# Mealy machines (6.3)

A Mealy machine is one where the outputs depend directly on the inputs. That has significantly more implications than you’d think.

* First of all, it means that the outputs will change soon after the inputs change and won’t wait for the next rising edge of the clock. This can be handy (fast response time) and annoying (recall our traffic light example—you really don’t want the light changing from green to yellow to green again).
* Secondly it means that we can sometimes reduce the number of states needed.
* It also means that the outputs need to show up on edges (arcs) of the state diagram rather than in the states. (Why is that?)

**Problem**:

Using a Moore machine which takes one input X and generates one output Z, design it so that Z goes high iff X has been high for the last two cycles.

Now solve the same problem with a Mealy machine.

Now, let’s look at the timing diagram associated with each machine.

Moore:

CLK

X

State

Z

Mealy:

CLK

X

State

Z

# Faster adders (3.4)

Ripple-carry adders are slow.

* How many gate delays do we have for a 4-bit ripple-carry adder (in the worst case)?
* For a 32-bit RCA?

They are however pretty small.

* How many gates total for a 32-bit RCA?

F

A

a3

c

o

s3

b3

F

A

a0

b0

ci

F

A

a2

s2

s1

s0

b2

F

A

a1

b1

*a*

One other option is that we could “just” write out the truth table for the adder (9 inputs for a 4-bit adder) and write the sum-of-products for the 4-bit adder.

* What would be our gate delay?
* How many gates would there be (this one is hard and we can’t really figure it out easily, but guess).

Pretty clearly 32-bit adders done as sum-of-products would be huge (we’ll discuss how huge later). And if we were limited to 2-input gates, things get crazy quickly.

## Start on look-head

In any case, let’s try something smarter. We could add some logic that figures out if there will be a carry in. *That* in theory could be 2-level logic. But as you can see, computing “c3” involves looking at c0, a0, b0, a1, b1, a2, and b2. Which sounds like our sum-of-products adder. And doing a 32-bit one seems crazy.

F

A

a3

c

o

s3

b3

F

A

a0

b0

ci

F

A

a2

s2

s1

s0

b2

F

A

a1

b1

*a*

F

A

c4

c3

c2

s3

s2

stage 3

stage 2

c1

s1

stage 1

c0

s0

c0

b0

b1

b2

b3

a0

a1

a2

a3

stage 0

c

out

look

ahead

look

ahead

look

ahead

Notice – no rippling of carry

That said, we could just limit ourselves to a 4-bit adder like this and then ripple the 4-bit adders (as shown below). That might be an interesting compromise between size and speed.

a3

a2

a1

a0

b3

s3

s2

s1

s0

c

out

c

out

cin

b2

b1

b0

4-bit adder

a3

a2

a1

a0

b3

s3

s2

s1

s0

s11-s8

s15-s12

a15-a12

b15-b12

a11-a8

b11-b8

c

out

cin

b2

b1

b0

4-bit adder

a3

a2

a1

a0

b3

s3

s2

s1

s0

c

out

s7

s6

s5

s4

cin

b2

b1

b0

a7

a6

a5

a4

b7

b6

b5

b4

4-bit adder

a3

a2

a1

a0

b3

s3

s2

s1

s0

s3

s2

s1

s0

c

out

cin

b2

b1

b0

a3

a2

a1

a0

b3

b2

b1

b0

4-bit adder

But we’d like to do better than that. This marginally might speed up things (at the cost of more logic) but it’s only a marginal improvement.

