CprE / ComS 583 Reconfigurable Computing

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

Lecture #5 – FPGA Arithmetic

Quick Points

- HW #1 due Thursday at 12:00pm
  - Any comments?

Recap

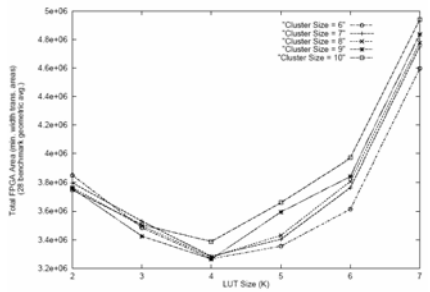
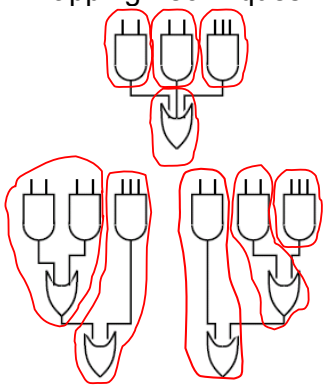


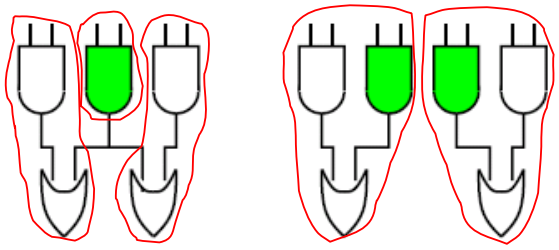
Fig. 8. Total Area for Clusters of Size 6 to 10

- Cluster size of N = [6-8] is good, K = [4-5]

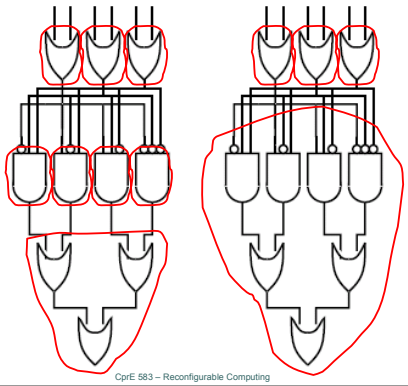
LUT Mapping Techniques



LUT Mapping Techniques (cont.)



LUT Mapping Techniques (cont.)

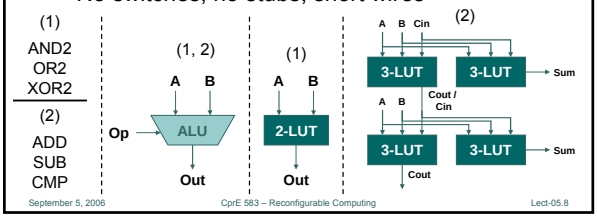


Outline

- Recap
- Motivation
- Carry / Cascade Logic
- Addition
  - Ripple Carry
  - Carry Bypass
  - Carry Select
  - Carry Lookahead
- Basic Multipliers

Motivation

- Traditional microprocessors, DSPs, etc. don't use LUTs
- Instead use a *w*-bit Arithmetic and Logic Unit (ALU)
  - Carry connections are hard-wired
  - No switches, no stubs, short wires



Adder Delays

- Assuming a ripple-carry adder:
  - 32-bit ALU delay – 6ns
  - 32-cascaded 4-LUTs – 32 x 2.5ns = 80ns

4-LUT delay

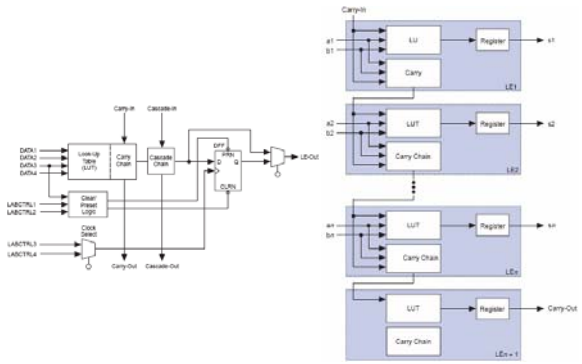
2.0 ns	Logic delay
0.5 ns	Single channel delay
2.5 ns	per bit

Compare: 32-bit ALU (0.6λ)

