POLITECHNIKA WROCŁAWSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

Układy cyfrowe i systemy wbudowane

Laboratorium 2

Termin zajęć: Czwartek TP, 7:30

Autorzy:

Daria Jeżowska, 252731

Kacper Śleziak, 252703

Prowadzący zajęcia:

dr inż. Jacek Mazurkiewicz

Wstęp

Naszym zadaniem było zbudowanie 3 układów w programie *Xillinx,* następnie pisząc kod w języku VHDL zasymulować jego działanie i uruchomić na prawdziwej płytce. Pierwszym zadaniem było zasymulowanie dowolnej bramki zawierającej dwa wejścia i jedno wyjście. Kolejnym zadaniem była symulacja funkcji logicznej G(w,x,y,z) = Π(0, 2, 3, 4, 6, 7, 9, 11, 12, 13, 15) . Jako ostatnie zadanie zostaliśmy translator kodu NKB (naturalnego kodu binarnego) na kod Aikena.

W zadaniu pierwszym nasz wybór padł na bramkę *AND*, która zwraca wartość 1 w przypadku, gdy na wejściu (w tym wypadku *WeX* oraz *WeY*) będą dwie jedynki.

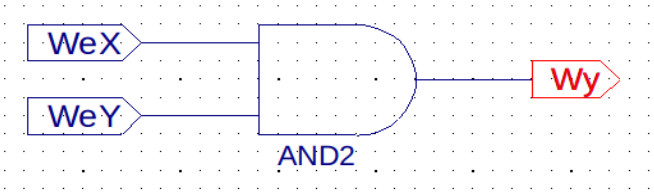
Drugie zadanie polegało na rozpisaniu tabeli prawdy, a następnie na minimalizacji siatką Karnaugh i stworzenia wzoru funkcji na tej podstawie. Do wykonania tego zadania przydatna była wiedza o funkcjach boolowskich oraz algebrze Bool'a.

Trzecie zadanie wymagało znajomości kodu Aikena, zwanego też kodem 2421. Wagi kolejnych bitów wynoszą właśnie 2421, czyli w przypadku liczby 5, nie będzie to 0101 jak w NKB, tylko 1011 (jedna dwójka, zero czwórek, kolejna dwójka i jedna jedynka). Kod ten posiada także oś „antysymetrii", co znaczy poszczególne liczby „uzupełniają się" – np. 3 w tym kodzie to 0011, a liczba 6 – 1100 czy liczby 1 i 8 – 0001 i 1110. W praktyce kod Aikena był wykorzystywany do przesyłania komunikatów cyfrowych.

Zadanie 1

Dowolna bramka – funktor – 2-wejścia, 1-wyjście.

Schemat układu



Schemat układu dla bramki *AND*

Kod w języku VHDL

Kod napisaliśmy w języku VHDL zmienia wartości *we1* co 100ns, natomiast dla *we2* co 200ns. Wynik działania kodu widać w symulacji niżej.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY zad_1_zad_1_sch_tb IS
END zad_1_zad_1_sch_tb;
ARCHITECTURE behavioral OF zad_1_zad_1_sch_tb IS

   COMPONENT zad_1
   PORT( wy1   :  OUT   STD_LOGIC;
         we1   :  IN STD_LOGIC;
         we2   :  IN STD_LOGIC);
   END COMPONENT;

   SIGNAL wy1  :  STD_LOGIC;
   SIGNAL we1  :  STD_LOGIC;
   SIGNAL we2  :  STD_LOGIC;

BEGIN

   UUT: zad_1 PORT MAP(
      wy1 => wy1,
      we1 => we1,
      we2 => we2
   );
   we1 <= '0', '1' after 100ns, '0' after 300ns;
   we2 <= '0', '1' after 200ns, '0' after 400ns;

-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      WAIT; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Przebieg Symulacji

Na poniższym zrzucie ekranu widać zmiany stanów dla *we1, we2* i *wy1*. Zgodnie z tym co jest napisane w kodzie *we1* zmienia swój stan z 0 na 1 po 100ns oraz ponownie na 0 po upływnie 300ns. Natomiast dla *we2* stan został zmieniony na 1 po 200ns i ponownie na 0 po upływie 400ns. Zgodnie z oczekiwaniami, gdy na wejściu mamy dwa stany 1 na naszym wyjściu także pojawia się jedynka.

