ECE 274 Digital Logic – Fall 2008

Datapath Components – Shifters, Comparators, Counters, Multipliers

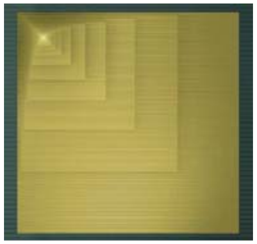
Digital Design 4.4 – 4.7

The University of Arizona
Tucson Arizona

Digital Design

Chapter 4:
Datapath Components

Slides to accompany the textbook *Digital Design*, First Edition, by Frank Vahid, John Wiley and Sons Publishers, 2007.
http://www.ddvahid.com

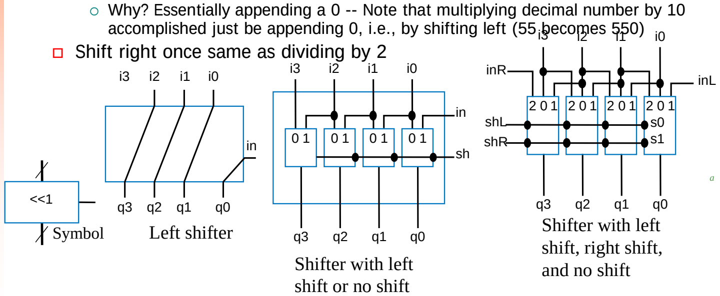


Datapath Components

Shifters

4.4

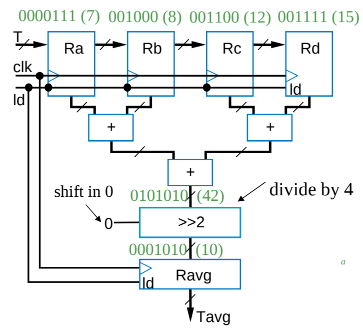
- Shifting (e.g., left shifting 0011 yields 0110) useful for:
  - Manipulating bits
  - Converting serial data to parallel (remember earlier above-mirror display example with shift registers)
  - Shift left once is same as multiplying by 2 (0011 (3) becomes 0110 (6))
    - Why? Essentially appending a 0 -- Note that multiplying decimal number by 10 accomplished just be appending 0, i.e., by shifting left (55 becomes 550)
  - Shift right once same as dividing by 2



Datapath Components

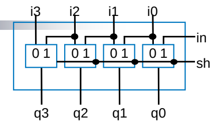
Shifter Example: Temperature Averager

- Four registers storing a history of temperatures
- Want to output the average of those temperatures
- Add, then divide by four
  - Same as shift right by 2
  - Use three adders, and right shift by two

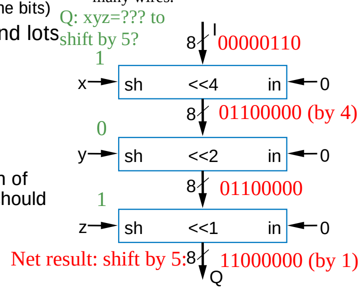


Datapath Components
Barrel Shifter

- A shifter that can shift by any amount
  - 4-bit barrel left shift can shift left by 0, 1, 2, or 3 positions
  - 8-bit barrel left shifter can shift left by 0, 1, 2, 3, 4, 5, 6, or 7 positions
    - (Shifting an 8-bit number by 8 positions is pointless -- you just lose all the bits)
- Could design using 8x1 muxes and lots of wires
  - Too many wires
- More elegant design
  - Chain three shifters: 4, 2, and 1
  - Can achieve any shift of 0..7 by enabling the correct combination of those three shifters, i.e., shifts should sum to desired amount

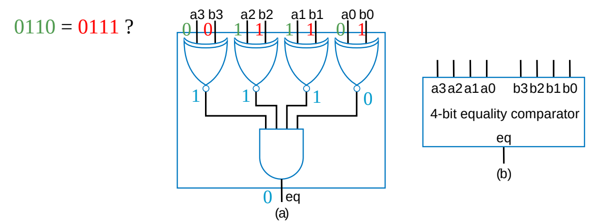


Shift by 1 shifter uses 2x1 muxes. 8x1 mux solution for 8-bit barrel shifter: too many wires.



