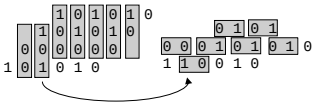
FPGA Arithmetic II

- More on multipliers
  - Faster 2-operand
  - Constant Multipliers
- Floating Point

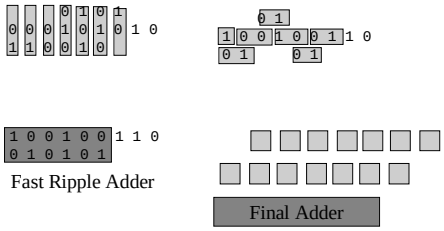
The Wallace Tree

```
      1 0 1 0 1 0  Multiplicand (N bits)
    x   1 0 1 1  Multiplier   (M bits)
      1 0 1 0 1 0
    1 0 1 0 1 0    Partial Products
  0 0 0 0 0 0
1 0 1 0 1 0
1 1 1 0 0 1 1 1 0  Product
```

Full adder: 3 inputs, 2 output (3-2 Counter)



Wallace Tree



Good: Depth is $\text{Log}_{3/2}(MN)$
Bad: Routing is Irregular

Partial Product Generation

- Simple: AND gates
- Complex: Little Multipliers

Ferrari-Stefanelli

```
      1 0 1 0 Multiplicand (N bits)
    x   1 0 1 1  Multiplier   (M bits)
      0 1 1 0 (10 X 11)
    0 1 1 0     (10 x 11)
    0 1 0 0     (10 x 10)
  0 1 0 0       (10 x 10)
  0 1 1 0 1 1 1 0 Product
```

4 bit multiplier = 2 CLBs

CSD Vectors

- CSD: Canonical Signed Digit representation

  Use subtraction to compress representation

$$D_i = B_i + C_i - 2C_{i+1}$$

C_{i+1} is carryout of $B_{i+1} + B_i + C_i$

```
  B = 00111101
  C = 01111000
-2C =   222200
  D = 01000101
```

- Subtractors are easy to build

Booth Encoding

- "Windowed" CSD encoding

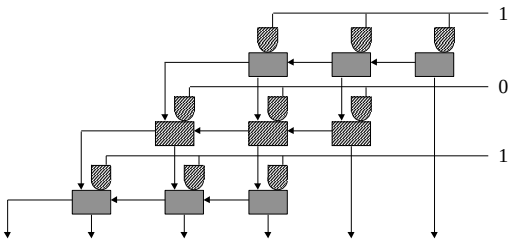
$$E_j = -2B_i + B_{i-1} + B_{i-2}$$

```
B = 00111101
B =0000111101 0 (Padded)
E = 0 1 0 1 1
```

- Reduces partial products by 1/2
- Speeds bit serial multiplication
- Automatically provides 2's complement multiply

Constant Multipliers

- Propagate constants thru array multiplier



Constant Multipliers

- Logic is function of # 1s in Multiplier
- CSD allows reduction by 1/2
- Common Sub-expressions:

X * 0**1011**000**1011**0

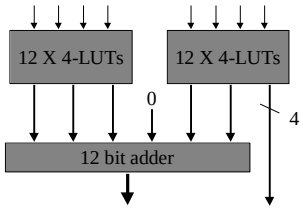
Add once (2 adds) then add together (1 add)

LUT-based Constant Multipliers

- N-LUT = constant multiply of N-bit number
- Break operand into N-bit quantities:
  - Example: 8-bit * 8-bit constant, 4-LUTs

```
                1 0 1 0 1 0 1 1
              x K K K K K K K K
        A A A A A A A A A A A A (K * 1011 (LSN))
B B B B B B B B B B B B_______ (K * 1010 (MSN))
S S S S S S S S S S S S S S S S Product
```

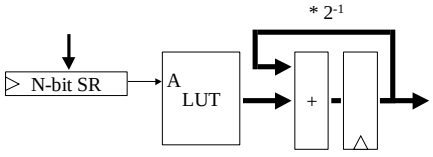
LUT-based Constant Multipliers



- Extensible
- Reprogrammable Constants

Distributed Arithmetic

- LUT-based Bit-serial Multiplication

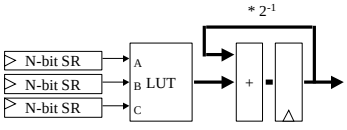


A	LUT Contents:
0	00000000
1	C_0

Distributed Arithmetic

- Weighted Sums:

$$X = C_0 * A + C_1 * B + C_2 * C$$



CBA	LUT Contents:
000	00000000
001	C_0
010	C_1
011	$C_0 + C_1$
100	C_2
101	$C_2 + C_0$
110	$C_2 + C_1$
111	$C_2 + C_0 + C_1$

Why Floating Point?

- Easier Development
  - Don't have to worry about scaling/overflow
  - Works outside of intended parameters
- Dynamic Range
  - Needed for some computations
  - Geometric Computation
  - Matrix Inversion

Shirazi Paper

- Barrel Shift VHDL?
  - What is the transformation?
  - Why not write this as a loop?
- Multiplication
  - What is "Method 3 & 4"
- Division
  - Off-chip RAM for reciprocal

Floating Point Multiplication

- Cost
  - Multiplier: 35% of 44%
  - Becomes more dominant w/ large mantissas
- By Constant?
  - Any Savings?

Floating Point Addition

- Cost:
  - Barrel Shift: 5% of 28%
  - Normalization: 7% of 28%
  - Scaling?
    - Mantissa size * Log(exponent size)

FP on FPGAs

- What features to include?