HDL Design Methods for Low-Power Implementation

By Kaushal Buch, eInfochips

Abstract

Increasing clock frequency and a continuous increase in the number of transistors on chip have made implementing low power techniques in the design compulsory. These low power techniques are being implemented across all levels of abstraction - system level to device level. Here, approaches related to front-end HDL based design styles, which can reduce power consumption, have been mentioned. As is known, power dissipation has a direct relation with the clock frequency and dynamic power also depends upon the rate at which the data toggles for a given circuit. The design styles mentioned here, focus on several areas of designing using HDL, which are many times not considered significant, as they do not affect the functionality. The guidelines mentioned here are quite simple to implement and mostly unravel techniques that are considered quite trivial, yet have a significant impact on the overall power consumption.

Low Power implementation approaches

Power dissipation in a CMOS transistor depends on the capacitance, supply voltage and the rate at which the data toggles.

$$P = f\, C_{load}\, V_{DD}^{2}$$

Where,

- C_{load} is the load capacitance of the CMOS transistor
- V_{DD} is the supply voltage
- f is the frequency at which the data transition happens.

If P_{dt} describes the probability of data transition then

$$f = P_{dt} * f_{clk.}$$

Example: For a random data P_{dt} = 0.5 so for a clock frequency of 50MHz, the value of f would be 0.5 * 50 MHz = 25 MHz

An efficient and high quality HDL code can reduce unwanted transitions and can save substantial amount of power in the design. Also logic optimization techniques like removing redundant logic and properly sharing the resource in design also helps in power reduction.

1. Minimizing data transitions on bus – In many cases the data on the bus keeps on transitioning from one value to another because there is no default state for assigning a constant value. This may not affect the design functionally as there may be some handshaking signal which indicates that the data is valid. But the transitions on data bus consume power.

```
// 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
```

2. Resource sharing – The RTL coding should be carried out in a manner that there are no unwanted or redundant logic elements. Any logic element will contribute to power consumption as it has a capacitance attached to it and transitioning of data through that logic will lead to power dissipation.

```
// 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
```

3. Avoiding unnecessary transition of signal – It is seen in many designs that certain signals transit when they are not required to, but they are not detected in functional verification, as they satisfy the logical requirements. Such signals, if checked properly and if the logic is tweaked to suppress those unwanted transitions, can also help avoid utilization of power.

4. State Machine Encoding – It is a well known fact that one-hot and Gray encoding consume lesser power as compared to binary encoding. This is because one-hot and gray encodings have only a single bit change while going from one state to another.

5. Control over counters – Counters are normally designed so that they can start and stop as per requirement. Certain times, due to improper coding, all the start and stop conditions are not taken care of and the counter may unnecessarily keep on counting.

```
//Example of unnecessary counter transitions

always@(posedge clk or negedge reset)
begin
   if(!reset)
   cnt <= 4'b0  ;
   else  if((cnt  ==  4'b0111)  |
cntr_reset)
   cnt <= 4'b0 ;
   else
    cnt <= cnt + 1'b1 ;
      end
```

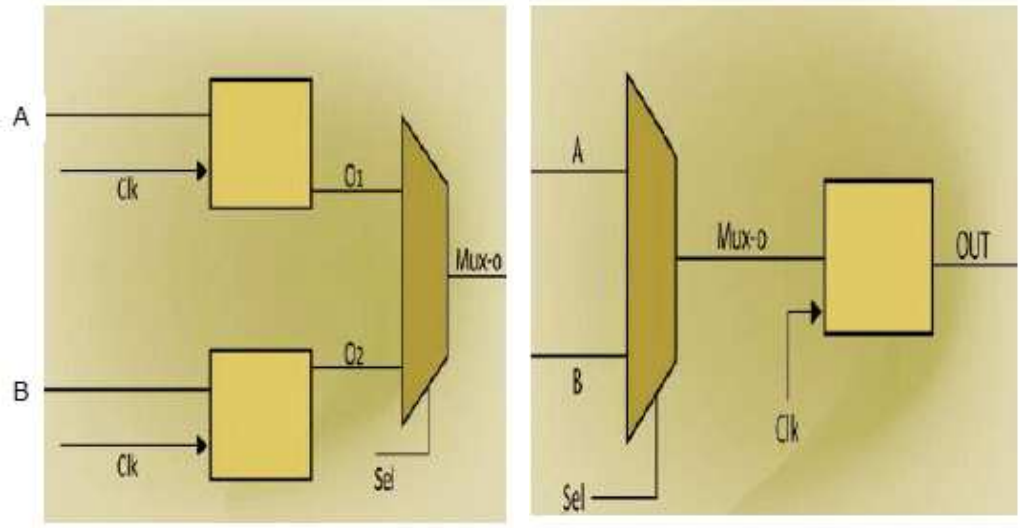
```
//Example that removes unnecessary counting transitions

always@(posedge clk or negedge reset)
begin
   if(!reset)
   cnt <= 4'b0  ;
   else if(cntr_reset)
   cnt <= 4'b0 ;
   else if(cnt < 4'b0111)
    cnt <= cnt + 1'b1 ;
     end
```