Datapath Components
Comparators

- ***N-bit equality comparator***: Outputs 1 if two N-bit numbers are equal
  - 4-bit equality comparator with inputs A and B
    - a3 must equal b3, a2 = b2, a1 = b1, a0 = b0
      - Two bits are equal if both 1, or both 0
      - eq = (a3b3 + a3'b3') * (a2b2 + a2'b2') * (a1b1 + a1'b1') * (a0b0 + a0'b0')
    - Recall that XNOR outputs 1 if its two input bits are the same
      - eq = (a3 xnor b3) * (a2 xnor b2) * (a1 xnor b1) * (a0 xnor b0)



Datapath Components
Magnitude Comparator

- ***N-bit magnitude comparator***
  - Indicates whether A>B, A=B, or A<B, for its two N-bit inputs A and B
  - How to design?
    - Consider how compare by hand.
    - First compare a3 and b3. If equal, compare a2 and b2. And so on. Stop if comparison not equal -- whichever's bit is 1 is greater. If never see unequal bit pair, A=B.

A=1011 B=1001

1011 1001 Equal

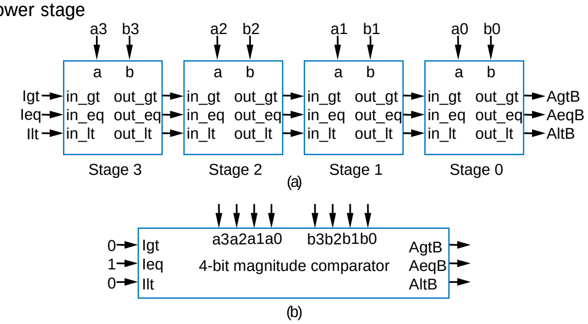
1011 1001 Equal

1011 1001 **Unequal**

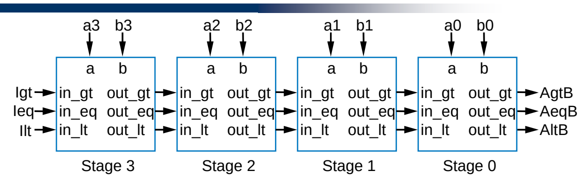
So A > B

Datapath Components
Magnitude Comparator

- By-hand example leads to idea for design
  - Start at left, compare each bit pair, pass results to the right
  - Each bit pair called a stage
  - Each stage has 3 inputs indicating results of higher stage, passes results to lower stage



Datapath Components
Magnitude Comparator



- 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)

9

Datapath Components
Magnitude Comparator Example: Minimum of Two Numbers

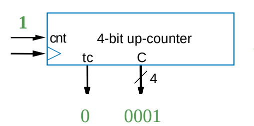
- Design a combinational component that computes the minimum of two 8-bit numbers

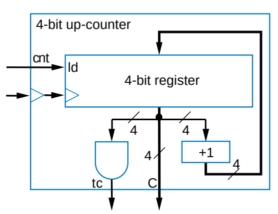
10

Datapath Components
Counters

4.6

- ***N-bit up-counter***: N-bit register that can increment (add 1) to its own value on each clock cycle
  - 0000, 0001, 0010, 0011, …., 1110, 1111, 0000
  - Note how count "rolls over" from 1111 to 0000
    - Terminal (last) count, tc, equals1 during value just before rollover
- Internal design
  - Register, incrementer, and N-input AND gate to detect terminal count

