Homework 2
Adder/Subtractor/Multiplier Designs: Non-Pipelined versus Pipelined

Handout: 2025/10/13
Due: 3 weeks later

1. Multiply-Add Designs with and without Common Factor Extraction
   Write two different designs, d=a*c+b*c and d=(a+b)*c, as shown below

```
module multiply_add (input [7:0] a, b, c, output [15:0] d);
assign d=a*c+b*c;    // two 8x8 multipliers ; one 16-bit adder
endmodule
```

```
module multiply_add_common_factor (input [7:0] a, b, c, output [15:0] d);
assign d=(a+b)*c;    // one 8-bit adder;    one 8x8 multiplier;
endmodule
```

   Compare the delay, area and power of the synthesized circuits for the above two designs under the same cycle time constraint. Provide your observations on the synthesized gate-level netlists.

2. Adder/Sutractor
   Compare the synthesis results (area, delay, power) of the following different codes for modeling adder/subtractor under the same synthesis constraint. Provide your observations on the synthesis results. .

```
module add_sub_1 (input [7:0] a, b, input s, output [15:0] d);
wire [7:0] tmp;
assign tmp = s ? b : -b;    // one 2-to-1 multiplexer for possible negation of b
assign d = a + tmp;      // an adder
endmodule
```

```
module add_sub_2 (input [7:0] a, b, input s, output [15:0] d);
assign d = s ? a+b : a-b;    // one adder, one subtractor, one 2-to-1 multiplexer
endmodule
```

```
module add_sub_3 (input [7:0] a, b, input s, output reg [15:0] d);
always @ (a or b or s)      // one adder, one subtractor, one 2-to-1 multiplexer
    if (s == 1) d = a+b; else d=a-b;
endmodule
```

3. Non-pipelined Add/Sub-Multiply

   Use Verilog to model a non-pipelined unit that performs addition/subtraction and multiplication with the following input/output ports.

   module hw2_nonpipe (input [7:0] a, b, c, input s, output [15:0] d);

   The hardware design performs d=(a+b)*c if s=1 and d=(a-b)*c if s=0. You can use continuous assignment **assign** … with ternary operator **?:** or. An alternative is using behavioral modeling **always @ (*) … if … else ….** Note that use the coding methods that leads to the smallest area based on the experiments in the previous problems.

4. RTL Simulation of Non-Pipelined Design

   Write a testbench to verify the non-pipelined design.

5. Logic Synthesis with Synopsys Design Compiler

   Use Synopsys Design Compiler (DC) to synthesize the RTL design into gate-level netlists with proper synthesis constraint. Verify again the synthesized gate-level netlists. What is the critical path delay? What is the area? What is the power?

6. Pipelined Design

   Use Verilog HDL to model a ***pipelined*** unit with two pipeline stages to compute the above addition/subtraction and multiplication with the following input/output ports.

   module hw2_pipe (input [7:0] a, b, c, input s, reset, output [15:0] d);

   The first pipeline stage performs addition or subtraction of two 8-bit numbers a, b depending on whether the control signal s is logic 1 or 0. The second pipelined stage performs the multiplication of c and the results in the first stage. Note that the three inputs a, b, c, s are provided as the input of the first pipelined stage and the output d appears at the second pipeline stage. *All the registers should be able to be initialized to zero using an asynchronous reset signal before execution.* The following statement describes an active-high asynchronous resettable flip-flop:

   ```
   always @ (posedge clk or posedge reset)
       if (reset) q <= 1'b0; else q <= d;
   ```

   Hint: the pipelined design contains two always blocks with each always block modeling one pipeline stage. Each always block looks like the following

   always @ (posedge clk or posedge reset) begin …. end

   It is better to use two separate modules to model the two pipelined stages, so that

you can more easily identify the delay of each pipeline stage.

7. Simulation of the Pipelined Design

   Verify the pipelined design in RTL by writing a testbench that provides input streams of a, b, c, s, and compare the output stream d with the expected values. Note that a data stream is a sequence of data samples. That is, the testbench provides a sequence of input data, and then verify the sequence of generated output data.

8. Logic Synthesis of the Pipelined Design

   Use Synopsys DC to synthesize the RTL design into gate-level netlists with proper synthesis constraint. Verify again the synthesized gate-level design using the same test patterns as used in the RTL simulation.

   What is the total area?

   What is the total power?

   What is the delay of the first pipeline stage T_1?

   What is the delay of the second pipeline stage T_2?

   What is the critical path delay $T=\max(T_1, T_2)$?

   What is the throughput (number of operations per second) 1/T?