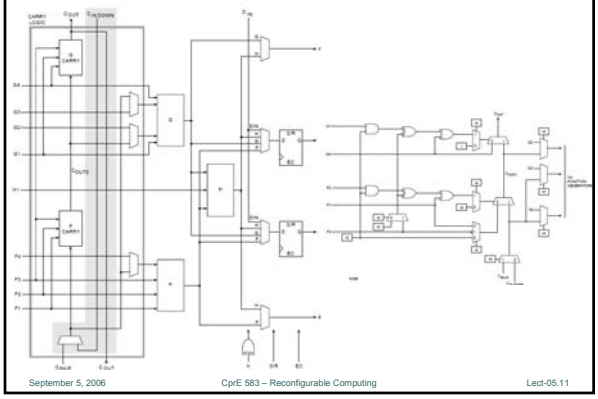
16 ns	Area optimized
6 ns	Delay optimized

- Motivates extra hardware to accelerate carry operations

Altera Flex 8000 Carry Chain



Xilinx XC4000 Carry Chain

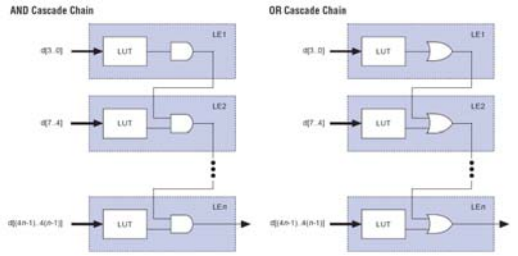


Cascades

- Large fanin operations (reductions):
  - Decoding
  - Matching
  - Completion detection
  - Many-to-one reductions
- Combining logic is simple
- Makes use of dedicated paths

Altera Cascade Logic

- LE delay – 2.4 ns
- Cascade delay – 0.6 ns

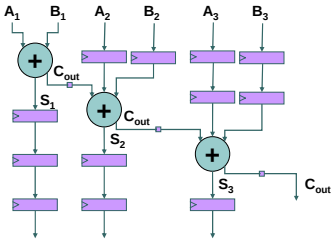


Why Look at Arithmetic?

- Parallelization
- Specialization
  - Architecture
  - Size
  - Inputs
- Adder problem – delay grows linearly with bit width
- Solutions for larger adders:
  - Pipelining
  - Carry bypass
  - Carry select

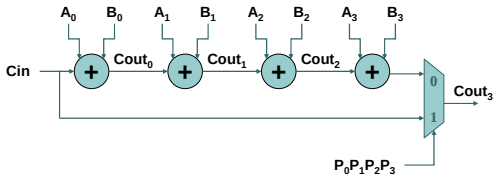
Adder Pipelining

- Not as practical in ASIC world (registers are expensive)
- Registers essentially "free" in FPGA logic blocks



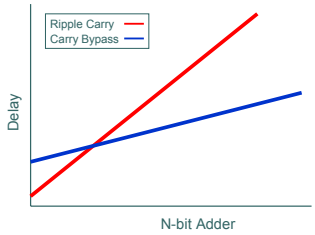
Carry Bypass Adders

- If all the propagates are 1 ($P_0P_1P_2P_3 = 1$) then $Cout_3$ = Cin
  - $P_i = A_i$ xor B_i
  - Skip all the carry logic
  - Inexpensive



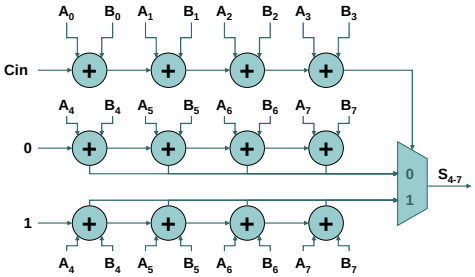
Carry Bypass Performance

- Small hardware cost:
  - 16-bit add – 4 CLB overhead
  - 32-bit add – 9 CLB overhead
- Delay growth still linear, smaller slope



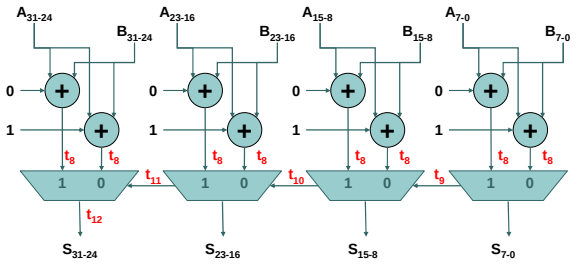
Carry Select Adders

- Precompute addition value for (Cin = 0) case and (Cin = 1) case
- Use mux to select between two with actual Cin value
- Cost of this approach?