Assume in a RCA each stage takes 1 unit of time. And assume this 4-bit “lookahead” version takes 2 units of time (which is probably reasonable if a bit optimistic). A 16-bit RCA will then take 16 units of time (1 for each of the 16 stages), while this hybrid scheme will take 8 units of time (2 for each of the 4 4-bit stages). So twice as fast. Which is great, but maybe we can do better…

(

a

)

b3

a3

s3

b2

a2

s2

b1

a1

s1

b0

a0

s0

1

1

0

0

1

carries:

c4 c3 c2 c1 c0

B:

A:

+

+

c

out

cin

1

1

1

1

1

+

0

1

0

1

1

+

1

0

0

1

1

+

c1

c0

b0

a0

if a0

x

or b0 = 1

then c1 = 1 if c0 = 1

(call this P: Propagate)

if a0b0 = 1

then c1 = 1

(call this G:Generate)



“Bad” lookahead

F

A

c4

c3

c2

s3

s2

stage 3

stage 2

c1

s1

stage 1

c0

s0

c0

b0

b1

b2

b3

a0

a1

a2

a3

stage 0

look

ahead

look

ahead

look

ahead

c

out

After plugging in:

c1 = G0 + P0c0

c2 = G1 + P1c1 = G1 + P1(G0 + P0c0)

c2 = G1 + P1G0 + P1P0c0

c3 = G2 + P2c2 = G2 + P2(G1 + P1G0 + P1P0c0)

c3 = G2 + P2G1 + P2P1G0 + P2P1P0c0

cout = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0c0

Much simpler than the “bad” lookahead

*a*

*a*

C

a

r

r

y

-loo

k

ahead lo

g

ic

G3

a3

b3

P3

c3

c

out

s3

G2

a2

b2

P2

c2

s2

G1

a1

b1

P1

c1

s1

G0

a0

b0

cin

P0

c0

s0

(

b

)

Half-adder

Half-adder

Half-adder

Half-adder



## Analysis of our 4-bit CLA compared to a RCA

Performing a fair comparison between two architectures can be surprisingly difficult. The difficulty lies in what assumptions we make. Let’s go through two models.

## #1 Gates-are-gates model

Here we’ll charge just as much for a 20-input gate as a 2-input gate. This is of course very unrealistic, but has the advantage of being easy. So let’s count total number of gates in both and worst-case delay (again, assuming all gates are equal).

|  |  |  |
| --- | --- | --- |
| **Adder type (4-bit)** | **RCA** | **CLA** |
| **Gate count** |  |  |
| **Gate delays** |  |  |

## #2 “Reasonable” gate scaling

Here for *count* we’ll charge each gate an amount equal to the number of inputs it has. So a 3-input OR gate costs 3 and a 2-input XOR gate costs 2. But for *delay* we’ll charge log2(inputs). So 1 input is free, 2-inputs costs 1, 3 inputs is 1.5 (okay, 1.58 but let it go), 4 inputs is 2, 5 inputs is 2.25 (2.3) and 6 inputs is 2.5 (2.6).

|  |  |  |
| --- | --- | --- |
| **Adder type (4-bit)** | **RCA** | **CLA** |
| **Gate-input count** |  |  |
| **Log2(gate-input) delays** |  |  |

What is your conclusion about these two schemes?

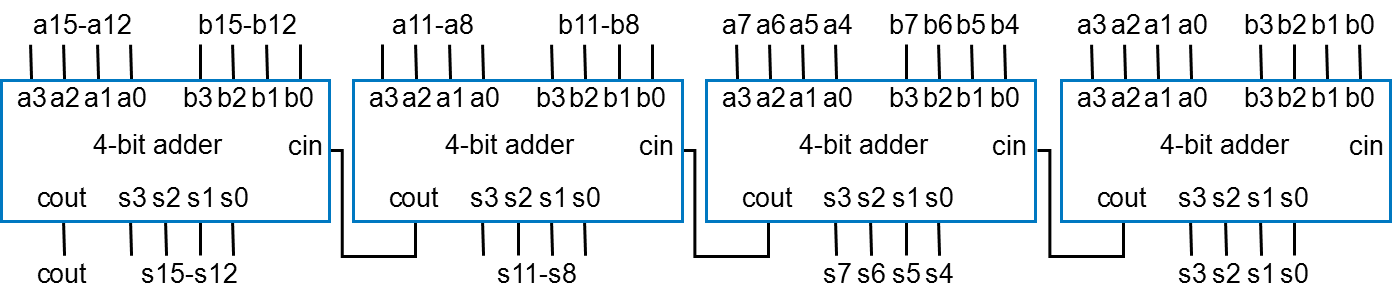
# Bigger than 4 bits?

