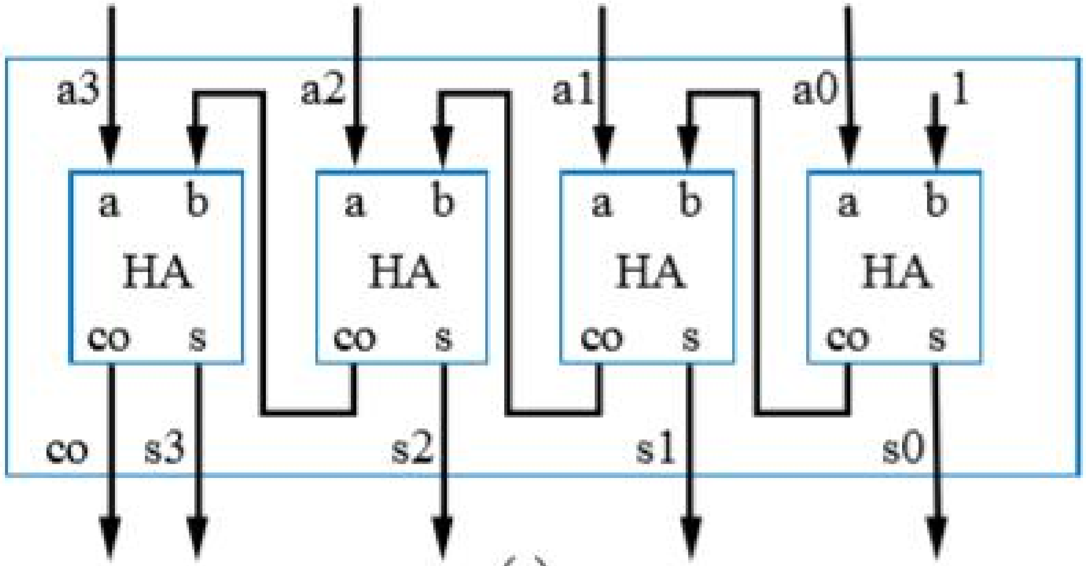
(Sum/propagate/generate block)



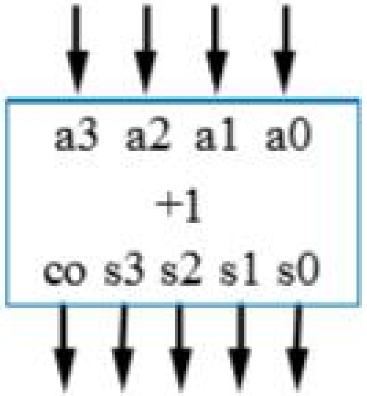
What’s the delay?

Incrementer

- Traditional: derive equation from truth table
- Slower but simpler: use half adders



(a)



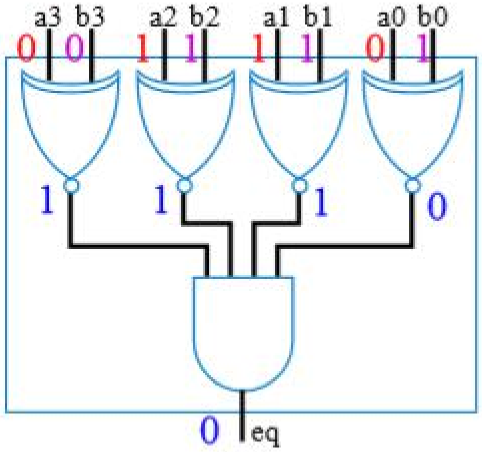
(b)

Inputs				Outputs				
a3	a2	a1	a0	c0	s3	s2	s1	s0
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0	0
0	1	0	0	0	0	1	0	1
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	1	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	1	0
1	0	1	0	0	1	0	1	1
1	0	1	1	0	1	1	0	0
1	1	0	0	0	1	1	0	1
1	1	0	1	0	1	1	1	0
1	1	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0

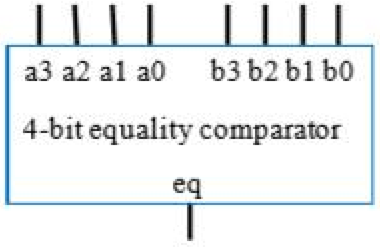
- c0 = a3a2a1a0
- ...
- s0 = a0'

