REC FPGA Seminar IAP 1998

Session 3:

Advanced Design Techniques, Optimizations, and Tricks

Outline

- Focus on Xilinx 4000E-style FPGA (one of the most common FPGAs)
- Thinking FPGA
- Black box optimizations
- Counter design
- Distributed arithmetic
- One-hot state machines
- Miscellaneous tricks

Thinking FPGA

- When starting a design, consider the implementation technology
- Architect your design to fit into an FPGA
  - memory granularity (16x1, 16x2, 32x1)
  - 4 or 5 input logic functions / 4 + 4 and 2-1 mux
    - fewer inputs per logic function is wasteful
    - more inputs is slower
  - routing limitations
    - limited number of tristate buffers and longlines
    - limited number of clock buffers
  - I/O cell features
    - flip flops in I/O cells
    - special delays and slew rate control

"Black Box" Optimization

- Most basic of FPGA design optimizations
  - Essentially performing manual hardware mapping
- Procedure:
  - break down design into combinational logic black boxes
    - inputs and outputs with stuff inbetween
    - arbitrarily complex logic inside the box, but CLB doesn't care since it is a LUT anyways
  - adjust the "level" of black-boxing until you have mostly 4 or 5 input functions or 4+4 input and 2-1 mux functions

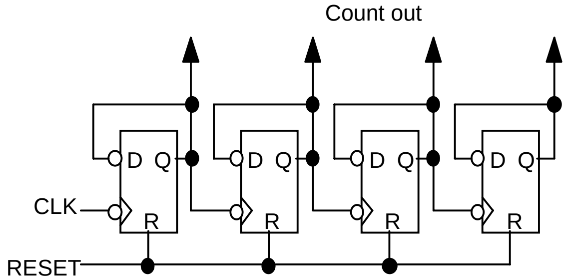
“Black Box” Example

- ALU
  - implements a 32-bit wide 2-input AND, OR, XOR, pass-through
- Example worked through on chalkboard
  - obvious implementation
    - 3 32-bit wide 2-input devices feeding into a mux or a tri-state bus
  - optimized implementation
    - 32 4-input devices: 66% or more savings in area; roughly 30-50% speed increase

Counter Design

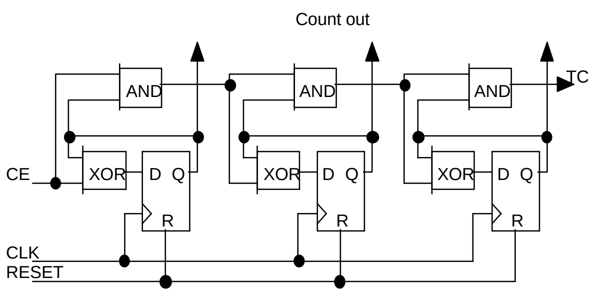
- Counters have many design options depending upon the application
  - basic ripple counter
  - ripple-carry
  - lookahead-carry
  - Johnson (mobius)
  - linear feedback shift register (LFSR)

Ripple Counter



- Ripple carry counter is not recommended in FPGA designs due to their asynchronous nature
- However, ripple carry counters are very efficient in terms of area
- k*O(n) delay growth with the number of bits, k is large (poor performance)
- Max counting states is 2^N

Ripple-Carry Counter

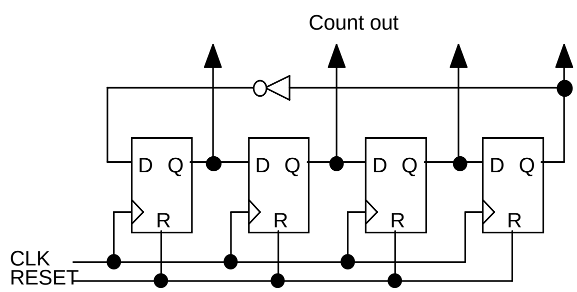


- Synchronous design
- k*O(n) delay growth with n bits, k small
- this is the basic counter provided in Xilinx libraries
- good area efficiency
- Max counting states is 2^N
- Loads or sync clears come for free in terms of area and speed

