Adders (cont.)

Bit-serial Addition:

\[ A + B \]

\[ \text{CLR} \rightarrow \text{FF} \rightarrow \text{FA} \rightarrow R \]

A, B, R are held in "shift-registers", they shift right once per clock cycle.
CLR is asserted by controller.

Addition of 2 n-bit numbers:
- take n clock cycles
- uses 1 FA cell, 1 FF. The bit streams may come from other circuits & go to

therefore may not be present.
Multiplication

\[
\begin{array}{cccccc}
 a_3 & a_2 & a_1 & a_0 & \leftarrow \text{"Multiplier"} \\
 \times & b_3 & b_2 & b_1 & b_0 & \leftarrow \text{"Multiplier"} \\
 \hline
 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 \\
 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 \\
 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 \\
 \hline
 & \ldots & a_0b_0 + a_0b_1 + a_0b_2 \\
\end{array}
\]

Many different circuits exist for multiplication. Each one has a different balance between speed (performance) \& amount of logic (cost).
"Shift & Add" Multiplier (unsigned)

- Adds in each partial product, one at a time
- In binary - each partial product is either shifted versions of A or φ.

Control algorithm:

1. P ← φ, A ← multiplicand, B ← multiplier
2. If LSB of B = 1 then add A to P
   else add φ
3. Shift [P][B] right 1
4. Repeat 2-3 n-1 times
5. [P][B] has product
Shift & Add approach:

cost \propto n

\tau = n \text{ clock cycles}

What is the critical path for determining min. \tau?

---

Signed Multiplication:

Remember for 2's complement numbers

MSB has negative weight

\[ x = \sum_{i=0}^{N-2} x_i 2^i - x_{n-1} 2^{n-1} \]

Ex: \(-6 = 1101_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 - 1 \cdot 2^4\]

\[ = 0 + 2 + 0 + 8 - 16 \]

\[ = -6 \]

Therefore for multiplication:

a) Subtract final partial product

b) Sign-extend partial products

Modifications to shift & add circuit:

a) Adder/Subtractor

b) Sign-extends on P shift register
Array Multiplier

Generates all \( n \) \( n \)-bit partial products simultaneously.

Each row's \( n \)-bit adder with AND gates.

What is the critical path?
Speeding up multiplication using "carry-save addition" is a matter of speeding up the addition of sets of numbers (partial products).

**Carry-save addition** passes (saves) carries to output rather than propagating them.

Example: sum 3 numbers

\[ 3_{10} = 0011, \quad 2_{10} = 0010, \quad 3_{10} = 0011 \]

\[
\begin{array}{c}
3_{10} \\
+ 2_{10} \\
\hline
C \quad 0100 = 4_{10} \\
S \quad 0001 = 1_{10}
\end{array}
\]

\[
\begin{array}{c}
3_{10} \\
+ 3_{10} \\
\hline
C \quad 0010 = 2_{10} \\
S \quad 0110 = 6_{10}
\end{array}
\]

Avoid carry until the end.
Array Multiplier using carry-save addition:
Carry Save Circuits

For adding sets of numbers carry-save can be used on all but the final sum:

standard adder (carry propagate)
Carry Save Addition
on sets of numbers is associative + commutative

ex.

\[((X_0 + X_1) + X_2) + X_3\] = 
\[((X_0 + X_1) + (X_2 + X_3))\]
Wallace Tree Multiplier

Partial products

Fast adder

delay $\propto \log_2 n$

delay $\propto \log_2 n$