

Logika cyfrowa

Praktyczna lista zadań nr 9+10

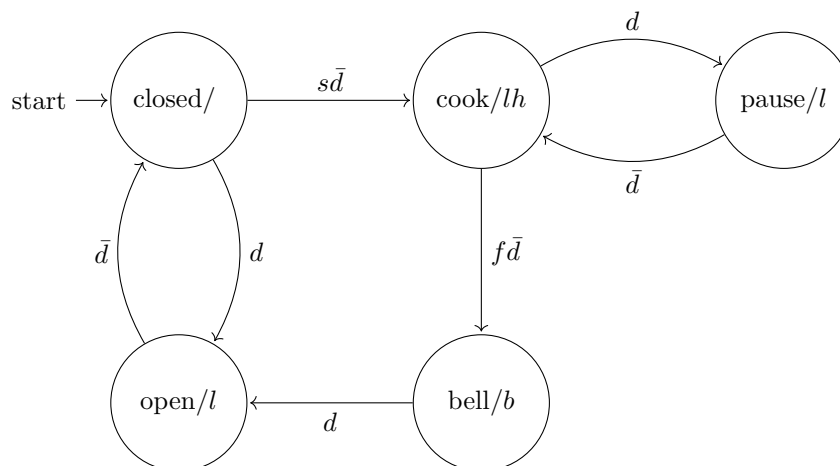
Termin: 15 maja 2024 godzina 30:00

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 skutkowało przyznaniem 0 punktów.**

1. Zaprojektuj prosty układ sterujący kuchenką mikrofalową. Powinien mieć następujące wejścia i wyjścia:

- `clk` – wejście zegara,
- `nrst` – zanegowane wejście resetu asynchronicznego (stan niski resetuje),
- `door` – jednobitowe wejście sygnalizujące otwarcie drzwiczek,
- `start` – jednobitowe wejście sygnalizujące wciśnięcie przycisku,
- `finish` – jednobitowe wejście sygnalizujące upływanie czasu przeznaczzonego na podgrzewanie,
- `heat` – jednobitowe wyjście sterujące magnetronem i napędem obracającym talerz,
- `light` – jednobitowe wyjście sterujące światłem wewnątrz kuchni,
- `bell` – jednobitowe wyjście sterujące sygnałem dźwiękowym sygnalizującym koniec podgrzewania.

Wykorzystaj automat Moore'a opisany poniższym diagramem stanów. Litery *d*, *s*, *f*, *h*, *l*, *b* oznaczają wejścia i wyjścia o nazwach zaczynających się od tych liter. Brak narysowanego przejścia dla niektórych wejść na poniższym diagramie oznacza przejście „pętelkę” do tego samego stanu („zignorowanie” wejścia).



2. Zaimplementuj kalkulator obliczający wyrażenia w odwrotnej notacji polskiej, operujący na 16-bitowych liczbach ze znakiem. Do przechowywania stosu wykorzystaj pamięć RAM $1K \times 16$ z jednym portem do odczytu i jednym portem do zapisu. Szczyt stosu przechowuj w osobnym rejestrze; w ten sposób unikniesz konieczności wykonywania dwóch odczytów pamięci w jednym cyklu zegara. Działanie układu powinno być sterowane narastającymi zboczami sygnału zegara. Każde zbocze zegara powinno wyzwać wykonanie jednej operacji kalkulatora.

Układ powinien mieć następujące wejścia i wyjścia:

- `nrst` – zanegowane wejście resetu asynchronicznego (stan niski resetuje), rejestry `out` i `cnt` powinny być inicjowane zerami, nie należy czyścić pamięci stosu podczas resetu,
- `step` – wejście sygnału zegara, wyzwalanie zboczem narastającym,
- `d` – 16-bitowe wejście ładowania,

- **push** – wejście 1-bitowe wybierające operację odłożenia wejścia na stos, znaczenie:
 - wartość wysoka – odłożenie wejścia **d** na stos,
 - wartość niska – wykonanie operacji opisanej wejściem **op**,
- **op** – dwubitowe wejście wybierające operację wykonywaną gdy **push**=0, znaczenie:
 - 0 – brak operacji,
 - 1 – minus unarny (zamiana wartości na szczycie stosu na przeciwną),
 - 2 – dodawanie,
 - 3 – mnożenie,
- **out** – 16-bitowe wyjście pokazujące wartość na szczycie stosu,
- **cnt** – 10-bitowe wyjście pokazujące liczbę elementów na stosie.

W tym zadaniu nie używaj modeli bramkowych przerzutników, liczników, sumatorów ani multiplikatorów. Pamięć zaimplementuj przy użyciu tablicy SystemVeriloga, zgodnie z zasadami podanymi na wykładzie.

W przypadku nieprawidłowej operacji na stosie (operacja binarna na mniej niż dwóch elementach, operacja unarna na pustym stosie, przepełnienie stosu) wartość wyjścia **out** może być dowolna. Jednak wyjście **cnt** **nie powinno się zawijać**. Na przykład, po próbie wykonania operacji binarnej gdy **cnt**=0, nowa wartość **cnt** musi nadal wynosić 0.

Przykładowe działanie układu jest opisane w poniższej tabeli. Wiersz *t* zawiera stan układu po *t*-tym zboczu narastającym zegara, powstały ze stanu i wartości wejść z wiersza *t* – 1.

t	d	push	op	out	cnt
0	2	1	x	0	0
1	3	1	x	2	1
2	x	0	1	3	2
3	x	0	3	-3	2
4	6	1	x	-6	1
5	x	0	2	6	2
6	x	x	x	0	1

Rozwiązanie można przetestować za pomocą poniższego skryptu Lua. Przykładowy ciąg operacji oraz wynik są wpisane do zmiennych **ops** oraz **res**; można je podmienić na dowolne inne.

```

sim.setinput("step", 0)
sim.setinput("nrst", 0)
sim.sleep(100)
assert(sim.getoutput("out"):tointeger() == 0
  and sim.getoutput("cnt"):tointeger() == 0, "Error: reset failed")
sim.setinput("nrst", 1)
sim.sleep(100)
ops = {2, 2, '*', '-', 2, '+', ' '}
res = -2
for k, v in ipairs(ops) do
  if type(v) == "number" then
    sim.setinput("push", 1)
    sim.setinput("d", v)
  elseif type(v) == "string" then
    sim.setinput("push", 0)
    if v == ' ' then
      sim.setinput("op", 0)
    elseif v == '-' then
      sim.setinput("op", 1)
    elseif v == '+' then
      sim.setinput("op", 2)
    elseif v == '*' then
      sim.setinput("op", 3)
    end
  end
end
sim.sleep(50)

```

```
    sim.setinput("step", 1)
    sim.sleep(50)
    sim.setinput("step", 0)
end
sim.sleep(50)
assert(sim.getoutput("out"):toegersigned() == res
       and sim.getoutput("cnt"):tointeger() == 1, "Error: invalid result")
print("OK!")
```