EENG 4250/ENAS 8750
Introduction to VLSI System Design

Rajit Manohar

Computer Systems Lab

http://csl.yale.edu/~rajit

Fall 2025

Yale

Datapath and control circuits

Normally a design gets partitioned into:

- regions that manipulate data (the *datapath*)
- regions that control data motion (the *control*)

Why?

Control

Standard design procedure:

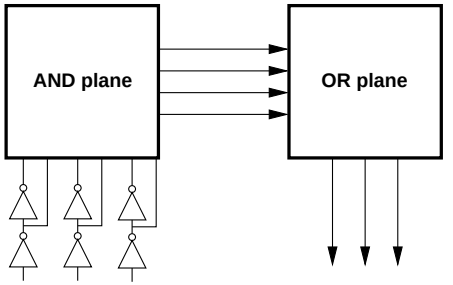
- design a state machine
- state assignment
- minimize logic equations

... and then, do the layout!

- ensure the layout is fast enough
- predictable timing is important

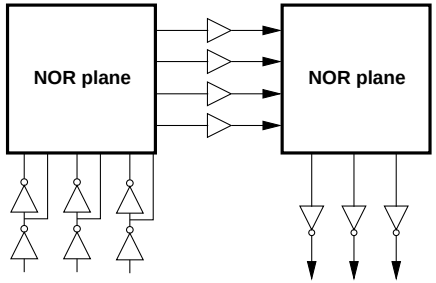
Programmable logic array (PLA)

PLA (programmable logic array)



Programmable logic array (PLA)

NOR-NOR PLA



Yale

Systematic datapath design

- Start with high-level specification
  e.g. processor: the ISA
- Pick storage elements and signals
  e.g. processor: registers, memory, ...
- Translate specification
  determine operations on storage elements
  (RTL, register transfer language)

Yale

Systematic datapath design

More steps...

- Pick computation blocks
  processor: alu
- Determine connections
  data-dependencies among different blocks
  i.e., the datapath
- Determine control inputs to datapath blocks

... and then put everything together.

Cross-cutting issue: clocking strategy
when do storage elements get updated?

Yale

Design example

Example: (unsigned integers)

```
x = 0;
for (i=1; i <= N; i++)
   x = x + i;
```

Here `N` is an input and we should produce the result that is stored in `x`.

Next: make the specification precise in terms of signals and clocks.

Observation: finite hardware storage

Yale

Design example

Better specification of I/O behavior:

- Input `N` arrives on a bus that is 4 bits wide and `N` is non-zero
- Output `x` is 7 bits wide
- Output `xdone` (1 bit) is set to 1 when the `x` output holds the correct data
- `xdone` stays high for 1 cycle, after which the next `N` input is read and the computation proceeds as before
- Signals change at the positive edge of the clock

Yale AVLSI

Pick storage elements

There is normally a choice here. For our example:

- `i`: 4 bit storage element
- `N`: 4 bit storage element
- `x`: 7 bit storage element
- `xdone`: 1 bit output signal

Another option: rewrite the loop and eliminate one of the storage elements!

```
x = 0;
for (i=N; i >= 1; i--)
   x = x + i;
```

Yale AVLSI

Translate specification

Pick states, and determine operations in each state.
A systematic way: translate program using gotos...

```
initial:
   xdone = 0; x = 0; i = N;
   goto loop;
loop:
   xdone = 0; x = x + i; i = i - 1;
   if (i == 0) goto done;
   else goto loop;
done:
   xdone = 1;
   goto initial
```

Yale AVLSI

Translate specification

Idea:

- Each label is a state
- Advance from one state to the next every cycle
- End of program fragment at each label is a `goto`
  - all paths lead to a goto!
  - no intervening labels

RTL: normally uses "<–" for assignment

Yale AVLSI

Computation blocks

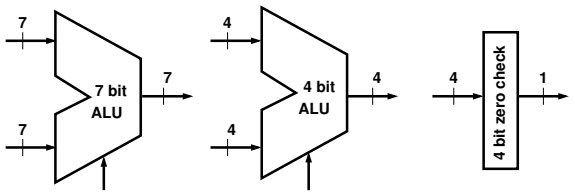
Operations on storage elements/signals:

- `i`
  - `i <- N`, `i <- i - 1`, `i == 0`
- `x`
  - `x <- 0`, `x <- x + i`
- `xdone`
  - `xdone <- 0`, `xdone <- 1`

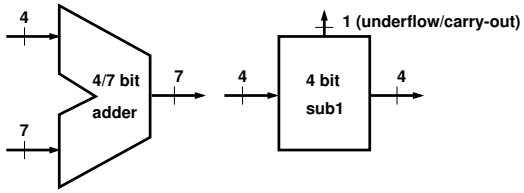
You can see why `x` and `i` are state-holding elements...

