Still More Arithmetic Circuits

CS255

Chris Pollett

Mar. 1, 2006.

Outline

- Circuits for Multiplication
- Clocked Circuits

Grade School Multiplication

- Want a circuit which takes as input two n-bit numbers a= <a_{n-1}, .. a_0> and b=<b_{n-1}, ...b_0> and which outputs their at most 2n-bit product, p = <p_{2n-1}, .. p_0>.
- The "grade school" algorithm first computes the **partial products** $m^{(i)} = a*b_i*2^i$ for each i from 0 to n-1. Then we compute $p = a \cdot b = \sum_{i=0}^{n-1} m^{(i)}$

Array Multipliers

- To make circuits for multiplication we will use an array multiplier. This is a circuit which does three things:
  1. Forms the partial products
  2. Sums the partial products using carry save adders
  3. Sums the two remaining numbers using either a ripple carry or carry lookahead addition.
- To form the partial $m^{(i)}$ each input bit a_j of a is AND'd with b_i to produce output bit $m^{(i)}_{j+i}$ of the partial product. The remaining bits of the partial product are 0.
- Then we carry save add 0, $m^{(0)}$, $m^{(1)}$ to produce $u^{(1)}$ and $v^{(1)}$. To these two outputs we add $m^{(2)}$ to get two outputs $u^{(2)}$ and $v^{(2)}$. We continue in this until we get two 2n-1 bit numbers $u^{(n-1)}$ and $v^{(n-1)}$.
- Finally, we ripple carry or carry lookahead add these two numbers.

Analysis

- Each of these carry save additions only needs to operate on n-1 bits.
- It takes $\Theta(1)$ depth/time to do one carry save addition. We are doing the additions of $m^{(i)}$'s in series and there are n of them. So the total time for this part $\Theta(n)$.
- Then if we do ripple carry addition we need a $\Theta(n)$ additional depth/time. If use carry lookahead addition we need $\Theta(\log n)$ additional depth/time.
- So the total time is $\Theta(n)$.
- The circuit uses n^2 AND gates and n^2-n full adders. So the total size is $\Theta(n^2)$.
- We'd now like to make circuits that can do the above in $\Theta(\log n)$ time.

Wallace-tree Multipliers

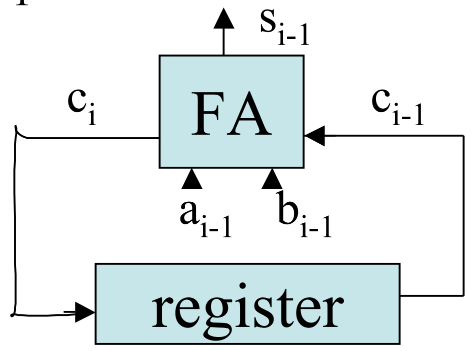
- A Wallace-tree allows us to parallelize the carry-save adder part of the above circuit.
- It uses $\lfloor n/3 \rfloor$ carry-save adders in parallel to convert the sum of n numbers to the sum of $\lceil 2n/3 \rceil$ numbers. To do this it carry-save adds $m^{(i)}, m^{(i+1)}, m^{(i+2)}$ for each $i = 0, \ldots \lfloor n/3 \rfloor - 1$.
- It then recursively constructs a Wallace tree on the resulting $\lceil 2n/3 \rceil$ numbers. This is recursed until only two numbers are left.
- So the depth satisfies the recurrence $D(n) = 0$ if $n<=2$; $D(n) = 1$ if $n =3$ and $D(n) = D(\lceil 2n/3 \rceil) +1$ if $n>=4$. So $D(n) = \Theta(\log n)$.
- So if we use Wallace-tree multipliers and do the last addition using carry-lookahead addition, the total depth of the circuit for multiplication will be $\Theta(\log n)$. Its size will again be $\Theta(n^2)$.

Clocked Circuits

- In our circuit so far each element in the circuit is only used once during the computation.
- We want to be able to reuse components more than once in practice in real hardware.
- Allowing for this will tend to make our circuits much smaller.
- To do this we will introduce **clocked circuits** (aka **sequential circuits**) and consider circuits for them.
- A clocked circuit consists of combinational circuitry together with one or more **registers** (clocked memory elements).
- Each register in the circuit is controlled by a periodic signal (aka clock). When the clock **ticks**, the register loads in and stores the value at its inputs.
- In a **globally clocked** circuit, every register works off the same clock.
- We assume the delay of the combinational elements of our circuit is less then the length of a tick.