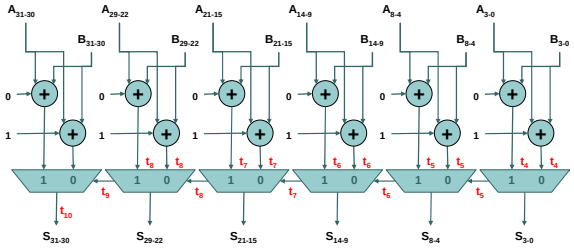
Linear Carry-Select

- Adder delay = *w*, mux delay = 1



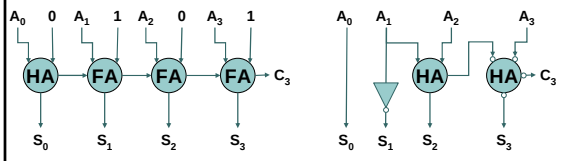
Square-Root Carry Select

- Each carry arrives when the corresponding sum is ready



Constant Addition

- If one operand is constant:
  - More speed?
  - Less hardware?



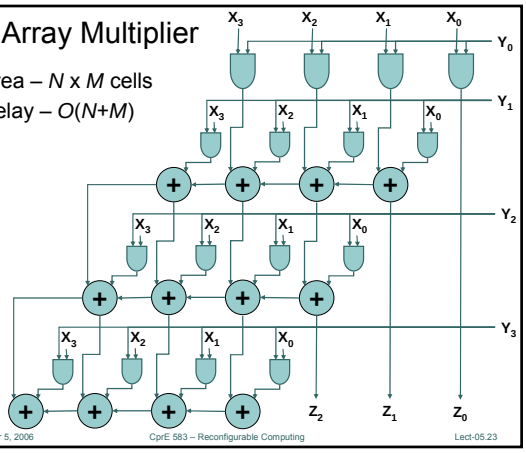
Multiplication

- Shift and add operations
- Need *N* bit adder, *M* cycles

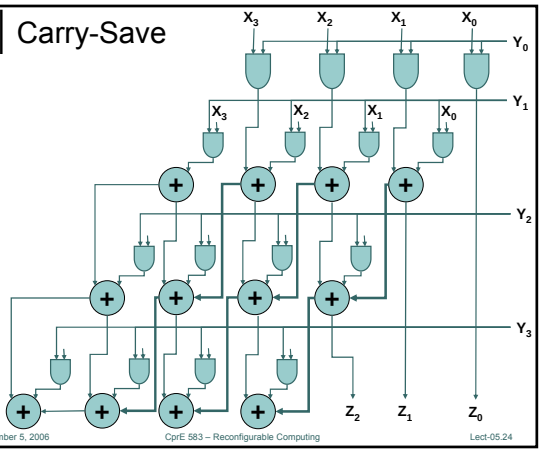
```
  42 =    101010  Multiplicand (N bits)
x 10 =    x 1010  Multiplier   (M bits)
         000000
        101010   Partial products
       000000
    + 101010
 420 = 111001110  Product     (N+M bits)
```


Array Multiplier

- Area – *N* x *M* cells
- Delay – *O*(*N*+*M*)



Carry-Save

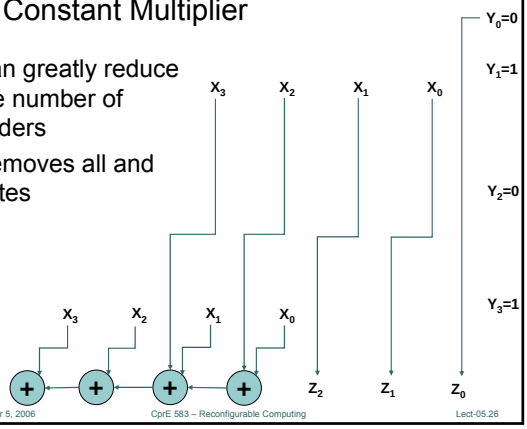


Multiplier Pipelining

- Register cost:
  - Multiplicand – (*N* bits/stage x *M* stages)
  - Multiplier – (M^2 + *M*) / 2 bits
  - Early output values – (M^2 + *M*) / 2 bits
  - Total – *M* x (*N* + *M* + 1) bits
- Critical path = max:
  - DFF + FA + setup
  - Bottom-level adder

Constant Multiplier

- Can greatly reduce the number of adders
- Removes all and gates



LUT-based Constant Multipliers

- *k*-LUT can perform constant multiply of *k*-bit number
- Break operand into *k*-bit quantities
- Example: 8-bit x 8-bit constant, *k*=4

```
        10101011
      x NNNNNNNN
  ----------------
      AAAAAAAAAA  (N * 1011 (LSN))
+ BBBBBBBBBBBBBB  (N * 1010 (MSN))
  ----------------
  SSSSSSSSSSSSSS   Product
```


Summary

- Latency overhead of programmable logic
- Several approaches to reducing design latency:
  - Fast carry
  - Cascade
  - Hardwired connections
- Multiplier optimization goals different from adder
- Other techniques:
  - Logarithmic v. linear (Wallace Tree multiplier)
  - Data encoding (Booth's multiplier)