95-1 Under-Graduate Project
Improving Timing, Area, and Power

Speaker: 施信毓
Adviser: Prof. An-Yeu Wu
Date: 2006/11/29
Introduction

- When design in RTL, the designer need to be aware of timing, area and power issues.
- Meeting timing is the most critical goal in design. Only optimize for power or area after timing is met.
- Synthesis tools operate in gate level, and cannot resolve all timing, area and power issues.
Timing Issues

- Timing v.s. Performance

- Latency
  - How long does it take to complete a particular operation?
  - Unit: ns, us, ms

- Throughput
  - How many operations can be completed per second?
  - Unit: Mbps, Gbps
Timing Requirement

- Synthesis tool reports timing in terms of clock cycle times.
- To meet timing, the resulting throughput need to be greater than the system specification.
- In IC design industry, the design must meet timing with margin, and using worst-case library model.
How to Improve Timing?

- **Pipelining**: Exploits temporal parallelism
  - Trade off latency to improve throughput
  - Reduce the clock cycle time
  - Insert pipeline registers without changing coherency of the data.
- **Parallel processing**: Exploits spatial parallelism
  - Execute tasks concurrently to improve throughput
  - Increase the system’s overall sampling rate
  - Incorporates multiple copies of hardware
Pipelining

original

\[ f_{\text{clock}} < \frac{1}{T_{\text{max}}} \]

\[ T_{\text{max}} \]

\[ \text{data_in} \rightarrow \text{Input Registers} \rightarrow \text{Multi-level Logic} \rightarrow \text{Output Registers} \rightarrow \text{data_out} \]

2-stage pipelining

\[ f_{\text{pipeline}} < \frac{2}{T_{\text{max}}} \]

\[ \frac{T_{\text{max}}}{2} \]

\[ \text{data_in} \rightarrow \text{Input Registers} \rightarrow \text{Comb Logic} \rightarrow \text{Pipeline Register} \rightarrow \text{Comb Logic} \rightarrow \text{Output Registers} \rightarrow \text{data_out} \]
Example 1: Simple Circuit

original

```verilog
class always@(posedge clock)
begin
    result <= A + B + C + D;
    compare <= A ^ B;
end
```

3-stage pipelining

```verilog
class always@(posedge clock)
begin
    interimResult1 <= A + B;
    Cnew <= C;
    Dint <= D;
    Dnew <= Dint;
    interimResult2 <= interimResult1 + Cnew;
    Result <= interimResult2 + Dnew;
    InterimCompare1 <= A ^ B;
    InterimCompare2 <= InterimCompare1;
    Compare <= InterimCompare2;
end
```

Critical path = 3 adders

Critical path = 1 adders
Example 2: Pipelined 16-bit Adder

Input Register: IR[32:0]

Pipeline Register: PR[7:0]

Output Register: OR[16:0]
Parallel Processing

original

```verilog
define reg [15:0] A, B, C, D, Largest;
define always @(A or B or C or D)
  begin
    if (A>B) Largest = A;
    else Largest = B;
    if (C > Largest) Largest = C;
    if (D > Largest) Largest = D;
  end
```

Parallel processing

```verilog
define reg [15:0] A, B, C, D;
define wire [15:0] Large1, Large2, Largest;

define assign Large1 = (A>B) ? A : B;
define assign Large2 = (C>D) ? C : D;
define assign Largest = (Large1 > Large2) ? Large1 : Large2;
```
Area Issues

- Area = Cost.
- During the design process, the designer should be “area aware”.
- Resource sharing

```verilog
wire [15:0] C;
wire [7:0] In1, In2, In3, InSelect;
wire MultSelect;
assign InSelect = MultSelect ? In2 : In3;
assign C = In1 * InSelect;

wire [15:0] A,B,C;
wire [7:0] In1, In2, In3;
wire MultSelect;
assign A = In1 * In2;
assign B = In1 * In3;
assign C = MultSelect ? A : B;
```
Power Consumption in CMOS

- Low power design is more and more important in today’s chip design due to heat dissipation, packaging, and portability needs.

- Power Consumption

\[ P = \sum N_{node} \times C_L \times V_{dd}^2 \times f_{clock} \]

- \( N_{node} \): switching activity
- \( f_{clock} \): clock frequency
- \( C_L \): node capacitance
- \( V_{dd} \): power supply voltage
Strategy for Low-Power Design (1/2)

- \( V_{dd} \) is technology-dependent.
- \( C_L \) can only be minimized by back-end design.
- Optimize \( f_{\text{clock}} \) and \( N_{\text{node}} \) are the most practical power reduction techniques.
Strategy for Low-Power Design (2/2)

- Reducing Clock Frequency
  - Design with clock rate that is ‘just right’
  - Clock Gating
  - Slow down clock in power saving mode

- Reducing switching activity
  - Avoid unnecessary circuit switching
  - Reducing switching activity at I/O pins
  - Use simple hardware if it gets the job done
Example : Multiplier

original

```verilog
reg [31:0] C;
reg [15:0] A,B;
always@((posedge clock)
  C = A * B;
```

Low-power design

```verilog
reg [31:0] C;
reg [15:0] A,B;
wire [15:0] Ain, Bin;
assign Ain = GoForthAndMultiply ? A : 16’h0;
assign Bin = GoForthAndMultiply ? B : 16’h0;
always@((posedge clock)
  C = Ain * Bin;
```

GoForthAndMultiply