

Logika cyfrowa

Praktyczna lista zadań nr 12

Termin: 8 czerwca 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 obwód sortujący dane zapisane w pamięci metodą sortowania przez wybór. Układ powinien posiadać wydzielony moduł pamięci (dwuportowej, z jednym portem do odczytu i jednym do zapisu, z **synchronicznym** odczytem i zapisem). Sugerowane jest wydzielenie ścieżki sterowania i danych w osobnych modułach. Obwód powinien mieć następujące wejścia i wyjścia:

- `clk` – wejście zegara,
- `nrst` – zanegowane wejście resetu asynchronicznego,
- `start` – 1-bitowe wejście uruchamiające obliczenia,
- `addr` – 3-bitowe wejście adresu interfejsu pamięci,
- `wr` – 1-bitowe wejście zapisu interfejsu pamięci,
- `datain` – 8-bitowe wejście interfejsu pamięci,
- `dataout` – 8-bitowe wyjście interfejsu pamięci,
- `ready` – 1-bitowe wyjście sygnalizujące gotowość układu do rozpoczęcia pracy,

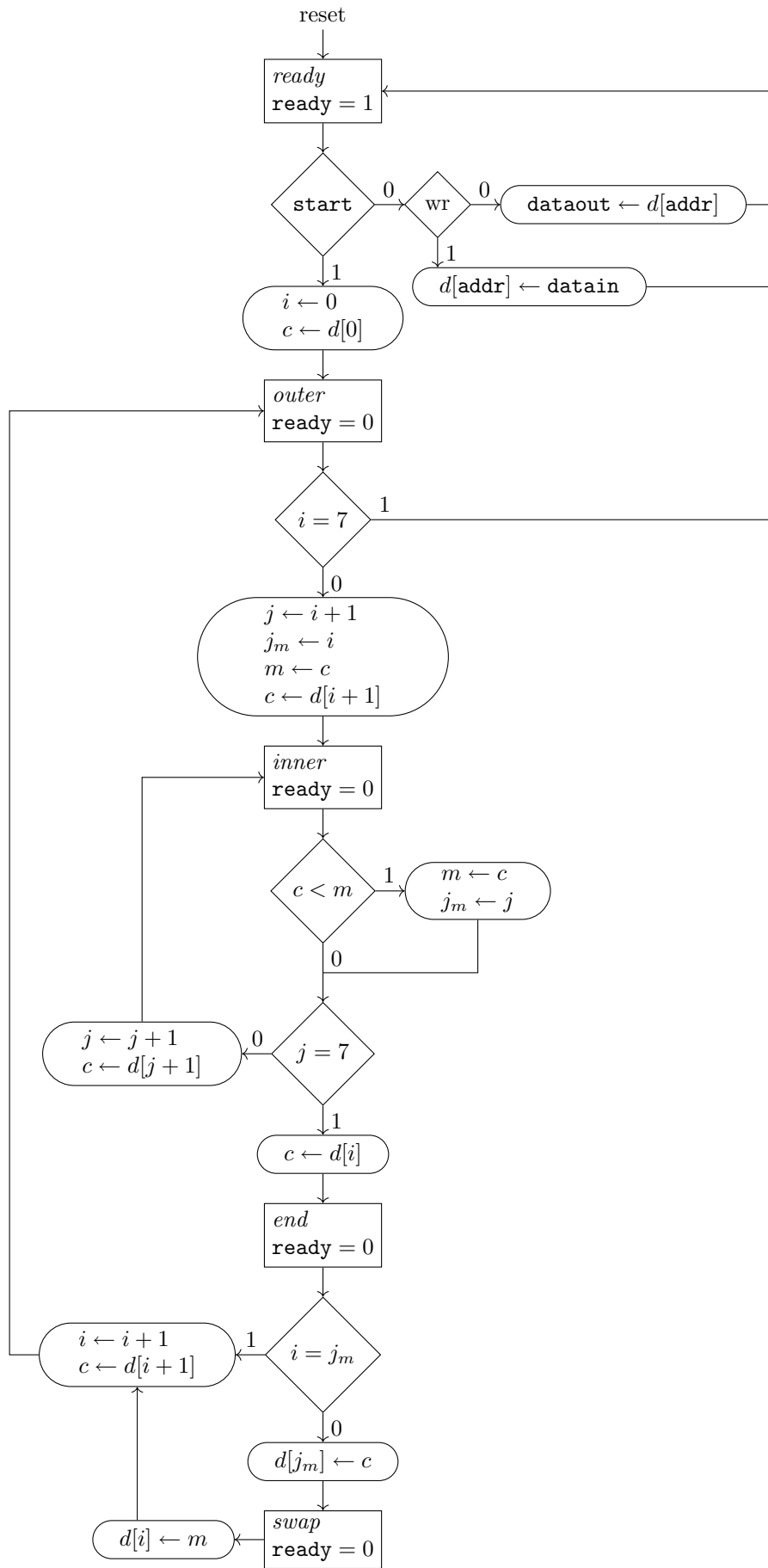
Podobnie jak w poprzednich zadaniach, rozpoczęcie pracy powinno nastąpić, gdy sygnał `start` będzie w stanie wysokim równocześnie ze stanem wysokim `ready`. W kolejnym cyklu stan `ready` powinien zmienić się na niski. Zmiana sygnału `ready` z niskiego na wysoki sygnalizuje ukończenie pracy.

