EECS150 - Digital Design
Lecture 23 - Arithmetic and Logic Circuits Part 4

April 19, 2005

John Wawrzynek

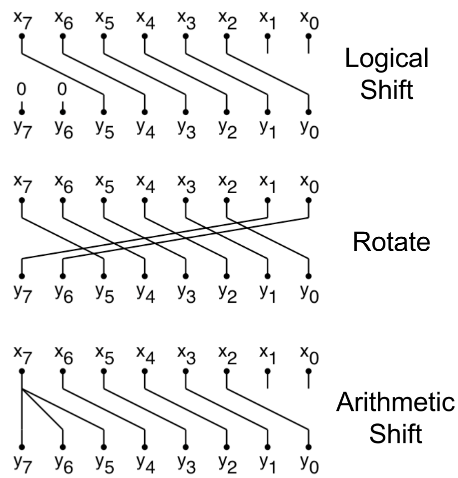
Outline

- Shifters / Rotators
  - Fixed shift amount
  - Variable shift amount
- Multiplication Revisited
  - Fixed multiplication value (multiplication by a constant)
  - Variable multiplication value (done last week)

Fixed Shifters / Rotators

- "fixed" shifters "hardwire" the shift amount into the circuit.
- Ex: X >> 2
  - (right shift X by 2 places)
- Fixed shift/rotator is nothing but wires!

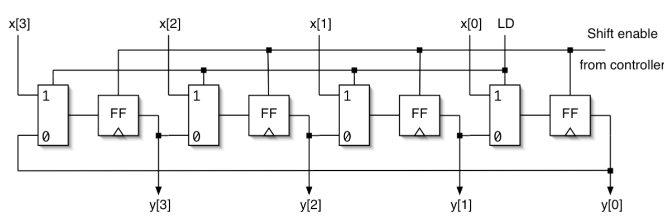
  So what?



Variable Shifters / Rotators

- Example: X >> S, where S is unknown when we design and build the circuit.
- Uses: shift instruction in processors (ARM includes a shift on every instruction), floating-point arithmetic, division/multiplication by powers of 2, etc.
- One way to build this is a simple shift-register:

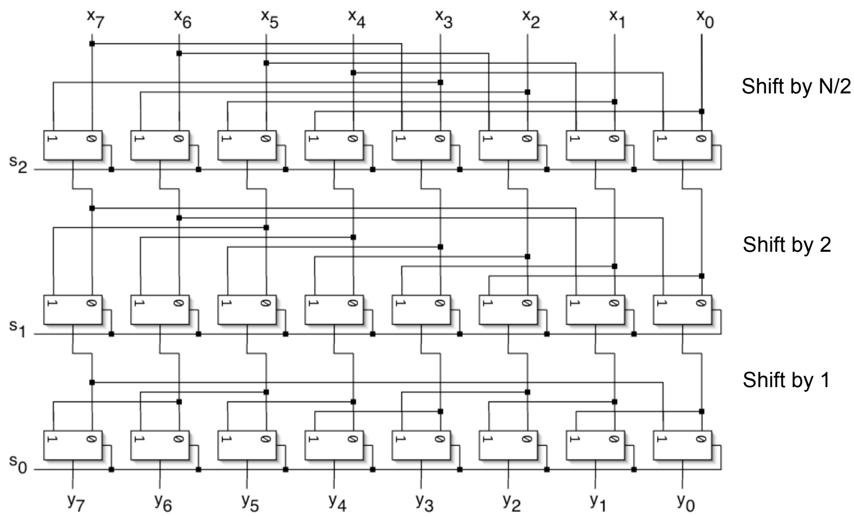
  a) Load word, b) shift enable for S cycles, c) read word.



- Worst case delay O(N) , not good for processor design.
- Can we do it in O(logN) time and fit it in one cycle?

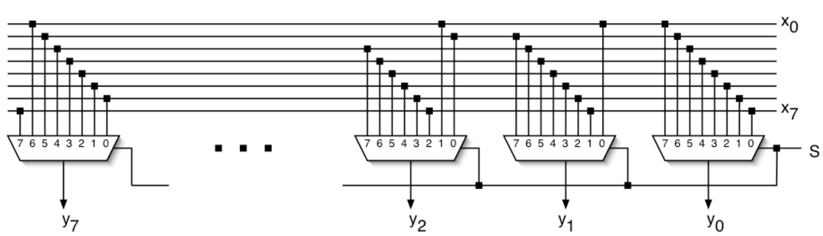
Funnel Shifter / Rotator

- Log(N) stages, each shifts (or not) by a power of 2 places, $S=[s_2;s_1;s_0]$:



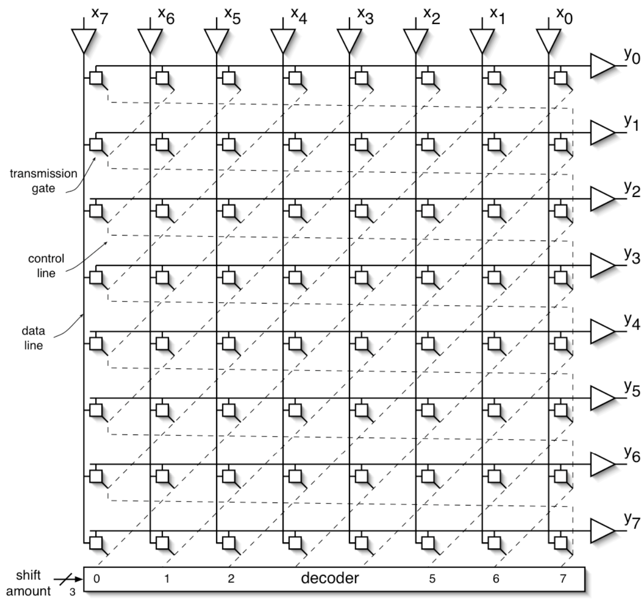
"Improved" Shifter / Rotator

- How about this approach? Could it lead to even less delay?



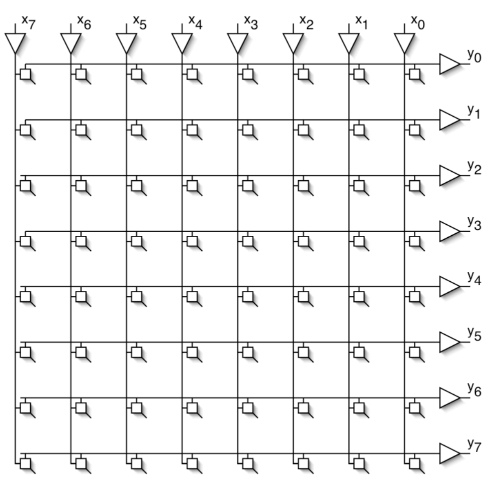
- What is the delay of these big muxes?
- How about a transistor-level optimization.

Barrel Shifter



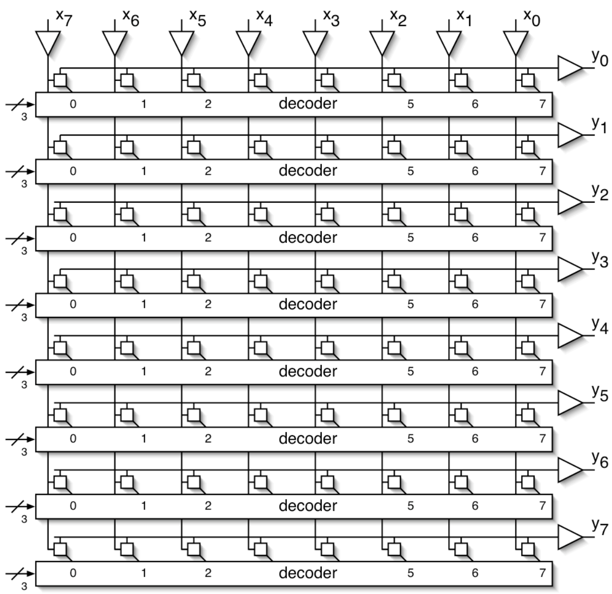
Cost/delay?
- (don't forget the decoder)

Connection Matrix



Generally useful structure:
- N^2 control points.
- What other interesting functions can it do?

Cross-bar Switch



- Nlog(N) control signals.
- Supports all interesting permutations
  - All one-to-one and one-to-many connections.
- Commonly used in communication hardware (switches, routers).

Multiplication Revisited

$$
\begin{array}{ccccccccl}
 & & & a_3 & a_2 & a_1 & a_0 & \leftarrow & \textit{Multiplicand} \\
 & & & b_3 & b_2 & b_1 & b_0 & \leftarrow & \textit{Multiplier} \\
\hline
 & & X & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & & \\
 & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & & \textit{Partial} \\
 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & & \textit{products} \\
a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & & & \\
\hline
 & & & \ldots & & a_1b_0+a_0b_1 & a_0b_0 & \leftarrow & \textit{Product}
\end{array}
$$

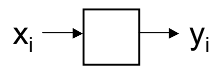
Multiplication Revisited

- *Our discussion so far has assumed both the multiplicand (A) and the multiplier (B) can vary at runtime.*
- What if one of the two is a constant?

$$Y = C * X$$

- "Constant Coefficient" multiplication comes up often in signal processing and other hardware. Ex:

$$y_i = \alpha y_{i-1} + x_i$$

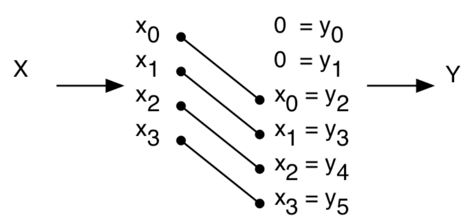


where α is an application dependent constant that is hard-wired into the circuit.