Blocks: "subtract 1," adder, compare to zero

Computation blocks



or...



Reduce, reuse, recycle...

Data dependencies

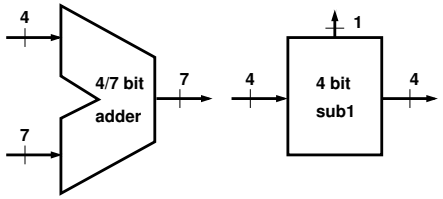
For each storage element, find out dependencies.

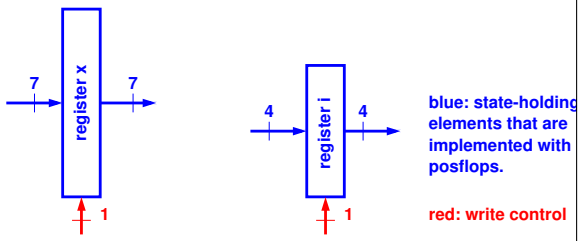
- `i`
  - set to `N`, output of `sub1` block
  - used by `sub1` block, `adder`
- `x`
  - set to `0`, output of `adder`
  - used by `adder`

(In general case, determine how computation blocks are interconnected too.)

What is the data flow?

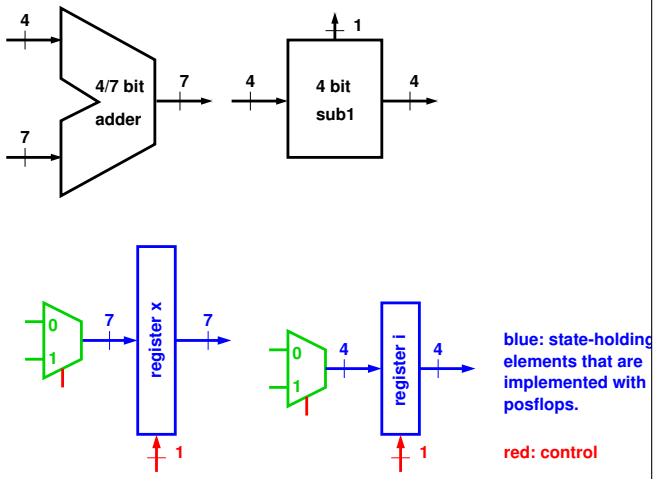
Data dependencies





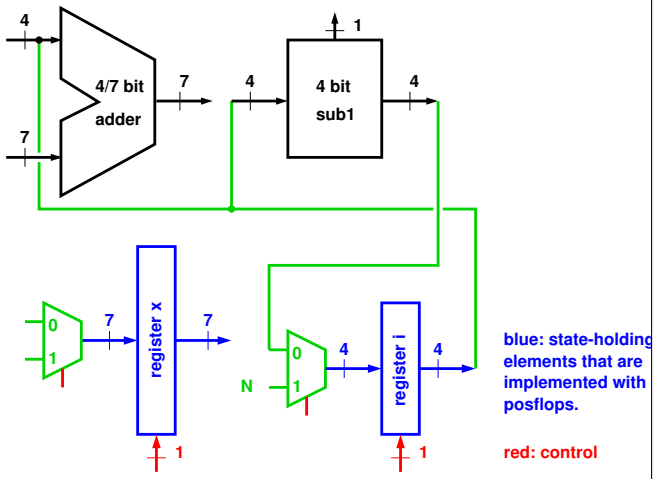
`i` can be set in two different ways...

Data dependencies



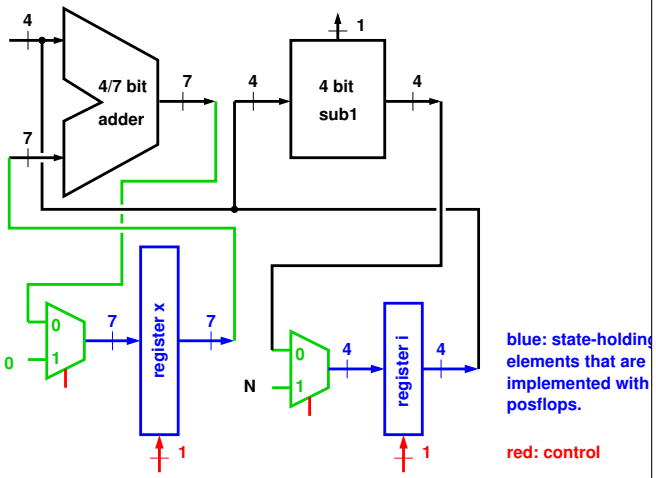
Use MUXes...

