CSE 311 Foundations of Computing I

Lecture 25
Circuits for FSMs, Carry-Look-Ahead Adders
Autumn 2011

Announcements

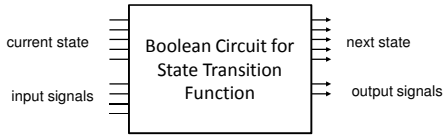
- Nice overview of adder circuits at
  http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html

Last lecture highlights

- Languages not recognized by any DFA
  - $\{ 0^n1^n : n \geq 0 \}$
  - Binary Palindromes
  - Strings of Balanced Parentheses
- Using DFAs for efficient pattern matching

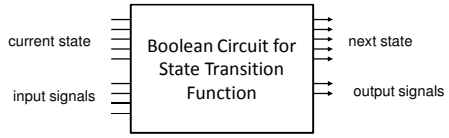
FSMs in Hardware

- Encode the states in binary: e.g. states 0,1,2,3 represented as 000,100, 010,001, or as 00,01,10,11.
- Encode the input symbols as binary signals
- Encode the outputs possible as binary signals
- Build combinational logic circuit to compute transition function:



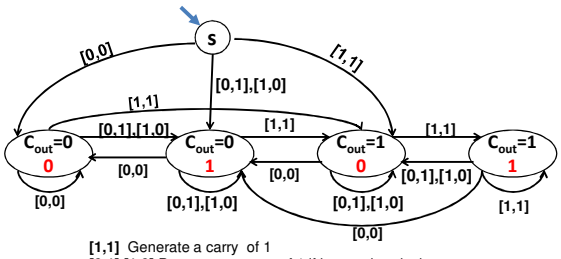
FSMs in Hardware

- Combine with sequential logic for
  - Registers to store bits of state
  - Clock pulse
- At start of clock pulse, current state bits from registers and input signals are released to the circuit
- At end of clock pulse, output bits are produced and next state bits are stored back in the same registers



FSM for binary addition

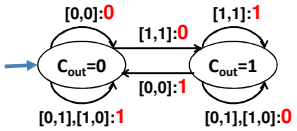
- Assume that the two integers are $\mathbf{a_{n-1}...a_2a_1a_0}$ and $\mathbf{b_{n-1}...b_2b_1b_0}$ and bits arrive together as $\mathbf{[a_0,b_0]}$ then $\mathbf{[a_1,b_1]}$ etc.



[1,1] Generate a carry of 1
[0,1],[1,0] Propagate a carry of 1 if it was already there

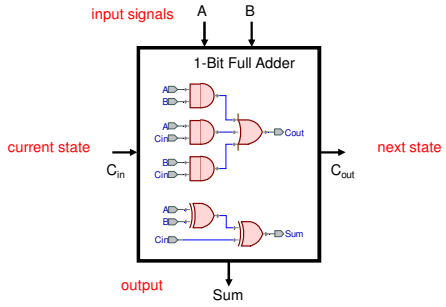
FSM for binary addition using output on edges

- Assume that the two integers are $\mathbf{a_{n-1}...a_2a_1a_0}$ and $\mathbf{b_{n-1}...b_2b_1b_0}$ and bits arrive together as $\mathbf{[a_0,b_0]}$ then $\mathbf{[a_1,b_1]}$ etc.



[1,1] Generate a carry of 1
[0,1],[1,0] Propagate a carry of 1 if it was already there

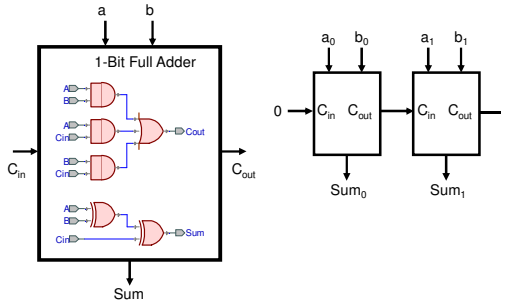
Example: 1-bit Full Adder



FSMs without sequential logic

- What if the entire input bit-strings are available at all once at the start?
  - E.g. 64-bit binary addition
- Don't want to wait for 64 clock cycles to compute the output!
- Suppose all input strings have length **n**
  - Can chain together **n** copies of the state transition circuit as one big combinational logic circuit

A 2-bit ripple-carry adder



Problem with Chaining Transition Circuits

- Resulting Boolean circuit is "deep"
- There is a small delay at each gate in a Boolean circuit
  - The clock pulse has to be long enough so that all combinational logic circuits can be evaluated during a single pulse
  - Deep circuits mean slow clock.