Comparator

- Equality comparator: output 1 if equal
- Magnitude comparator
  Indicate if A>B, A=B, or A<B

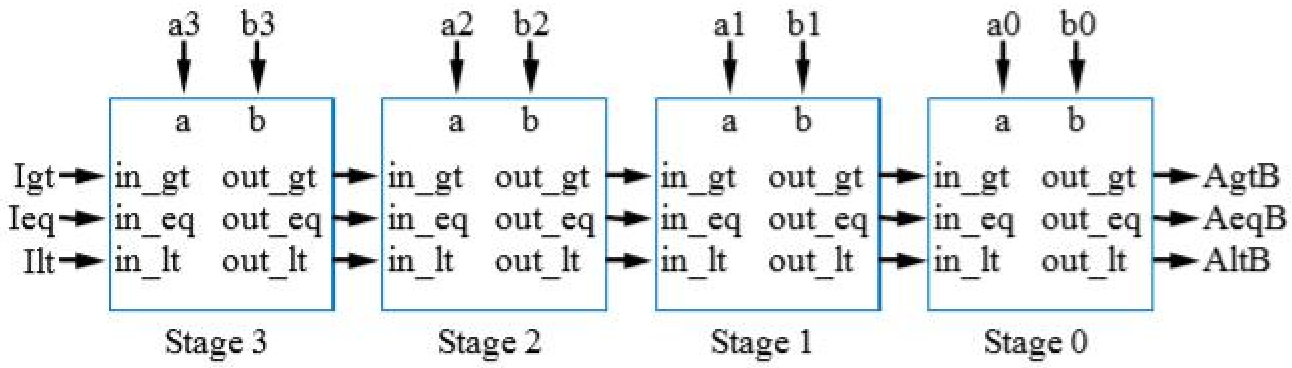


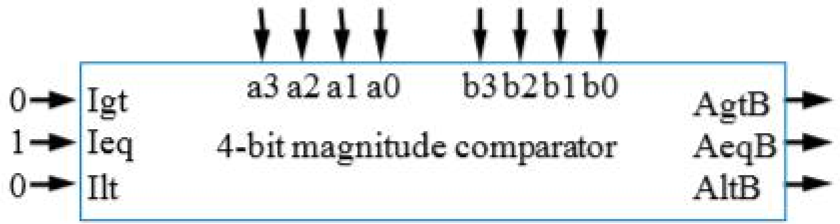
(a)



(b)

Magnitude comparator

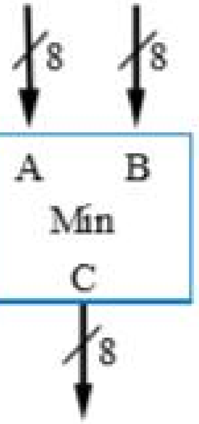




(b)

Each stage:

- out_gt = in_gt + (in_eq * a * b')
  - A>B (so far) if already determined in higher stage, or if higher stages equal but in this stage a=1 and b=0
- out_lt = in_lt + (in_eq * a' * b)
  - A<B (so far) if already determined in higher stage, or if higher stages equal but in this stage a=0 and b=1
- out_eq = in_eq * (a XNOR b)
  - A=B (so far) if already determined in higher stage and in this stage a=b too
- Simple circuit inside each stage, just a few gates (not shown)



(b)

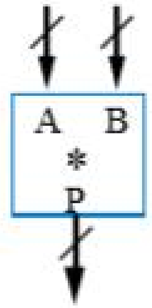
Multiplier

- Mimics hand calculating
- Derive equation for pp1~pp4

pp1={b0a3,…,b0a0};

pp2={b1a3,…,b1a0}+pp1;

