**Laboratuvar Çalışması 0x2**

Aritmetik Mantık Devreleri

Bu labın amacı

* Aritmetik devreleri oluşturabilmek
* Kombinasyonel devreleri senkronize edebilmek

## Problemler

### Problem 1 - Hafıza elemanları karşılaştırması

HDL ifadelerini kullanarak Latch, Rising-edge triggered Flip-Flop ve Falling-edge triggered Flip-Flop oluşturun. (flop.sv dosyasında mevcut). Herbirinin girişine aynı sinyale bağlayın ve çıkışlarını ayrı sinyallere bağlayın. Ekte verildiği gibi bir testbench kullanarak devrenizi test edin ve yorumlayın. (RTL ve Tech Map şemalarını karşılaştırın.)

### Problem 2 - ALU Tasarımı

1. 32-bitlik NZVC destekli ALU tasarlayın. Tasarlarken always\_comb blogu ve if statement lar kullanabilirsiniz. ALU nun opcode ları ve gerçeklemeniz gereken fonksiyonları Tablo 1 de verilmiştir.
2. Latch oluşmaması için always\_comb bloğu içinde kullandığınız bütün atamaların her bir dalda olduğuna emin olun. Örnek olarak if (...) a = 4; gibi bir atamada if bloğu içine girmezse kod a ya atama yapılmadığı için latch oluşacaktır.
3. Flagların hangi koşullarda olduğunu iyi anlayın, mesela negative flag sonucun sadece 31. bitine bakacak, zero flag sonucun bütün bitleri 0 ise yanacak, carry sadece toplama, çıkarma ve shift operasyonlarında oluşabilir, vs. Shift ile alakalı Carry oluşturma şemaları EK B de verilmiştir.
4. Devrenizi basit birkaç girişle test edin ve doğruluğunu gözlemleyin.
5. Size verilen yeni testbench ile devrenizi test edip hatalarınızı düzeltin. İki testbench arasındaki farkı yorumlayın. (<https://github.com/fcayci/sv-digital-design/blob/master/tb/tb_alu_auto.sv>) ve txt dosyası.
6. Son devrenin sentezleme sonuçlarını raporunuza ekleyip yorumlayın. (Şemalar, kullanım vs.)

module alu (

input logic [31:0] a, b,

input logic [ 2:0] op,

output logic [31:0] s,

output logic n, z, v, c,

output logic hata

);

|  |  |  |
| --- | --- | --- |
| **op**2 | **operasyon** | **açıklama** |
| 000 | addition | A + B |
| 001 | subtraction | A - B |
| 100 | shift left logical | A sinyalini B kadar sola kaydır1 |
| 010 | xor | A XOR B |
| 101 | shift right logical | A sinyalini B kadar sağa kaydır1 (0 padded) |
| 110 | shift right arithmetic | A sinyalini B kadar sağa kaydır1 (sign padded) |
| 011 | or | A OR B |
| 111 | and | A AND B |

Tablo 1

Not 1: B nin son 5 bitine bakarak kaydırma yapılacak. Geri kalan bitler ihmal edilebilir.

Not 2: **Eğer yukarıdaki oplar haricinde bir giriş gelirse, hata biti oluşturulacak.**

### Problem 3 - Senkron Tasarım ve Zamanlama

Problem 2 de tasarladığınız ALU nun giriş ve çıkışlarının hepsine birer register bağlayın. Daha sonra devrenin timing analizini yaparak, max frekansının ne kadar olduğunu, critical path inizin ne çıktığı hakkında yorum yapın.

## IV. Ekler

### EK A - Problem 1 testi için örnek bir testbench. Clk generation.

|  |
| --- |
| /\* tb\_ornek.sv  \*  \* Hazırlayanlar:  \* Furkan Çaycı  \*  \* Notlar:  \* ELM235 2021 Bahar Lab2 testbench örneği  \*  \*/  // Time Units and resolution of the simulation  `timescale 1ns/1ps  module tb\_ornek ();  logic d, clk;  logic q1, q2, q3;  ornek uut0(...);    // always blogu surekli calistirilacak  // clock oluşturmak için clk sinyalini bir lojik  // seviyeye çekip belli bir süre bekleyip, geri çekiyoruz  // aşağıdaki haliyle 20ns lik bir clk oluşur.  always  begin  clk = 0; #10; clk = 1; #10;  end  initial begin  d = 0; #7; d = 1; #5; d = 0; #2;  d = 1; #4; d = 0; #3; d = 1; #3;  d = 0; #2; d = 1; #2; d = 0; #4;  d = 1; #2; d = 0; #2; d = 1; #6;  d = 0; #10;  $stop; // stop the simulation  end  endmodule |

EK B - Shift Operatörleri için carry oluşturma

