How to Design Complex Digital Systems

Dr. Paul D. Franzon

Outline
1. Steps in an organized design approach
2. Achieving performance goals
3. Achieving design efficiency
4. Examples

References
1. Smith & Franzon, Chapters 6, 10
2. D. Smith, has many good examples
Steps in High Level Design

1. Determine MicroOperations to be performed on datapath units
   - e.g. adds, subtracts, multiplies, memory references, etc.

2. Design datapath units to perform these operations efficiently
   - Design to RTL level
   - Note later sections on efficiency

3. Identify control points
   - Control lines
   - Status lines

4. Determine reset/start/stop/transition actions

5. Determine control sequence
   - Generally MicroOp sequence required to perform overall task
   - Gives sequence of control events and status line responses

6. Determine control strategy
   - Mix of FSMs and/or counters

7. Verify before coding
Tricks to Achieving Performance

• Performance Metrics
  ♦ Throughput (operations/second)
  ♦ Latency (seconds from start to finish of operation)
  ♦ Performance/Area
  ♦ Performance/Power

• Exploit potential parallelism of hardware
  ● Parallel vs. Serial operation
  ● Pipelining to speed up serial operations that can not be parallelized (or don’t need parallelization)
  Theoretically, a bit-serial machine with one flop between each bit operator is the most area-performance efficient architecture
  ♦ Real-world constraints usually take you away from such a parallel/pipelined solution
Improving Performance

- **Watch those memory references**
  - Effective memory bandwidth usually determines overall performance
    - Strive to maximize utilization of memory channels
    - Memories are usually large
      - Increasing # of other functional units to keep memory 100% busy is usually a small area hit in comparison
    - For DRAM maximize use of burst/page modes
  - For DRAM maximize use of burst/page modes

- **Keep an eye out for algorithmic techniques to improve performance, for example**
  - Search algorithms that minimize memory accesses
    - Might have to trade memory size or constrain organization
  - shift instead of */2
  - etc.
Implementation Efficiency

• Look for opportunities to share resources

• Look for opportunities to shave clock cycles
  ♦ Watch critical path though

• Avoid large FSMs
  ♦ Can be slow
  ♦ Partition into smaller FSMs instead

• Watch out for those ‘large units’
  ♦ On-chip memories, floating point, multiply, divide, etc.

• Seek Simplicity
Examples

module ParallelSerialMult (clock, reset, load, multiplicandIn, multiplierIn, Product);

input clock;
input reset;
input load;
input [7:0] multiplicandIn;
input [7:0] multiplierIn;
output [15:0] Product;

reg [7:0] multiplicand;
reg [15:0] Product;
reg [3:0] Count;
reg [15:0] next_Product;
reg  next_E;
reg [7:0] AdderOut;
always@(posedge clock)
    begin
        if (load) multiplicand <= multiplicandIn;
        Product <= next_Product;
        if (!reset) Count <= 4'b0;
            else if (load) Count <= 4'h0;
            else if (~Count[3]) Count <= Count + 1'b1;
    end

always@(multiplicand or Product[15:8])
    {next_E, AdderOut} = multiplicand + Product[15:8];

always@(next_E or AdderOut or Product or Count[3] or multiplierIn or load)
    casex ({Product[0], Count[3], load})
        3'bxx1 : next_Product[15:0] = {8'b0, multiplierIn[7:0]};
        3'b100 : next_Product[15:0] = {next_E, AdderOut[7:0], Product[7:1]};
        3'b000 : next_Product[15:0] = {1'b0, Product[15:1]};
    default: next_Product[15:0] = Product[15:0];
    endcase

endmodule