…



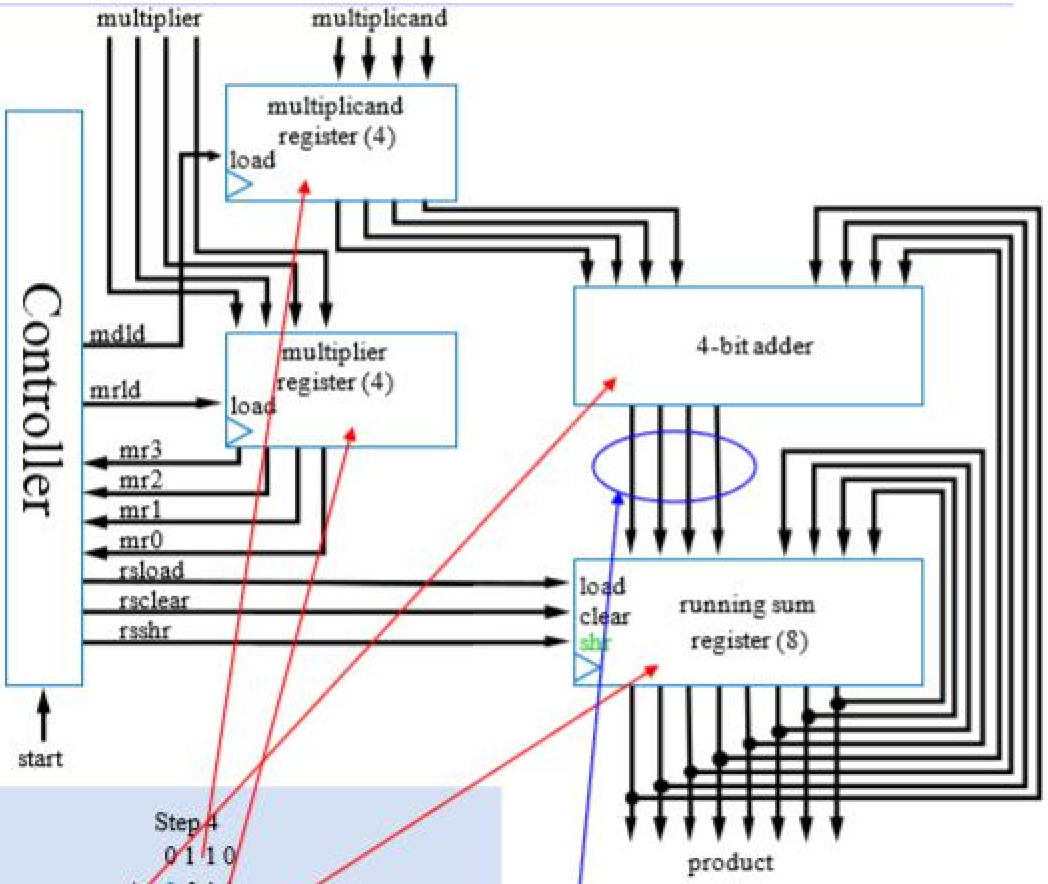
Block symbol

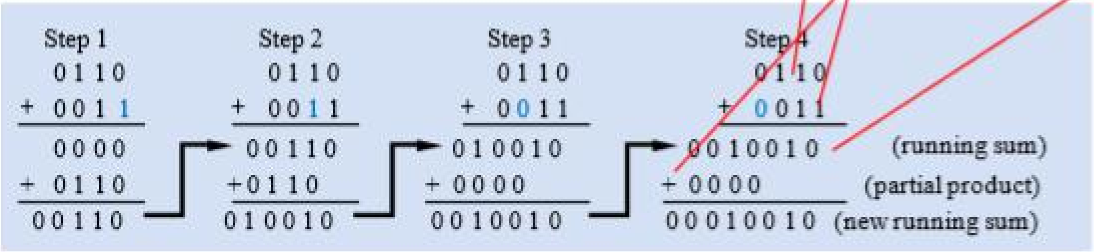
				a3	a2	a1	a0	
			x	b3	b2	b1	b0	
				b0a3	b0a2	b0a1	b0a0	(pp1)
			b1a3	b1a2	b1a1	b1a0	0	(pp2)
		b2a3	b2a2	b2a1	b2a0	0	0	(pp3)
+	b3a3	b3a2	b3a1	b3a0	0	0	0	(pp4)
p7	p6	p5	p4	p3	p2	p1	p0	

Smaller Multiplier

- Running sum

- Design circuit that computes one partial product at a time, adds to running sum
  - Note that shifting running sum right (relative to partial product) after each step ensures partial product added to correct running sum bits





What about carry?

What if 2's complement?

- Thank you！