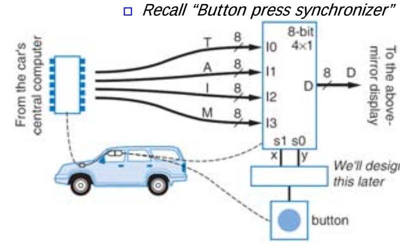


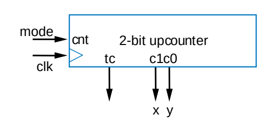


11

Datapath Components
Counter Example: Above Mirror Display

- Recall above-mirror display example from Chapter 2
  - Assumed component that incremented xy input each time button pressed: 00, 01, 10, 11, 00, 01, 10, 11, 00, ...
  - Can use 2-bit up-counter
    - Assumes mode=1 for just one clock cycle during each button press
      - *Recall "Button press synchronizer" circuit*

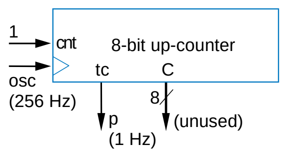




12

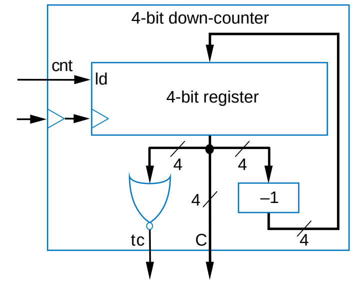
Datapath Components
Counter Example: 1 Hz Pulse Generator Using 256 Hz Oscillator

- Suppose have 256 Hz oscillator, but want 1 Hz pulse
  - 1 Hz is 1 pulse per second -- useful for keeping time
  - Design using 8-bit up-counter, use tc output as pulse
    - Counts from 0 to 255 (256 counts), so pulses tc every 256 cycles



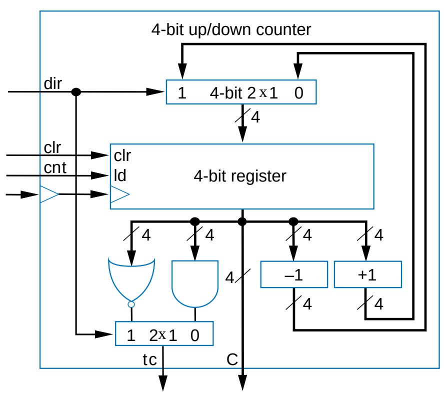
Datapath Components
Down-Counter

- 4-bit down-counter
  - 1111, 1110, 1101, 1100, …, 0011, 0010, 0001, 0000, 1111, …
  - Terminal count is 0000
    - Use NOR gate to detect
  - Need decrementer (-1) – design like designed incrementer



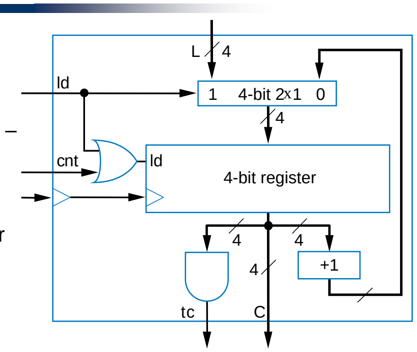
Datapath Components
Up/Down-Counter

- Can count either up or down
  - Includes both incrementer and decrementer
  - Use dir input to select, using 2x1: dir=0 means up
  - Likewise, dir selects appropriate terminal count value



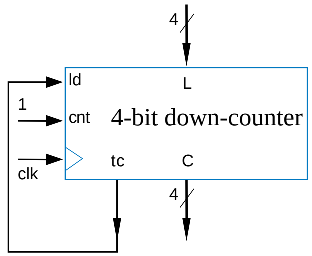
Datapath Components
Counter with Parallel Load

- Up-counter that can be loaded with external value
  - Designed using 2x1 mux – ld input selects incremented value or external value
  - Load the internal register when loading external value or when counting