Testowanie na prawdziwym układzie

Testowanie przebiegło pomyślnie. Trochę mylącym było, że na układzie przy stanie 0 świeciła się dioda LED, zamiast przy stanie 1. Mimo wszystko zgodnie z założeniami - dioda gasła (czyli na wyjściu był stan 1) tylko przy wciśnięciu dwóch guzików.

Zadanie 2

Implementacja funkcji logicznej.
(w procesie minimalizacji podkreślenie oznacza negację wejścia)

Funkcja logiczna i tabela prawdy

$$G(w,x,y,z) = \Pi(0,2,3,4,6,7,9,11,12,13,15)$$

	w	x	y	z	S
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

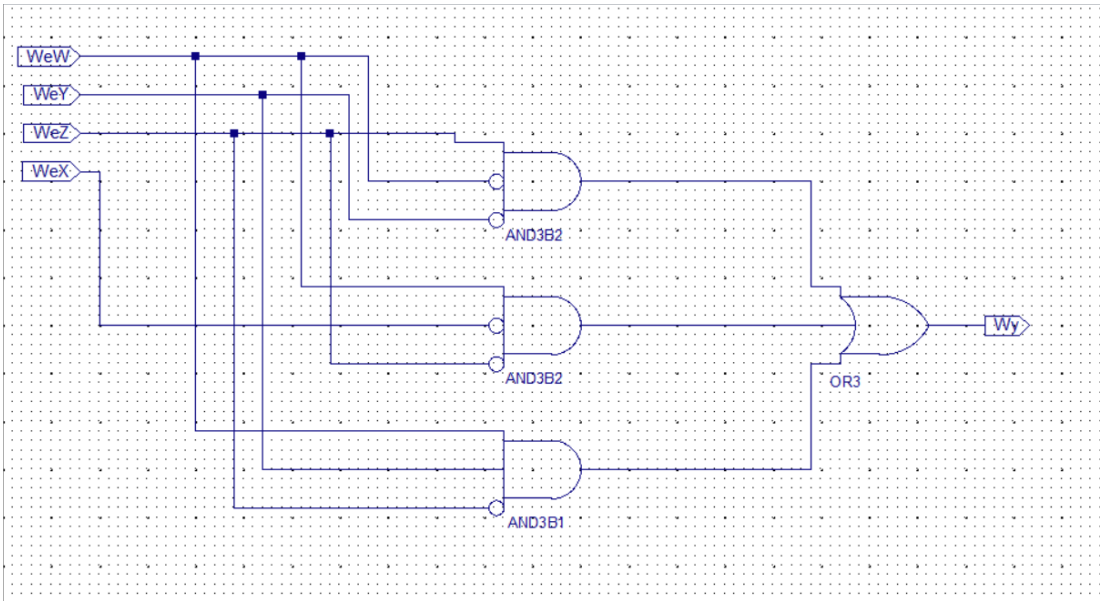
Minimalizacja siatkami Karnaugh

wx\yz	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	0	0	0	1
10	1	0	0	1

$$S = \overline{wy}z + w\overline{xz} + wy\bar{z}$$

Schemat układu

Na schemacie widać 4 wejścia, kolejno *WeW, WeY, WeZ, WeX*. Wejścia reprezentują kolejno zmienne *w,y,z* oraz *x* z powyższej minimalizacji. Zmienna S z tabeli prawdy w układzie zapisana jest jako *Wy*. Do stworzenia schematu wykorzystaliśmy trzy 3-wejściowe bramki *AND* oraz jedną 3-wejściową bramkę *OR.*

