Logika cyfrowa

Praktyczna lista zadań nr 8

Termin: 11 kwietnia 2022 godzina 30:00

Uwaga! Poniższe zadania należy rozwiązać przy użyciu języka SystemVerilog, sprawdzić w DigitalJS oraz wysłać w systemie Web-CAT na SKOS. Należy pamiętać, aby nazwy portów nadesłanego modułu zgadzały się z podanymi w treści zadania. Wysłany plik powinien mieć nazwę `toplevel.sv`. **Nie przestrzeganie tych zasad będzie skutkować przyznaniem 0 punktów.**

1. Zaimplementuj generator sygnału PWM (Pulse Width Modulation – modulacja szerokości impulsu). Generator taki składa się z licznika, rejestru porównania, rejestru wartości szczytowej oraz układów porównujących. Rejestry oraz licznik są podłączone do wspólnego sygnału zegara. Działanie jest następujące:

   - licznik rozpoczyna odliczanie od wartości 0;
   - gdy wartość licznika jest większa lub równa wartości szczytowej, licznik jest resetowany do 0 (synchronicznie);
   - sygnał wynikowy jest w stanie niskim, gdy wartość licznika jest większa lub równa wartości rejestru porównania, w przeciwnym wypadku jest w stanie wysokim;
   - do każdego z elementów składowych (włącznie z licznikiem) można załadować nową wartość, nie zatrzymując przy tym działania licznika (chyba, że właśnie licznik jest ładowany).

   Układ powinien mieć następujące wejścia i wyjścia:

   - `clk` – wejście sygnału zegara (wyzwalanie zboczem narastającym),
   - `d` – szesnastobitowe wejście ładowania,
   - `sel` – dwubitowe wejście wybierające ładowany rejestr, znaczenie:
     - 0 – nic nie jest ładowane,
     - 1 – ładowany jest rejestr porównania,
     - 2 – ładowany jest rejestr wartości szczytowej,
     - 3 – ładowany jest licznik,
   - `cnt` – szesnastobitowe wyjście bieżącej wartości licznika,
   - `cmp` – szesnastobitowe wyjście bieżącej wartości rejestru porównania,
   - `top` – szesnastobitowe wyjście bieżącej wartości rejestru wartosci szczytowej,
   - `out` – jednobitowy sygnał wyjściowy generatora.

   W tym zadaniu nie używaj modeli bramkowych przerzutników ani liczników. Do modelowania elementów synchronicznych wykorzystaj bloki `always_ff`.

   Rozwiązanie można przetestować przy użyciu poniższego skryptu Lua.

```
sim.setinput("d", 0)
for sel = 1, 3 do
  sim.setinput("sel", sel)
  sim.wait(sim.posedge("clk"))
end
sim.sleep(10)
for x = 1, 100 do
  sel = math.random(0, 3)
  prevcnt = sim.getoutput("cnt"):tointeger()
  prevtop = sim.getoutput("top"):tointeger()
  newd = math.random(0, 65535)
  sim.setinput("d", newd)
  sim.setinput("sel", sel)
```

```
  sim.wait(sim.posedge("clk"))
  sim.sleep(10)
  cnt = sim.getoutput("cnt"):tointeger()
  cmp = sim.getoutput("cmp"):tointeger()
  top = sim.getoutput("top"):tointeger()
  if sel == 3 then
    assert(cnt == newd, "Error: cnt didn't load to " .. newd)
  elseif prevcnt >= prevtop then
    assert(cnt == 0, "Error: cnt didn't roll over to 0")
  else
    assert(cnt == prevcnt + 1, "Error: cnt didn't increment")
  end
  if sel == 2 then
    assert(top == newd, "Error: top didn't load to " .. newd)
  elseif sel == 1 then
    assert(cmp == newd, "Error: cmp didn't load to " .. newd)
  end
  assert(sim.getoutput("out") == vec.frombool(cnt < cmp),
    "Error: PWM output invalid")
end
print("OK!")
```