Lecture 20
Power Optimization (Part 1)

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese461/
Power Dissipation

- Dynamic power consumption
  - switching current
- Static power consumption
  - short-circuit current
  - leakage current

\[
P_{\text{avg}} = P_{\text{dyn}} + P_{\text{short}} + P_{\text{lkg}} + P_{\text{static}}
\]
Dynamic Power

• Switching current
  - energy dissipated in half a cycle

\[
\int_0^{1/(2f)} CV \left( \frac{dV}{dt} \right) dt = \int_0^{V_{DD}} CVdV = \frac{1}{2} CV_{DD}^2
\]

- dynamic power expression
- \( \alpha \) is the average number of rising transitions in one cycle

\[
P_{dyn} = \alpha C_{out} V_{DD}^2 f
\]
Static Power

- Short-circuit currents
  - $\beta$ is the gain factor of a MOSFET
  - $V_{th}$ is the threshold voltage
  - $\tau$ is the rise/fall time

$$P_{\text{short}} = \frac{\alpha \beta}{12} (V_{DD} - 2V_{th})^3 f \tau$$
Static Power

- Leakage currents

\[ P_{\text{lk}} = (I_{\text{diode}} + I_{\text{subthreshold}}) \cdot V_{DD} \]

- Other static power
  - current flow from VDD to GND during idle time
  - historically, NMOS circuits has high static power
  - CMOS static power should be 0
  - might result from bus conflict where multiple drivers attempt to drive a signal to different logic values
Low Power Design Methodologies

- Adapt process technology
  - reduce capacitance
  - $C_{fo}$ is the input capacitance of fan-out gates
  - $C_w$ is the wire capacitance
  - $C_p$ is the parasitic capacitance

$$C_{out} = C_{fo} + C_w + C_p$$

- reduce leakage current
- reduce supply voltage

$$T_d = \frac{C_{out}V_{DD}}{I} = \frac{C_{out}V_{DD}}{\eta\left(\frac{W}{L}\right)(V_{DD} - V_{th})^2}$$
Low Power Design Methodologies

- Reduce switch activity
  - minimize glitches

- nodes logically deeper more prone to signal glitches
Low Power Design Methodologies

- Reduce switch activity
  - minimize glitches
  - minimize number of operations
  - example: vector quantization (VQ) algorithm
  - $X_i$ are the elements of the input vector
  - $C_{ij}$ are the elements of the codebook vector

$$D_i = \sum_{j=0}^{15} \left( X_j - C_{ij} \right)^2$$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># of Memory Access</th>
<th># of Multiplications</th>
<th># of Adds</th>
<th># of Subs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Search</td>
<td>4096</td>
<td>4096</td>
<td>3840</td>
<td>4096</td>
</tr>
<tr>
<td>Tree Search</td>
<td>256</td>
<td>256</td>
<td>240</td>
<td>264</td>
</tr>
<tr>
<td>Differential Tree Search</td>
<td>136</td>
<td>128</td>
<td>128</td>
<td>0</td>
</tr>
</tbody>
</table>
Low Power Design Methodologies

• Reduce switch activity
  - minimize glitches
  - minimize number of operations
  - low power bus

• Examples
  - one-hot coding
  - gray coding
  - bus-inversion coding

\[
(B(t), \text{INV}(t)) = \begin{cases} 
  (b(t), 0) & \text{if } H \leq N/2 \\
  (b'(t), 1) & \text{Otherwise}
\end{cases}
\]

\[N: \text{number of bus lines, } H: \text{Hamming Distance}\]
Minimize Data Transition on Bus

// Code that resets the bus to default status after valid gets de-asserted

always @(posedge clk or negedge reset)
begin
  if (!reset)
    data_bus <= 16'b0;
  else if (data_bus_valid)
    data_bus <= data_o;
  else
    data_bus <= 16'b0;
end

// Code that holds the bus to its previous value after valid gets de-asserted

always @(posedge clk or negedge reset)
begin
  if (!reset)
    data_bus <= 16'b0;
  else if (data_bus_valid)
    data_bus <= data_o;
end

<table>
<thead>
<tr>
<th>Bus Width</th>
<th>Total random data input</th>
<th>Number of Transitions without coding (A)</th>
<th>Number of transition with bus invert coding (B)</th>
<th>Percentage improvement in bus invert coding against non-coded data (B) / (A) * 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit</td>
<td>5000000</td>
<td>8005314</td>
<td>6879054</td>
<td>14.06 %</td>
</tr>
<tr>
<td>64-bit</td>
<td>1000000</td>
<td>32000980</td>
<td>28513273</td>
<td>10.89</td>
</tr>
</tbody>
</table>
Bus Coding

Less switching activity

b(t): Source word
B(t): Code word
Bus Invert Coding

<table>
<thead>
<tr>
<th>Binary (31 Trs)</th>
<th>BIC (19 Trs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101010</td>
<td>00101010</td>
</tr>
<tr>
<td>00111011</td>
<td>00111011 0</td>
</tr>
<tr>
<td>11010100</td>
<td>00101011 1</td>
</tr>
<tr>
<td>11110100</td>
<td>00001011 1</td>
</tr>
<tr>
<td>00001101</td>
<td>00001101 0</td>
</tr>
<tr>
<td>01110110</td>
<td>10001001 1</td>
</tr>
<tr>
<td>00010001</td>
<td>00010001 0</td>
</tr>
<tr>
<td>10000100</td>
<td>10000100 0</td>
</tr>
</tbody>
</table>

00101010
00111011
11010100
11110100
00001101
01110110
00010001
10000100
Low Power Design Methodologies

• Reduce switch activity
  - minimize glitches
  - minimize number of operations
  - low power bus (state machine encoding)
  - scheduling and binding optimization

control-data-flow graph (CDFG)
Low Power Design Methodologies

- Power down modes
  - clock gating
  - enabled flip-flops
  - memory partitioning
  - power gating
How Effective is Clock-Gating?

- 90% FF clock-gated
- 70% power reduction
Low Power Design Methodologies

- Voltage optimization and scaling
  - multi-Vth optimization
Low Power Design Methodologies

- Voltage optimization and scaling
  - multi-voltage domain
  - dynamic voltage frequency scaling
Resource Sharing

• Save area and power

```verilog
// Example where resource sharing is not possible
always@(in1 or in2 or sel)
  if(sel)
    out1 = in1 + in2;
  else
    out1 = 4'b0;

always@(in3 or in4 or sel)
  if(!sel)
    out2 = in3 + in4;
  else
    out2 = 4'b0;

// Example where resource sharing is possible
always@(in1 or in2 or sel or in3 or in4)
  if(sel)
    begin
      out1 = in1 + in2;
      out2 = 4'b0;
    end
  else
    begin
      out1 = 4'b0;
      out2 = in3 + in4;
    end
```
Questions?

Comments?

Discussion?