Laboratorium nr 1 — grupa 2, zestaw zadań

Małgorzata Michalczyk, Karol Jabłoński, Piotr Krauze, Krzysztof Mazur

30 marca 2020

1 Bramka XOR (10 punktów)

`zad1(a,b,c)`
Wejścia: 1-bitowa liczba `a`, 1-bitowa liczba `b`
Wyjścia: 1-bitowa liczba `c`

Napisz w języku Verilog moduł o nazwie '`zad1`' o dwóch jednobitowych wejściach a oraz b oraz wyjściu c. Na wyjściu c ma być generowana alternatywa wykluczająca (XOR) wejść a i b.

2 Translator kodu ZM na U2 (40 puntków)

`zad2(a,b)`
Wejścia: 4-bitowa liczba `a`
Wyjścia: 4-bitowa liczba `b`

Napisz w języku Verilog moduł o nazwie '`zad2`', który będzie translatorem kodu binarnego. Wejściem będzie 4-bitowa liczba a w kodzie znak-moduł (ZM), a wyjściem 4-bitowa liczba b w kodzie uzupełnienia do 2 (U2).

3 Poszukiwanie ostatniego zera (90 punktów)

`zad3(a,b)`
Wejścia: 8-bitowa liczba `a`
Wyjścia: 3-bitowa liczba `b`

Zadaniem modułu jest podać jako `b` pozycję ostatniego (najbardziej znaczącego, o największym numerze) zera w `a`. W przypadku braku zer stan wyjścia dowolny. Bity numerowane są następująco:

7	6	5	4	3	2	1	0

4 Potrójna redundancja modularna (40 punktów)

`zad4(a,b,c,y)`
Wejścia: 1-bitowe liczby `a`, `b` i `c`
Wyjścia: 1-bitowa liczba `y`

Moduł ma pełnić funkcję potrójnej redundancji modularnej (ang. Triple Modular Redundancy). W tego typu układach założeniem jest, że na wejścia `a`, `b` i `c` podane są wyniki

działania trzech redundantnych systemów. Zadaniem projektowanego modułu jest podać na wyjściu wynik wskazywany przez większość redundantnych systemów.

Przykład 1: Pierwsze i drugie urządzenie wskazały odpowiednio na wejściach `a`, `b` stan 1, trzecie urządzenie wskazało wynik przeciwny. Na wyjściu modułu powinien pojawić się stan 1.

Przykład 2: Pierwsze i trzecie urządzenie wskazały odpowiednio na wejściach `a`, `c` stan 0, trzecie urządzenie wskazało wynik przeciwny. Na wyjściu modułu powinien pojawić się stan 0.

5 Transmisja odporna na błędy

Celem następnych dwóch zadań jest stworzenie jednokierunkowego systemu transmisyjnego pomiędzy dwoma urządzeniami przesyłającego 4-bitowe dane `d` (4 linie danych) oraz 4 dodatkowe bity parzystości `p`. Linie `d` i `p` uszeregowane są w następujący sposób:

7	6	5	4	3	2	1	0
p3	p2	p1	p0	d3	d2	d1	d0

System ten ma być odporny na awarię jednej dowolnej linii.

Poszczególne bity parzystości są wyznaczane na podstawie odpowiednich par linii danych, zgodnie z poniższych rysunkiem. Bit parzystości przyjmuje wartość 1, jeśli suma jedynek we wskazanej parze bitów jest nieparzysta.

d0	d1
d2	d3

$\Longrightarrow$ p0

$\Longrightarrow$ p1

$\Downarrow$ p2 $\Downarrow$ p3

Zakładając, że następuje maksymalnie jedno przekłamanie i bazując na tym które bity parzystości się nie zgadzają można określić które przekłamanie wystąpiło i je skorygować. Jeśli przekłamany jest bit danych to przekłamane są zawsze 2 bity parzystości. Jeśli przekłamany jest bit parzystości to tylko on jest błędny.

Przykłady

Transmitowana są dane (`d3 d2 d1 d0`) = 0001, po wyznaczeniu czterech bitów parzystości kod wszystkich linii jest następujący:
(`p3 p2 p1 p0 d3 d2 d1 d0`) = 01010001.
Jeśli w wyniku transmisji otrzymamy błędny kod 01010101 to na podstawie bitów danych wyznaczone zostaną bity parzystości 0011. Analizując sposób wyznaczania bitów parzystości łatwo można się domyślić, że musiał być uszkodzony bit danych `d2`, bo jest on kontrolowany przez bity parzystości `p1` i `p2`.

5.1 Enkoder (50 punktów)

`zad5a(a,b)`
Wejścia: 4-bitowa liczba `a`
Wyjścia: 8-bitowa liczba `b`

Napisz moduł o nazwie '`zad5a`' zamieniający 4 bity danych, wejście `a`, na 8-bitowy kod linii, wyjście `b`.

5.2 Dekoder (120 punktów)

`zad5b(a,b)`
Wejścia: 8-bitowa liczba `a`
Wyjścia: 4-bitowa liczba `b`

Napisz moduł o nazwie '`zad5b`' zamieniający 8-bitowy kod linii, wejście `a`, na 4-bity danych `b`. Dekoder ten musi dawać poprawne wyniki dla braku przekłamań na liniach, lub jednego przekłamania.

Wysyłając zadanie '`zad5b`' należy się upewnić, że w tym samym pliku jest też kod rozwiązania '`zad5a`'.