Speeding things up?

- To go faster, need to work on both 1st half and 2nd half of the input at once
  - How can you determine action of FSM on 2nd half without knowing state reached after reading 1st half?

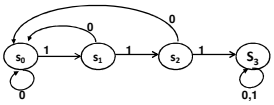
$$b_1b_2...b_{n/2}\,b_{n/2+1}\,...b_{n-1}b_n$$

what state?

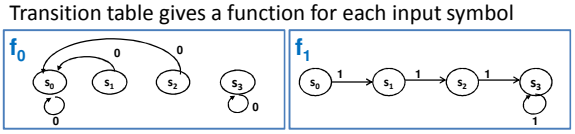
- Idea: Figure out what happens in 2nd half for *all* possible values of the middle state at once

Transition Function Composition

State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3



Transition table gives a function for each input symbol



State reached on input $\mathbf{b_1 \ldots b_n}$ is

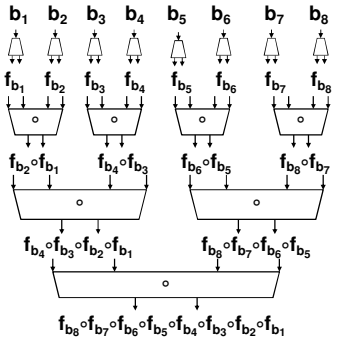
$$\mathbf{f_{b_n}}(\mathbf{f_{b_{n-1}}} \ldots (\mathbf{f_{b_2}}(\mathbf{f_{b_1}}(\text{start}))) \ldots) = \mathbf{f_{b_n}} \circ \mathbf{f_{b_{n-1}}} \ldots \circ \mathbf{f_{b_2}} \circ \mathbf{f_{b_1}}(\text{start})$$

Transition Function Composition

Constant size 2-level Boolean logic to
- convert input symbol to bits for transition function

- compute composition of two transition functions

Total depth 2 $\log_2 \mathbf{n}$ and size $\approx \mathbf{n}$



Carry-Look-Ahead Adder

Compute generate $G_i = a_i \wedge b_i$ [1,1]
propagate $P_i = a_i \oplus b_i$ [0,1],[1,0]

These determine transition and output functions
- Carry $C_i = G_i \vee (P_i \wedge C_{i-1})$ also written $C_i = G_i + P_i C_{i-1}$
- $\text{Sum}_i = P_i \oplus C_{i-1}$

Unwinding, we get

$C_0 = G_0$ $C_1 = G_1 + G_0 P_1$ $C_2 = G_2 + G_1 P_2 + G_0 P_1 P_2$
$C_3 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3$
$C_4 = G_4 + G_3 P_4 + G_2 P_3 P_4 + G_1 P_2 P_3 P_4 + G_0 P_1 P_2 P_3 P_4$
etc.

Carry-Look-Ahead Adder

Compute all generate $G_i = a_i \wedge b_i$ [1,1]
propagate $P_i = a_i \oplus b_i$ [0,1],[1,0]

Then compute all:

$C_0 = G_0$ $C_1 = G_1 + G_0 P_1$ $C_2 = G_2 + G_1 P_2 + G_0 P_1 P_2$
$C_3 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3$
$C_4 = G_4 + G_3 P_4 + G_2 P_3 P_4 + G_1 P_2 P_3 P_4 + G_0 P_1 P_2 P_3 P_4$ etc.