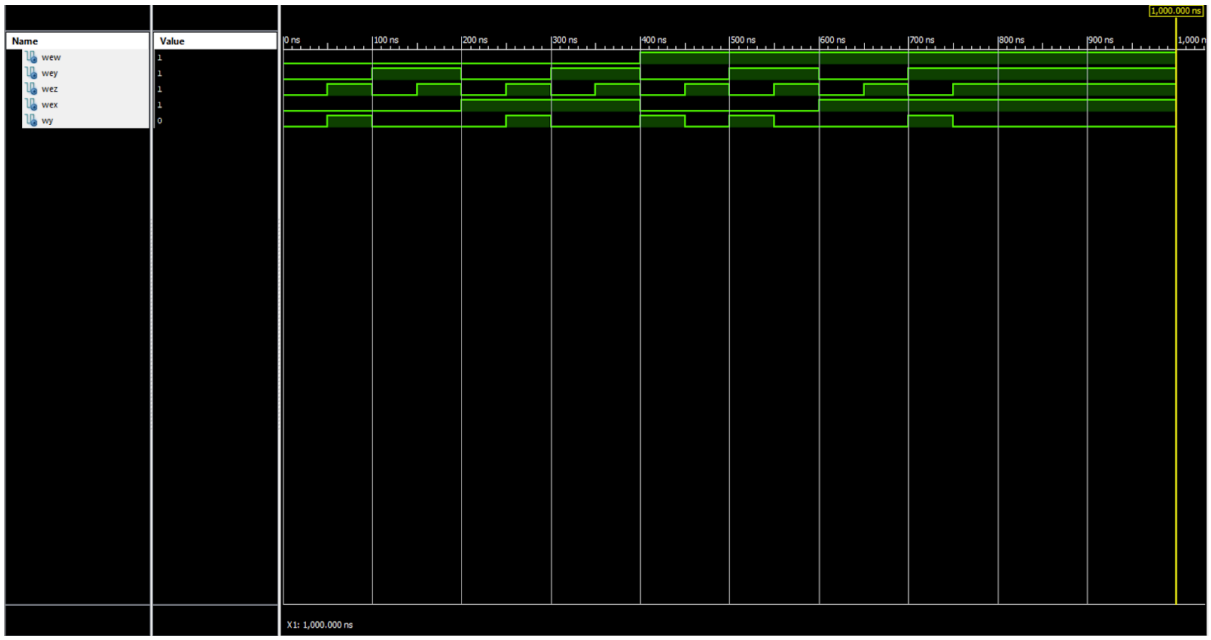


Kod w języku VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY Uklad_Uklad_sch_tb IS
END Uklad_Uklad_sch_tb;
ARCHITECTURE behavioral OF Uklad_Uklad_sch_tb IS

   COMPONENT Uklad
   PORT( WeW   :  IN STD_LOGIC;
         WeY   :  IN STD_LOGIC;
         WeZ   :  IN STD_LOGIC;
         WeX   :  IN STD_LOGIC;
         Wy    :  OUT   STD_LOGIC);
   END COMPONENT;

   SIGNAL WeW  :  STD_LOGIC;
   SIGNAL WeY  :  STD_LOGIC;
   SIGNAL WeZ  :  STD_LOGIC;
   SIGNAL WeX  :  STD_LOGIC;
   SIGNAL Wy   :  STD_LOGIC;

BEGIN

   UUT: Uklad PORT MAP(
      WeW => WeW,
      WeY => WeY,
      WeZ => WeZ,
      WeX => WeX,
      Wy => Wy
   );

WeZ <= '0', '1' after 50ns, '0' after 100ns, '1' after 150ns, '0' after 200ns, '1' after 250ns, '0' after 300ns, '1' after 350ns,
'0' after 400ns, '1' after 450ns, '0' after 500ns, '1' after 550ns, '0' after 600ns, '1' after 650ns, '0' after 700ns, '1' after 750ns;

