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:

FSM for binary addition

• Assume that the two integers are \( a_{n-1} \ldots a_1 a_0 \) and \( b_{n-1} \ldots b_1 b_0 \) and bits arrive together as \([a_0, b_0]\) then \([a_1, b_1]\) etc.

\[ C_{out} = \begin{cases} 1 & [0,1], [1,0] \\ 0 & [0,0], [0,1], [1,1] \\ 1 & [1,0] \end{cases} \]

[Diagram of FSM for binary addition with state transitions and carry-out conditions]
FSM for binary addition using output on edges

- Assume that the two integers are $a_{n-1}a_{n-2}...a_2a_1a_0$ and $b_{n-1}...b_1b_0$ and bits arrive together as $[a_0, b_0]$ then $[a_1, b_1]$ etc.

$$\begin{align*}
&[0,0]: 0 \\
&[1,0]: 1 \\
&[0,1]: 1 \\
&[1,1]: 0
\end{align*}$$

- [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?
  - Suppose $b_1b_2...b_{n/2}b_{n/2+1}...b_{n-1}b_n$ is the input
    - What state?
  - Idea: Figure out what happens in 2nd half for all possible values of the middle state at once
Transition Function Composition

Transition table gives a function for each input symbol

State reached on input $b_2\ldots b_n$ is

$T_0: f_0|t_0|s_0^1 \ldots (f_0|t_0|s_0^{n-1}) = f_{b_0} \circ f_{b_1} \circ \ldots \circ f_{b_2}$

Unwinding, we get

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 $n$ and size $n$

Carry-Look-Ahead Adder

Compute generate $G = a_i \land b_i$ [1,1]
propagate $P = a_i \lor b_i$ [0,1],[1,0]
These determine transition and output functions
- Carry $C_i = G \lor (P \land C_{i-1})$ also written $C_i = G \lor P \lor C_{i-1}$
- Sum $S_i = P \oplus C_i$

Unwinding, we get

Carry-Look-Ahead Adder

Compute all generate $G = a_i \land b_i$ [1,1]
propagate $P = a_i \lor b_i$ [0,1],[1,0]
Then compute all:

Carry-Look-Ahead Adder

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.

Adders using Composition

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

Use this for the circuit component in transition function composition tree
- Computes $C_i$ in depth 2 log $n$ and size $n$
  - But we need all of $C_0, C_1, \ldots, C_{n-1}$ not just $C_{n-1}$
Transition Function Composition

- 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
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)
- \( f_{b_0} \left( f_{b_0} \circ f_{b_0} \right) \)

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 \( G'' = G' + G \) and \( P'' = P' \)
- We just show it for the part for computing \( P'' \) which is a Parallel Prefix AND Circuit

The Parallel Prefix AND Circuit

- \( n/2 \) AND gates per level
- \( \log_2 n \) levels

Parallel Prefix Adder

- Circuit depth \( 2 \log_2 n \)
- Circuit size \( 4 \log_2 n \)
- 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.ece.tohoku.ac.jp/arith/mg/algorith.html