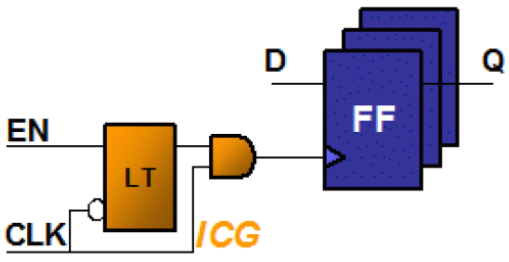
   What is the maximum working frequency of the design $f_{max}=1/T$?

   What is the total latency $T_{latency}= T_1+T_2$?

   Throughput (unit of operations per second) is the speed performance to process a sequence of input data. Latenc denotes the required time to generate the first output data. Note that pipelining can increase throughput rate, NOT latency, when executing a sequence of input data samples.

9. Clock Gating

   Modify the previous pipelined design using the clock gating technique so that the multiplication in the second pipeline stage is disabled with outputs set to zero whenever the input c is zero. That is, the two inputs of the multiplier in the second stage are kept unchanged when c=0, in order to reduce the dynamic switching power of the multiplier. The following figure shows a safe and robust clock-gating circuit to keep the contents of registers unchanged.



10. Simulation and Synthesis of Clock-Gated Pipelined Design

    Simulate the clock-gated pipelined design in both RTL and synthesized gate-level.

Measure the delay, area and power. Compare with the non-clock-gating pipelined design. Comment on the differences. Note that Synopsys DC reports delay and power information without user-specified inputs. Thus, the power information cannot truly reflect the actual power of the clock-gated design, especially when input c has high chance of being zero.

11. Synopsys PrimeTime Simulation

In addition to the *static timing analysis* in the Synopsys Design Compiler, another *dynamic timing analysis* tool called PrimeTime can also be used to extract the delay and power information by explicitly providing a sequence of input patterns. Use PrimeTime with properly specified sequence of input patterns to measure the delay and power of the non-pipelined, pipelined, and clock-gated designs. In order to see the power-saving effect of clock-gating, you should specify the sequence of input patterns with a, b, c randomly generated *where the probability of input signal c=0 is 1/2.* Compare the delay and power of PrimeTime with those obtained from synthesis report.

12. Comparison

Use a table (see below) to compare the area/delay/power of the non-pipelined pipelined, and clock-gated designs with delay/power results from Design Compiler and from PtimeTime. In the table, the total area is divided into combinational part and sequential part. The delay means critical path delay which is the reciprocal of maximum working frequency. The latency is the delay from the primary inputs to the primary outputs, i.e., the sum of the pipelined delays in all the pipeline stages if applicable. Note that the delay and power from Design Compiler and from PrimeTime might be different. Explain the reasons for the differences.

	Area (um^2)			Delay	Latency	Power (W)		
	CL	SL	Total	(ns)	(ns)	dynamic	leakage	total
Non-pipelined (DC)								
Non-pipelined (PrimeTime)								
Pipelined (DC)								
Pipelined (PtimeTime)								
Clock-gated (DC)								
Clock-gated (PrimeTime)								

13. Implementation using FPGA

Use Xilinx tools to Synthesize the RTL models of the non-pipelined, pipelined, and clock-gated designs and implement them onto a selected FPGA chip and verify the designs again using the same test patterns used in RTL simulation. What is the maximum working frequency? What is the FPGA resource (LUT, DSP, BRASM, …) used ?

Report Requirement

1. Design Complier(70%)

submitted files should include: (包括 non-pipelined & pipelined design)

I. Verilog RTL code & testbench**(10%)**

II. gate-level code (ddc、.v、.sdf、.sdc) & testbench**(15%)**

homework report file should include : (包括 non-pipelined & pipelined design)

I. RTL and gate-level simulation waveforms including explanation (RTL 波形 & gate-level 波形並解釋) **(10%)**

II. synthesis area information and critical path (比較數據&截圖) **(10%)**

III. timing information (critical path delay) in the two pipeline stages (測量二階不同 pipeline 內的 timing 資訊) **(10%)**

IV. automatic verification mechanism using either testbench or other software such as C or Python (自動驗證運算結果，可使用 testbench、C 等軟體程式驗證) **(15%)**

2. Xilinx Vivado (30%)

Name the project as HDL_HW2_MXXXXXXXXX

submitted files should include :

I. HDL_HW2_MXXXXXXXXX.xpr.zip**(15%)**

II. xdc、wcfg **(2.5%)**

report should include:

I. waveforms of both behavioral level and post-implementation, including explanations (Behavior 波形 & post-implement 波形並解釋) **(10%)**

II. Snapshot of project summary-overview, including utilization and timing (Project Summary-Overview 截圖，含 Utilization、Timing) **(2.5%)**

Please submit only Report.pdf to 網路大學. Place all Vivado files in your 'HW2' folder and upload the entire folder to the 'Homework-Submit' server folder." **(Please refer to the TA's slides for more information).**