Sygnały zapewniające dostęp do pamięci (`addr`, `datain`, `dataout`, `wr`) powinny być aktywne tylko wtedy, gdy `ready` jest w stanie wysokim; w przeciwnym razie wejścia (`addr`, `datain`, `wr`) powinny być ignorowane, a wyjścia (`dataout`) mogą mieć dowolną wartość. Celem istnienia tych sygnałów jest ładowanie zawartości pamięci przed rozpoczęciem pracy i odczytanie jej po zakończeniu.

Układ pamięci musi być współdzielony między interfejsem zewnętrznym a układem sortującym. Sygnał `ready` mówi, kto kontroluje pamięć: czy interfejs zewnętrzny (`ready` wysoki), czy układ sortujący (`ready` niski). Przełączanie dostępu do pamięci może być realizowane w ścieżce danych lub w osobnym module arbitrującym.

Możliwy sposób działania układu opisuje następujący diagram algorytmiczny. Można ten układ też zaimplementować inaczej – pod warunkiem, że jest zgodny z opisem powyżej.

Do testowania układu można wykorzystać skrypt testujący Lua dołączony do treści zadania. Sugerowane jest wykorzystanie edytora zawartości pamięci DigitalJS, aby obserwować wykonywane dostępy do pamięci w trakcie pracy układu.



```

sim.setinput("wr", 0)
sim.setinput("start", 0)
sim.setinput("nrst", 0)
sim.sleep(100)
sim.setinput("nrst", 1)
sim.sleep(100)
assert(sim.getoutput("ready"):ishigh(), "Reset failed!")
sim.wait(sim.posedge("clk"))
sim.setinput("wr", 1)
arr = {}
for i = 0, 7 do
    arr[i+1] = math.random(0, 255)
    sim.setinput("addr", i)
    sim.setinput("datain", arr[i+1])
    sim.wait(sim.posedge("clk"))
end
sim.setinput("wr", 0)
sim.setinput("start", 1)
sim.wait(sim.posedge("clk"))
sim.setinput("start", 0)
sim.sleep(10)
assert(sim.getoutput("ready"):islow(), "Start failed!")
while sim.getoutput("ready"):islow() do
    sim.wait(sim.posedge("clk"))
    sim.sleep(10)
end
table.sort(arr)
for i = 0, 7 do
    sim.setinput("addr", i)
    sim.wait(sim.posedge("clk"))
    sim.sleep(10)
    assert(sim.getoutput("dataout"):tointeger() == arr[i+1],
           "Sorting failed!")
end
print("OK!")

```