WeY <= '0', '1' after 100ns, '0' after 200ns, '1' after 300ns, '0' after 400ns, '1' after 500ns, '0' after 600ns, '1' after 700ns;
WeX <= '0', '1' after 200ns, '0' after 400ns, '1' after 600ns;
WeW <= '0' , '1' after 400ns;


END;
```

Przebieg symulacji

Na symulacji widać, że układ został zaprojektowany i zaprogramowany poprawnie. Przykładowo w czasie 100ns po włączeniu symulacji nasze wejścia są ustawione na kolejno 0100 (czyli 2 zapisane w systemie dwójkowym). Na wyjściu otrzymujemy 0 i tym samym potwierdzamy, że symulacja przebiegła zgodnie z założeniami.



Testowanie na prawdziwym układzie

Układ poprawnie reagował na wciśnięcie kolejnych przycisków zgodnie z tabelą prawdy. Dioda przestała świecić (czyli na wyjściu dostaliśmy stan 1), gdy z przycisków były zrobione sekwencje 0001, 0101, 0110, 1000, 1010, 1110 – tak jak w założeniu zadania.

Zadanie 3

Implementacja układu translatora kodu.

Układ translacji kodu:

4-bit kod NKB na 4-bit kod Aikena

Dziesiętnie	NKB				Aikena (2421)			
	a	b	c	d	A	B	C	D
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	1	0	0
7	0	1	1	1	1	1	0	1
8	1	0	0	0	1	1	1	0
9	1	0	0	1	1	1	1	1
10	1	0	1	0	-	-	-	-
11	1	0	1	1	-	-	-	-
12	1	1	0	0	-	-	-	-
13	1	1	0	1	-	-	-	-
14	1	1	1	0	-	-	-	-
15	1	1	1	1	-	-	-	-

Minimalizacja siatkami Karnaugh

A

ab\cd	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	-	-	-	-
10	1	1	-	-

$A = a + bd + bc = b(d+c) + a$

B

ab\cd	00	01	11	10
00	0	0	0	0
01	1	0	1	1
11	-	-	-	-
10	1	1	-	-

$B = b\bar{d} + a + bc = a + b(c + \bar{d})$

C

ab\cd	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	-	-	-	-
10	1	1	-	-

$C = a + b\bar{c}d + \bar{a}\bar{b}c$

D

ab\cd	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	-	-	-	-
10	0	1	-	-

$D = d$

Po minimalizacji:

$A = a + bd + bc$

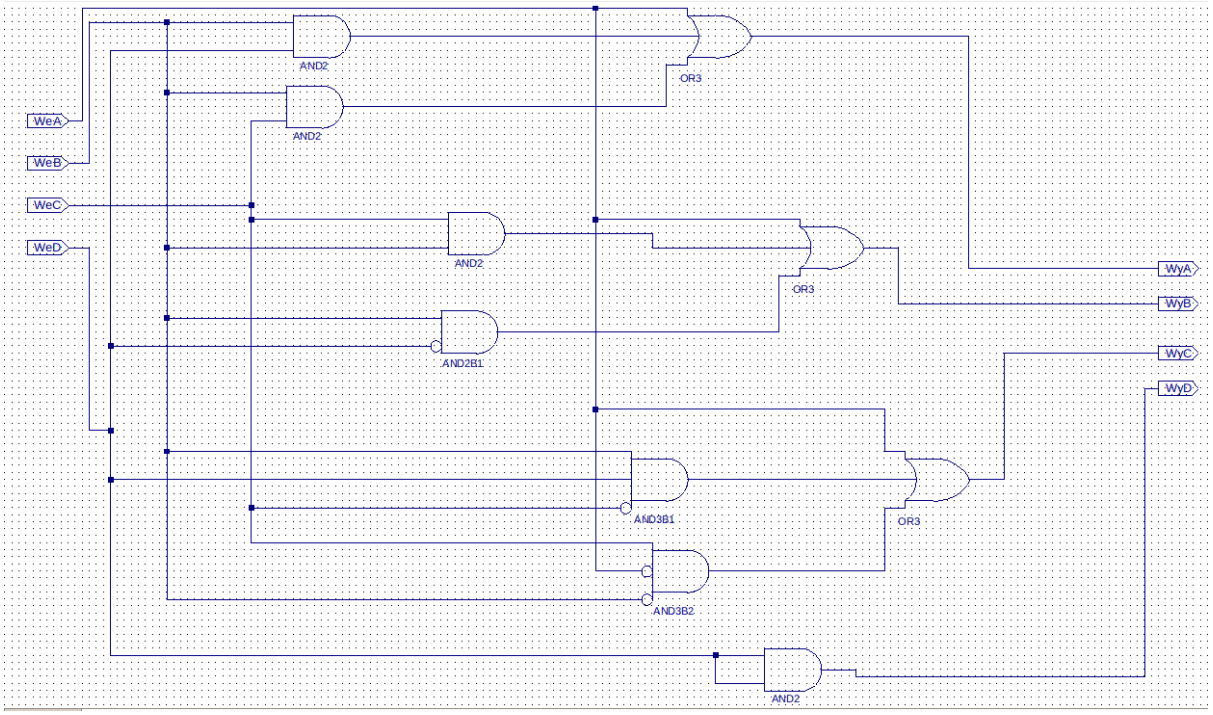