Carry-Lookahead counter

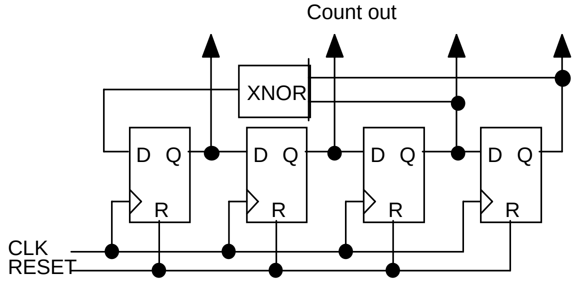
- Like ripple-carry but carry input to n^{th} counter element is computed using a full sum-of-products of the previous (n-1) bits counter state
- Can have near O(1) delay growth up to a few bits
- Good performance
- Requires a lot of gates
- Combinations of carry-lookahead and ripple-carry can be used to get the best of both worlds
- Max counting states is 2^N

Johnson or Mobius Counter



- O(1) delay growth for most applications
- Well-suited for clock division or count-limit only applications
- Non-binary counter
- Counts to 2 * n, where n is the number of flip flops
- Excellent area and speed characteristics
- Near toggle-rate speeds

LFSR Counters



- O(1) delay growth for most applications
- non-binary counter
- 2^N-1 states in a pseudorandom sequence
- excellent area and speed characteristics
- near toggle-rate speeds
- ideal for applications where count sequence is irrelevant (FIFO, timers)

LFSR application

- FIFO application
  - Count sequence doesn't matter
    - just need to address unique memory locations
    - last count value and half-full count values can be predetermined and logic created to detect these conditions
  - Saves area, increases performance
    - no carry look-ahead structures, O(1) delay growth with increasing FIFO depth

Distributed Arithmetic

- Parallel multipliers are expensive to implement in FPGAs
  - requires very wide logic functions or the use of carry-chains
  - hardware and delay growth $O(n^2)$
- Distributed arithmetic serializes multiplies using partial products
  - partial products can be computed in parallel
  - serialized multiplies fit well into FPGA architectures
  - can achieve same throughput as parallel multiplier silicon macros but with longer latency

Distributed Arithmetic

- DA takes advantage of associative and commutative properties of addition

Digit nomenclature: $A = a_n a_{n-1} \dots a_2 a_1$

In base 10:

$A * B = P_n + P_{n-1} + \dots P_2 + P_1$ where $P_n = A * b_n * 10^{n-1}$

So 42 * 121 = 42 * 1 * 100 + 42 * 2 * 10 + 42 * 1 * 1

In base 2:

$A * B = P_n + P_{n-1} + \dots P_2 + P_1$ where $P_n = A * b_n * 2^{n-1}$

So 101 * 1101 = (101 * 1) << 3 + (101 * 1) << 2 + (101 * 0) << 1 + (101 * 1) << 0

multiply operator breaks down to AND operation in one-digit binary; be careful of sign extensions for signed numbers!

Distributed Arithmetic

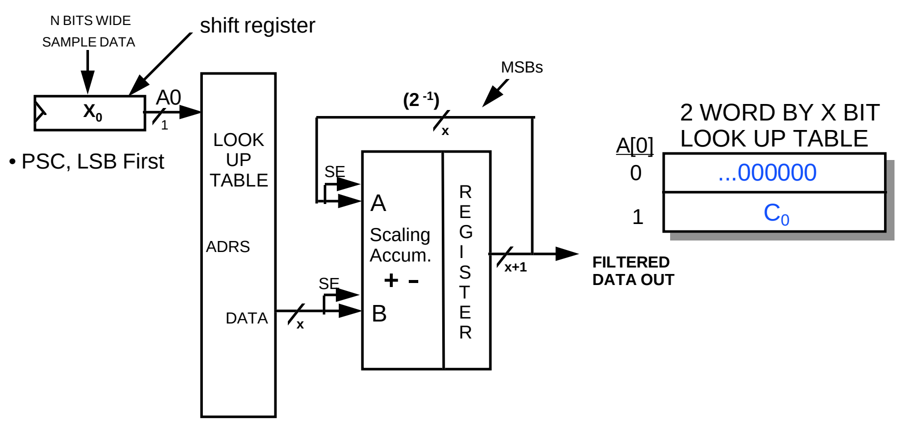
- Looking at the relation

101 * 1101 = (101 * 1) << 3 + (101 * 1) << 2 + (101 * 0) << 1 + (101 * 1) << 0

- One sees a basic functional unit- the scaling multiply. This, combined with an accumulator and bit-serial input stream (via “time skew buffer”), is the essence of the DA multiplier
- Note that the DA implementation discussed here works best for constant * variable expressions, which is ideally suited for applications such as convolutions and DSP filters
  - replace the (A * b_n) multiply kernel by a lookup-table instead of several AND gates
  - LUTs in some architectures are more efficient than AND gates
- Time to compute = number of bits in input * time to do scaling multiply