For example, for a random probability data (P = 0.5) and clock frequency of 100 MHz, the transition frequency would be around 50 MHz. For a bus capacitance of 25 pF and supply voltage of 1.2 V, this would result in 1.8 mW power consumption.

6. Allow synthesis optimization – Certain constraints and coding styles can be followed which reduce the area utilization or logic optimization. This is because extra logic will add extra capacitance and in turn will consume more power. Also, one way of checking redundant hardware generation is by tactfully analyzing the code coverage reports.

7. Register Retiming – Register timing is a concept mostly used in improving timing by reordering the combinational and sequential logic in a given data path. However in certain cases, there is a saving of logic and thus can help improve upon power consumption. Of course, this is possible only if the design can support the additional timing overhead.



Without Retiming **With Retiming**

8. Using Gray coding for addressing memories – It is seen that addressing memories via gray coding significantly reduces the power as there are lesser number of transitions that the address counter performs. A detailed explanation and trade-offs of the same is mentioned in topic 3.

9. Using Bus Invert Coding for I/Os or long data paths– Bus invert coding (topic 2) is a technique in which if the hamming distance between the current data and the next data is more that N/2 (where N is the bus width), then one can invert the bits and send it, so as to minimize the number of transitions on the bus. In that case a control bit goes along with the data to indicate the receiving end, whether the data is inverted or not. The following are the results of a simulation carried out to understand the reduction in the number of transitions due to bus invert coding.

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

10. Using systolic or pipelined design for DSP implementation – A detailed understanding of systolic architecture and pipelined architecture for implementing a DSP block are mentioned in (topic 4). Pipelining reduces power by registering the inputs at regular intervals and thereby reduces the overall net-lengths and minimizes glitches. Systolic architectures have high modularity and help reduce long interconnect path delays. Depending on the requirements of latency and hardware, one can choose one of these approaches

Conclusion –

A significant reduction in the power dissipation was observed by following the techniques described in this paper. A good practice would be to not only verify the design for its functional adherence, but also verify it from the low power perspective, by employing methods and strategies that target detection of unwanted transitions and logic redundancy.

References –

[1] Gary Yeap, "Practical Low power Digital VLSI design", Kluwer Academic Publishers, 1998.

[2] Mircea R. Stan and Wayne P. Burleson, "Bus Invert Coding for Low-Power I/O", IEEE Transactions on VLSI systems, Vol.3, No. 1, March 1995, pp 49 – 58.

[3] Hichem Belhadj, Vishal Aggrawal, Ajay Pradhan, Amal Zerrouki, "Power Aware FPGA design – Part 3", Programmable Logic Design Line, 17th February, 2009.

[4] Roger Woods, John McAllister, Gaye Lightbody and Ying Yi, "FPGA implementation of signal processing systems", Wiley, 2008.

About the Author –

Kaushal Buch has been working with eInfochips in the area of FPGA / ASIC design for about 4 years. His work involves defining SoC architectures for IPs, digital design, RTL development, synthesis and timing closure. Kaushal holds a graduate degree in Electronics and Communications engineering from Nirma Institute of Technology, Gujarat University, Ahmedabad, India. His areas of interest are SoC microarchitecture, DSP implementation on SoCs, high speed computations on chip, low power design, probabilistic power analysis, design synthesis and timing analysis.