OK, it seems that this will start to get ugly past 4 bits. How could we build a 16-bit adder?

4-bit adder

## Ripple-carry (again)

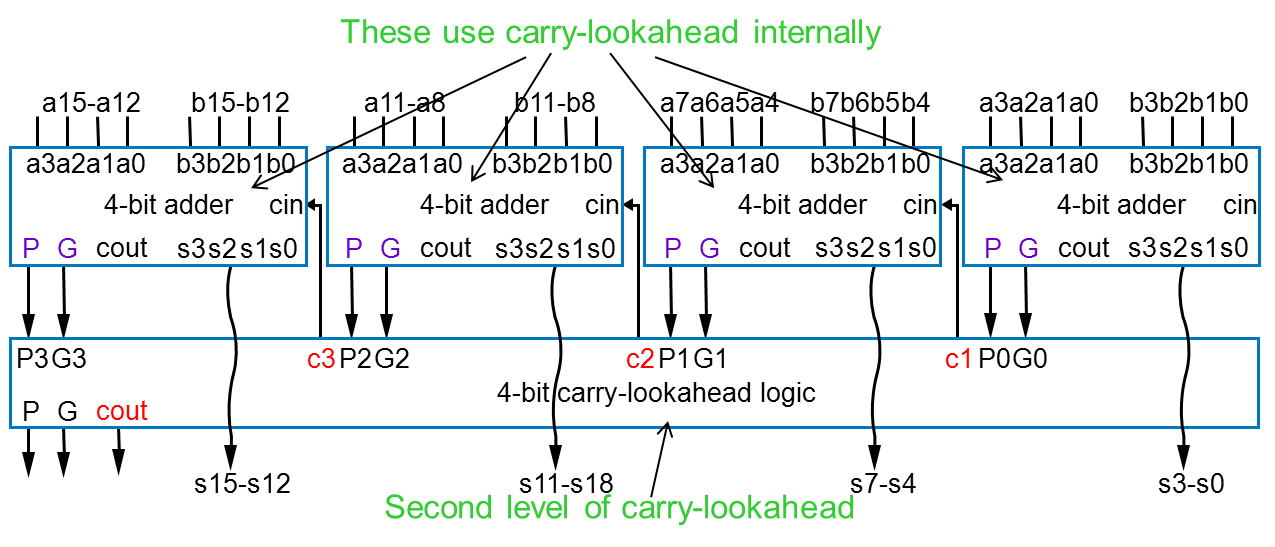
One obvious way is to just ripple the carry. What is the delay and count for each of the two schemes?



|  |  |  |
| --- | --- | --- |
| **Adder type (16-bit)** | **RCA** | **CLA** |
| **Gate count** |  |  |
| **Gate delays** |  |  |

|  |  |  |
| --- | --- | --- |
| **Adder type (16-bit)** | **RCA** | **CLA** |
| **Gate-input count** |  |  |
| **Log2(gate-input) delays** |  |  |

## Carry-lookahead (again)

Or we could try to get tricky. Obviously (?), we could use the lookahead logic again.

|  |  |
| --- | --- |
| **Adder type (16-bit)** | **CLA** |
| **Gate count** |  |
| **Gate delays** |  |

|  |  |
| --- | --- |
| **Adder type (16-bit)** | **CLA** |
| **Gate-input count** |  |
| **Log2(gate-input) delays** |  |

# Of course, we have another option: 2-level logic

  
It turns out counting the number of gates for the 2-level adder is pretty hard. But here’s a graph (figure 4.24 from the text) that shows the number of *transistors* needed. It’s a case of exponential growth, so a 32-bit adder needs something like 200-billion transistors (a modern CPU has about 1 billion)... In any case, or 4-bit 2-level adder needs about 100 gates.

What do you estimate delay will be for a 4-input adder using our log model for delay? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_