Distributed Arithmetic

- Implementation for variable * C_0; computes result in N clock cycles
  – diagram courtesy Xilinx



Distributed Arithmetic

- so what?
  - the real power of DA comes in when you try to do multiple-tap FIR filters

$$y[n] = \sum x[k] * h[n - k]$$

y[1] = x[0] * h[1] + x[1] * h[0]
Example: 101 * 011 + 110 * 100
= (101 * 0) << 2 + (101 * 1) << 1 + (101 * 1) << 0 +
(110 * 1) << 2 + (110 * 0) << 1 + (110 * 0) << 0

= ((101 * 0) + (110 * 1)) << 2 +
((101 * 1) + (110 * 0)) << 1 +
((101 * 1) + (110 * 0)) << 0

These boxes are about as complex as the boxes used in the one-tap case!

Distributed Arithmetic for a 3-Tap Filter

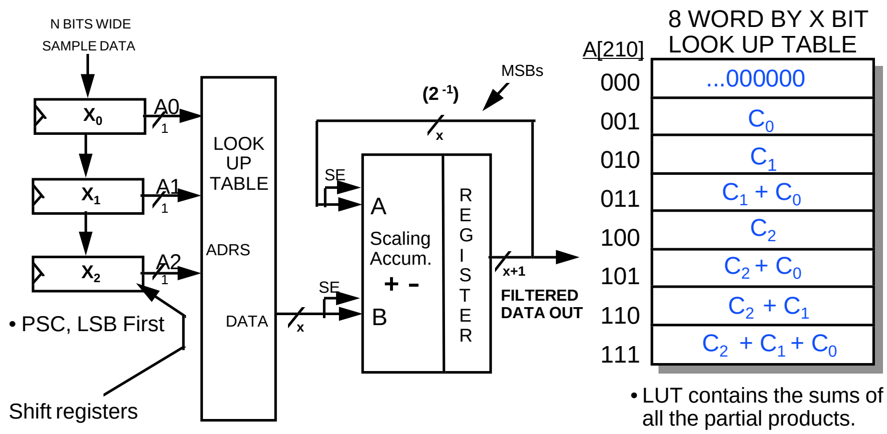
```
        -2^3 2^2 2^1 2^0         -2^3 2^2 2^1 2^0         -2^3 2^2 2^1 2^0
          1  0  0  1 (-7)          0  1  1  0 ( 6)          0  0  1  0 ( 2)
 X        0  1  1  1 ( 7)     X    0  1  0  1 ( 5)     X    0  1  1  1 ( 7)
 (        1  0  0  1    +          0  1  1  0    +          0  0  1  0)  -->  <--  0 0 0 1
 (     1  0  0  1       +       0  0  0  0       +       0  0  1  0   )  -->  <--  1 0 1 1
 (  1  0  0  1          +    0  1  1  0          +    0  0  1  0      )  -->  <--  0 0 0 1
 (0 0  0  0             +  0  0  0  0            +  0  0  0  0        )  -->  <--  0 0 0 0
 1 1 0 0 1 1 1 1 (-49)    0 0 0 1 1 1 1 0 ( 30)    1 1 0 0 1 1 1 1 ( 14)    = 1 1 1 1 1 0 1 1
                                                                                 (-5)
```

- Partial Products of equal weight are added together before being summed to next higher partial product weight.