Data dependencies



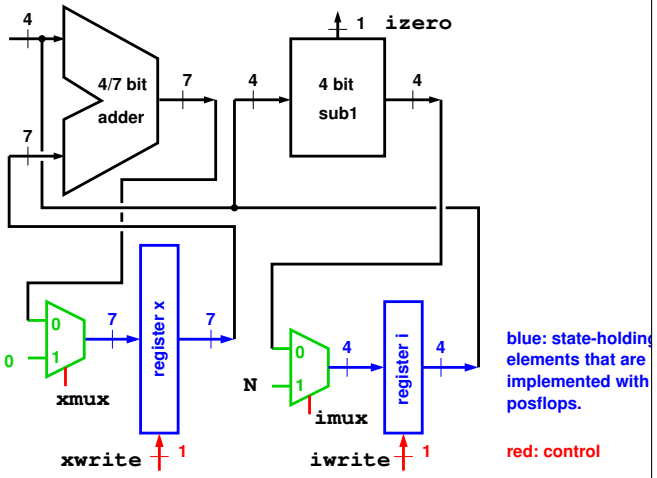
Connections for `i`.

Data dependencies



Connections for `x`.

Data dependencies



Finally: control

Control

```
initial:
   xdone <- 0;
   x <- 0;
   i <- N;
   goto loop;
loop:
   xdone <- 0;  x <- x + i;
   i <- i - 1;
   if (i == 0) goto done;
   else goto loop;
done:
   xdone <- 1;
   goto initial
```

Specify operations on data by using control signals!

Yale AVLSI

Control

```
initial:
   xdone <- 0;
   xwrite <- 1; xmux <- 1;
   iwrite <- 1; imux <- 1;
   goto loop;
loop:
   xdone <- 0; xwrite <- 1; xmux <- 0;
   iwrite <- 1; imux <- 0;
   if (izero == 1) goto done;
   else goto loop;
done:
   xdone <- 1; xwrite <- 0; iwrite <- 0;
   goto initial
```

Specify operations on data by using control signals!

Yale AVLSI

Control

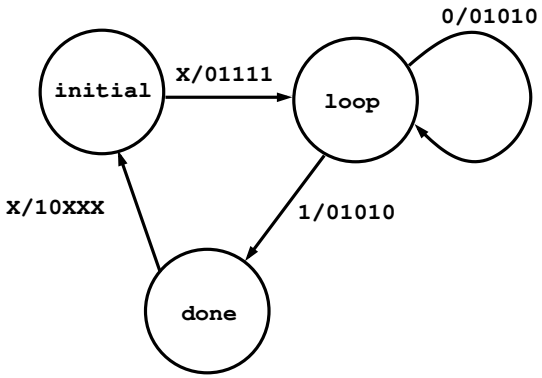
Important points:

- What about `xmux, imux` in the done state?
  ⇒ `iwrite, xwrite` are 0, so don't cares.
- Note: *each signal is set in each state!*
  ⇒ we can generate them using combinational logic!
- If this is not the case, need a state-holding element to remember what the old value of the variable was... (sometimes referred to as an *implied flop/latch*)

Yale AVLSI

Control

Format: izero/xdone xwrite xmux iwrite imux



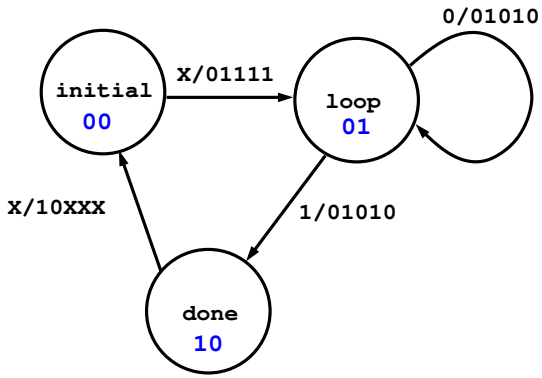
Next step: state assignment/state table

Yale AVLSI

Control

Format: izero/xdone xwrite xmux iwrite imux

s0 s1



Next: state tables, logic design

Yale

AVLSI

Control

After some logic synthesis...

- xdone = s0
- xwrite = $\overline{s0}$
- imux = $\overline{s0} \cdot \overline{s1}$
- xmux = $\overline{s0} \cdot \overline{s1}$
- iwrite = 1
- News0 = $\overline{s0} \cdot s1 \cdot izero \cdot \overline{Reset}$
- News1 = $(\overline{s0} \cdot \overline{s1} + s1 \cdot \overline{izero}) \cdot \overline{Reset}$

Yale

AVLSI