Finally, use these to compute

$\text{Sum}_0 = P_0$ $\text{Sum}_1 = P_1 \oplus C_0$ $\text{Sum}_2 = P_2 \oplus C_1$
$\text{Sum}_3 = P_3 \oplus C_2$ $\text{Sum}_4 = P_4 \oplus C_3$ $\text{Sum}_5 = P_5 \oplus C_4$ etc

If all C_i are computed using 2-level logic, total depth is 4.

Adders using Composition

Carry-look-ahead circuit for carry C_{n-1} has $2 + 3 + \ldots + n = (n+2)(n-1)/2$ gates
- a lot more than ripple-carry adder circuit.

Composition gives an alternative approach

Composition: (G,P) followed by (G',P') gives the same effect as (G'',P'') where
- $G'' = G' \vee (G \wedge P')$ and $P'' = P' \wedge P$ also written as $G'' = G' + G\,P'$ and $P'' = P'P$

Adders using Composition

Composition: (G,P) followed by (G',P') gives the same effect as (G'',P'') where
- $G'' = G' \vee (G \wedge P')$ and $P'' = P' \wedge P$ also written as $G'' = G' + G\,P'$ and $P'' = P'P$

Use this for the circuit component

in transition function composition tree
- Computes C_{n-1} in depth $2 \log_2 n$ and size $\approx n$

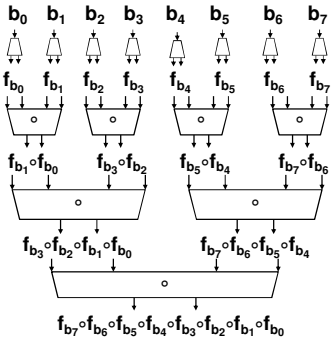
- But we need all of $C_0, C_1, \ldots, C_{n-1}$ not just C_{n-1}

Transition Function Composition

Constant size 2-level Boolean logic to
- convert input symbol to bits for transition function

- compute composition of two transition functions

Total depth 2 $\log_2$ **n** and size ≈**n**



Computing all the values

- We need to compute all of

$f_{b_7}\circ f_{b_6}\circ f_{b_5}\circ f_{b_4}\circ f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0}$ Already computed

$f_{b_6}\circ f_{b_5}\circ f_{b_4}\circ f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0}$ $= f_{b_6}\circ(f_{b_5}\circ f_{b_4})\circ(f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0})$

$f_{b_5}\circ f_{b_4}\circ f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0}$ $= (f_{b_5}\circ f_{b_4})\circ(f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0})$

$f_{b_4}\circ f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0}$ $= f_{b_4}\circ(f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0})$

$f_{b_3}\circ f_{b_2}\circ f_{b_1}\circ f_{b_0}$ Already computed

$f_{b_2}\circ f_{b_1}\circ f_{b_0}$ $= f_{b_2}\circ(f_{b_1}\circ f_{b_0})$

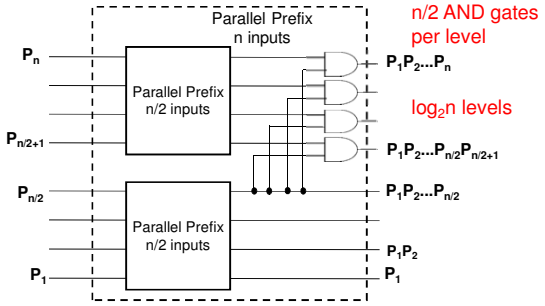
$f_{b_1}\circ f_{b_0}$ Already computed

f_{b_0} Already computed

Parallel Prefix Circuit

- The general way of doing this efficiently is called a parallel prefix circuit
  - Designed and analyzed by Michael Fischer and Richard Ladner (University of Washington)
- Uses the adder composition operation that sets G''= G'+G P' and P''= P'P
  - we just show it for the part for computing P'' which is a Parallel Prefix AND Circuit

The Parallel Prefix AND Circuit



Parallel Prefix Adder

- Circuit depth 2 $\log_2$ n
- Circuit size 4 n $\log_2$ n
- Actual adder circuits in hardware use combinations of these ideas and more but this gives the basics
- Nice overview of adder circuits at
  http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html