- How do we build and array style (combinational) multiplier that takes advantage of the constancy of one of the operands?

Multiplication by a Constant

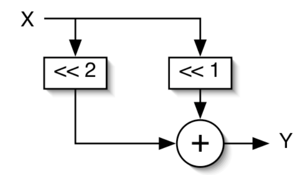
- If the constant C in C*X is a power of 2, then the multiplication is simply a shift of X.
- Ex: 4*X



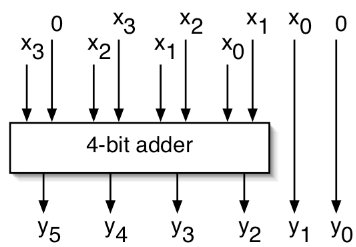
- What about division?
- What about multiplication by non- powers of 2?

Multiplication by a Constant

- In general, a combination of fixed shifts and addition:
  - Ex: $6*X = 0110 * X = (2^2 + 2^1)*X$

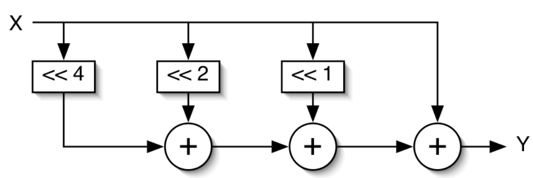


  - Details:



Multiplication by a Constant

- Another example: $C = 23_{10} = 010111$



- *In general, the number of additions equals one minus the number of 1's in the constant, C.*
- Using carry-save adders (for all but one of these) helps reduce the delay and cost, but the number of adders is still the number of 1's in C minus 1.

- Is there a way to further reduce the number of adders (and thus the cost and delay)?

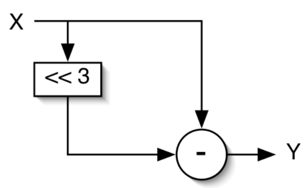
Multiplication using Subtraction

- *Subtraction is the same cost and delay as addition.*
- Consider C*X where C is the constant value 15_{10} = 01111.
  - C*X requires 3 adders (probably 2 CSA and 1 CPA).
- We can "recode" 15

  from 01111 = $(2^3 + 2^2 + 2^1 + 2^0)$

  to $1000\bar{1} = (2^4 - 2^0)$

  where $\bar{1}$ means negative weight.
- Therefore, 15*X can be implemented with only one subtractor.



Canonic Signed Digit Representation

- CSD represents numbers using 1, $\bar{1}$, & 0 with the least possible number of non-zero digits.
  - Strings of 2 or more non-zero digits are replaced.
  - Leads to a unique representation.
- To form CSD representation might take 2 passes:
  - First pass: replace all occurrences of 2 or more 1's:

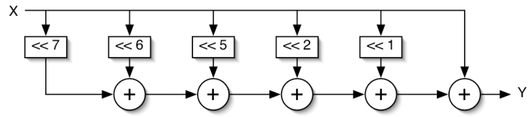
    01..10 by $10..\bar{1}0$
  - Second pass: same as a above, plus replace $0\bar{1}10$ by $00\bar{1}0$
- Examples:

011101 = 29	0010111 = 23	0110110 = 54
$100\bar{1}01$ = 32 - 4 + 1	$001100\bar{1}$	$10\bar{1}10\bar{1}0$
	$010\bar{1}00\bar{1}$ = 32 - 8 - 1	$100\bar{1}0\bar{1}0$ = 64 - 8 - 2

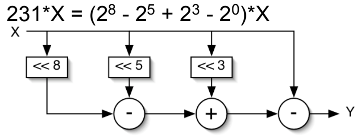
- Can we further simplify the multiplier circuits?

"Constant Coefficient Multiplication" (KCM)

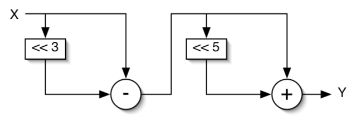
Binary multiplier: $Y = 231*X = (2^7 + 2^6 + 2^5 + 2^2 + 2^1+2^0)*X$



- CSD helps, but the multipliers are limited to shifts followed by adds.
  - CSD multiplier: $Y = 231*X = (2^8 - 2^5 + 2^3 - 2^0)*X$



- How about shift/add/shift/add …?
  - KCM multiplier: $Y = 231*X = 7*33*X = (2^3 - 2^0)*(2^5 + 2^0)*X$



- No simple algorithm exists to determine the optimal KCM representation.
- Most use exhaustive search method.