Bit-serial addition

- Now suppose we want to do addition of two n-bit numbers a, b on a clocked circuit.
- We can use a single full adder in the arrangement below.
- At time 0, we load the register with 0 and feed a_0, b_0 into the inputs of the register.
- At time i, we load the register with the carry output of time i-1, we store the sum s_{i-1} to some external memory, and we supply a_{i-1}, b_{i-1} to the full adder's inputs.



- The size is thus $\Theta(1)$.
- The time will be $\Theta(n)$

Ripple-carry versus Bit-Serial Addition

- Notice a ripple-carry adder is like a replicated bit-serial adder with registers replaced by direct connections between combinational elements.
- i.e., it is like an “unrolled” version of a bit-serial adder.
- In general, we can replace any clocked circuit by an equivalent combinational circuit with the same asymptotic time delay by doing this kind of unrolling.
- The clocked circuit has the advantage of fewer elements, but the combinational has less control circuitry and will typically run faster because it doesn’t have to wait for clock periods to end to propagate values.

Linear Array Multipliers

- Our circuits for multiply above were of $\Theta(n^2)$ size.
- We now investigate clocked multipliers which are of linear size.
- The first such multiplier implements the Russian peasant's algorithm for multiplication.
- In this algorithm we make two columns one for the input *a*, one for the input *b*.
- We then make rows where in a row, we shift right *a* by 1 (dropping the low order bit) and we shift left *b* by 1 (multiplying by 2). We keep making rows until the *a* column contains just 1 (the high order bit of *a*).
- Then we add the *b* column for rows where the *a* column was odd.

Russian Peasant Example

a	b
19	29
9	58
4	116
2	232
1	464
	551

a	b
10011	11101
1001	111010
100	1110100
10	11101000
1	111010000
	1000100111

So very much like doing multiplication in binary except we have compressed the format of the table.

Implement the Russian Peasant Algorithm

- We use a clocked circuit consisting of a linear array of 2n cells.
- Each cell contains three registers.
- One register holds a bit from an *a* entry, one holds a bit from a *b* entry and one holds a bit of the product *p*.
- We use superscripts to denote cell values at just before a time.

  i.e., $a^{(j)} = <a_{2n-1}^{(j)}, \dots a_0^{(j)}>$ for *a* just before time *j*.
- The invariant we will use is that $a^{(j)}*b^{(j)} + p^{(j)} = a*b$.
- Initially, $a^{(0)} = a$, $b^{(0)} = b$, $p^{(0)} = 0$.
- The *j*th step of of the computation then does:
  - If $a^{(j)}$ is odd, then add b into p: $p^{(j+1)} = b^{(j)} + p^{(j)}$ using ripple carry. (Adding $b^{(j)}$ is all that needs to be done here as we'll be shifting *b* below.) Other wise, if $a^{(j)}$ is even, $p^{(j+1)} = p^{(j)}$.
  - Shift *a* right by one bit position
  - Shift *b* left by one bit position.
- After n steps we output the result. So the size is $\Theta(n)$ and the runtime is $\Theta(n^2)$ as the addition in each time takes linear time.

Fast Linear Array Multipliers

- We want to reduce the time from $\Theta(n^2)$ to $\Theta(n)$.
- To do this we modifier the above algorithm to use carry save addition.
- Our invariant is now $a^{(j)}*b^{(j)} + u^{(j)} + v^{(j)} = a*b$.
- Here u and v are the outputs of a full-adder.
- That is, if $a^{(j)}$ is odd then:

  $u^{(j+1)}_i = \text{parity}(b^{(j)}_{i,}\ u^{(j)}_i, v^{(j)}_i)$

  $v^{(j+1)}_i = \text{majority}(b^{(j)}_{i-1,}\ u^{(j)}_{i-1}, v^{(j)}_{i-1})$ if $i>0$

  0 otherwise

  and if it is even then:

  $u^{(j+1)}_i = \text{parity}(0_,\ u^{(j)}_i, v^{(j)}_i)$

  $v^{(j+1)}_i = \text{majority}(0,_,\ u^{(j)}_{i-1}, v^{(j)}_{i-1})$ if $i>0$

  0 otherwise
- It turns out that at the last step $v^{(2n-1)}=0$ so $u^{(2n-1)}$ will have the product.
- The total time is thus $\Theta(n)$.