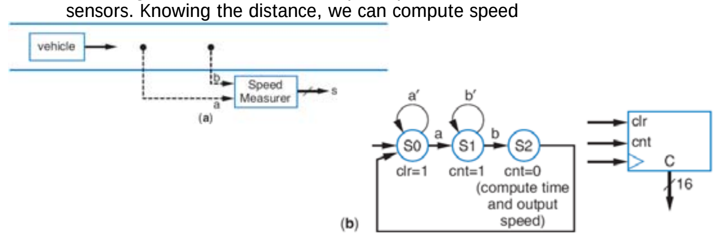
Datapath Components
Counter with Parallel Load

- Useful to create pulses at specific multiples of clock
  - Not just at N-bit counter's natural wrap-around of 2^N
- Example: Pulse every 9 clock cycles
  - Use 4-bit down-counter with parallel load
  - Set parallel load input to 8 (1000)
  - Use terminal count to reload
    - When count reaches 0, next cycle loads 8.
  - Why load 8 and not 9? Because 0 is included in count sequence:
    - 8, 7, 6, 5, 4, 3, 2, 1, 0 → 9 counts



Datapath Components
Counter Example: Timer

- A type of counter used to measure time
  - If we know the counter's clock frequency and the count, we know the time that's been counted
- Example: Compute car's speed using two sensors
  - First sensor (a) clears and starts timer
  - Second sensor (b) stops timer
  - Assuming clock of 1kHz, timer output represents time to travel between sensors. Knowing the distance, we can compute speed



Datapath Components
Multipliers – Array Style

4.7

- Can build multiplier that mimics multiplication by hand
  - Notice that multiplying multiplicand by 1 is same as ANDing with 1

```
    0110  (the top number is called the multiplicand)
    0011  (the bottom number is called the multiplier)
    ----  (each row below is called a partial product)
    0110  (because the rightmost bit of the multiplier is 1, and 0110*1=0110)
   0110   (because the second bit of the multiplier is 1, and 0110*1=0110)
  0000    (because the third bit of the multiplier is 0, and 0110*0=0000)
+0000     (because the leftmost bit of the multiplier is 0, and 0110*0=0000)
--------
00010010  (the product is the sum of all the partial products: 18, which is 6*3)
```


Datapath Components
Multipliers – Array Style

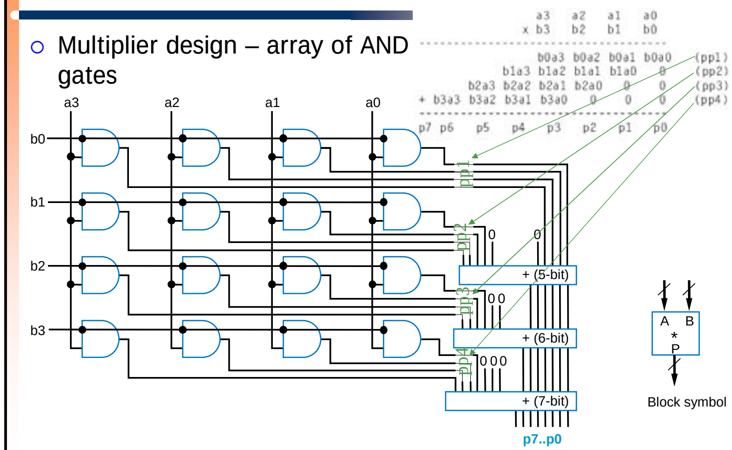
- Generalized representation of multiplication by hand

```
                          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    0    (pp4)
   ------------------------------------------
 p7  p6    p5   p4   p3   p2   p1   p0
```

Datapath Components

Multipliers – Array Style

- Multiplier design – array of AND gates



In-class Exercise

- Design a somewhat accurate Celsius to Fahrenheit converter.
  - The conversion circuit receives a digitized temperature in Celsius as a 16-bit binary number *C* and outputs the temperature in Fahrenheit as a 16-bit output *F* using the following approximation:
  - F = C*30/16 + 32.