← = Sign Extension

(slide courtesy Xilinx)

Distributed Arithmetic



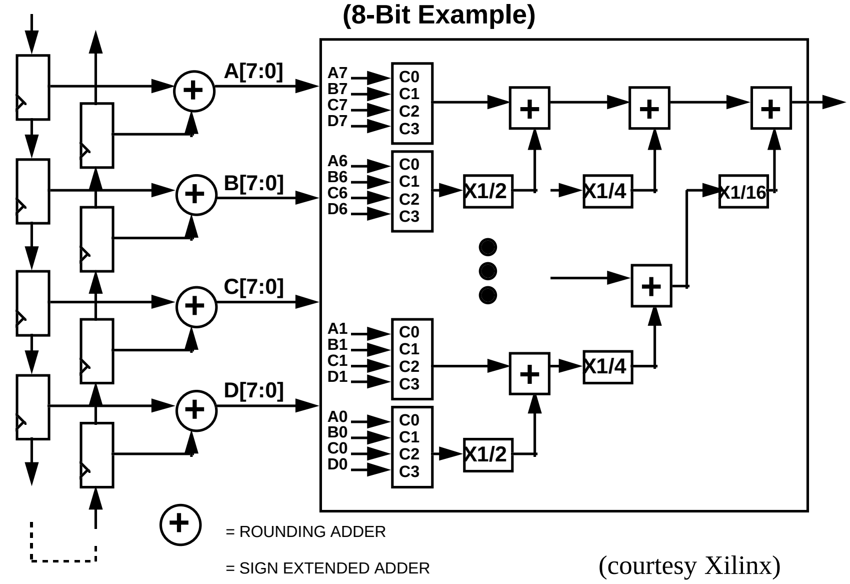
(slide courtesy Xilinx)

Distributed Arithmetic

- k $O(2^n)$ + j O(1), k is relatively small (for area)
- very close to O(1) performance scaling
- DA can be parallelized and pipelined to gain even more performance
  - Each bit can have its own LUT and adder
  - All bits computed in parallel
  - One result per clock cycle max throughput

8-Tap Symmetric Slice

(8-Bit Example)



(courtesy Xilinx)

Distributed Arithmetic

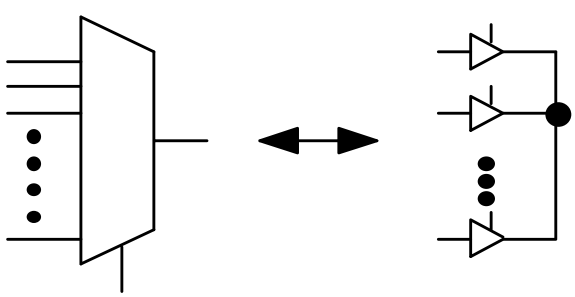
- Performance
  - Serial Distributed Arithmetic (SDA), 10-tap FIR
    - 7.8 Msamp/s for 8 bit samples @ 42 CLBs
    - 4.1 Msamp/s for 16 bit samples @ 50 CLBs
    - old numbers; probably 50% faster now
  - Parallel Distributed Arithmetic (PDA), 8-tap FIR
    - 50-70 Msamp/s for 8 bit samples @ 122 CLBs
    - pipelined, hand-optimized
  - For reference, the XC4008E has 324 CLBs (18 x 18 array)

One-Hot State Machines

- Conventional state machines use $\log_2$(states) bits to implement function
  - output is decoded from state number
  - next state is a combinational function of states
  - state transition rate limited by state number decoding and next state logic delays
- One-hot state machines use as many bits as there are states to implement function
  - only one flip flop storing "1" at any time
  - output is decoded as an OR of appropriate state FFs
  - state transition rate limited only by next state logic delays, which in many cases is zero

Miscellaneous Tricks

- Tri-state mux
  - saves on area, especially for wide muxes
  - may have better or worse performance depending on architecture and device characteristics
  - not shown in illustration is decoder for tri-state buffers



Miscellaneous Tricks

- Use IOBs to register inputs
  - gives faster setup/hold times (eliminates routing delays from setup time)
  - introduces additional latency
  - can save on logic array flip flop usage
- Inverters come for free in most architectures
- Use longlines for timing-critical signals
  - use sparingly since this is a precious resource in Xilinx 4K architectures
  - all wires in Altera "Fast Track" architecture are longlines so routes are always "fast"
- Use pipeline stages to improve pin-locked routing in Altera 8K designs
- When you can afford it, pipeline your design
  - latency versus clock speed tradeoff
- Double-wide half-rate logic (area versus speed)