$B = b\bar{d} + a + bc$

$C = a + \ b\bar{c}d + \overline{\overline{a}bc}$

$D = d$

Schemat układu

Na schemacie są cztery wejścia – *WeA, WeB, WeC, WeD* oraz cztery wyjścia - *WyA, WyB, WyC, WyD.* Na wejściach podajemy naturalny kod binarny, a na wyjściu otrzymujemy NKB przetłumaczony na kod Aikena. Do stworzenia układu użyliśmy bramek *AND* oraz *OR*, zgodnie ze wzorami.



Kod w języku VHDL

Aby poprawnie zasymulować działanie translatora kodu trzeba odpowiednio zmieniać stany. Zgodnie z tabelą prawdy na wejściu *WeA* zmieniamy stan co 50ns, na wejściu *WeB* co 100ns, na *WeC* zmieniamy stan co 200ns, a na *WeD* zmieniamy stan raz – po 400ns. W ten sposób możemy przetestować czy translator faktycznie działa.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY schem_1_schem_1_sch_tb IS
END schem_1_schem_1_sch_tb;
ARCHITECTURE behavioral OF schem_1_schem_1_sch_tb IS

   COMPONENT schem_1
   PORT( WyC   :  OUT   STD_LOGIC;
         WyB   :  OUT   STD_LOGIC;
         WyA   :  OUT   STD_LOGIC;
         WeA   :  IN STD_LOGIC;
         WeB   :  IN STD_LOGIC;
         WeC   :  IN STD_LOGIC;
         WyD   :  OUT   STD_LOGIC;
         WeD   :  IN STD_LOGIC);
   END COMPONENT;

   SIGNAL WyC  :  STD_LOGIC;
   SIGNAL WyB  :  STD_LOGIC;
   SIGNAL WyA  :  STD_LOGIC;
   SIGNAL WeA  :  STD_LOGIC;
   SIGNAL WeB  :  STD_LOGIC;
   SIGNAL WeC  :  STD_LOGIC;
   SIGNAL WyD  :  STD_LOGIC;
   SIGNAL WeD  :  STD_LOGIC;

BEGIN

   UUT: schem_1 PORT MAP(
      WyC => WyC,
      WyB => WyB,
      WyA => WyA,
      WeA => WeA,
      WeB => WeB,
      WeC => WeC,
      WyD => WyD,
      WeD => WeD
   );

   WeD <= '0', '1' after 50ns, '0' after 100ns, '1' after 150ns, '0' after 200ns, '1' after 250ns, '0' after 300ns, '1' after 350ns,
          '0' after 400ns, '1' after 450ns, '0' after 500ns, '1' after 550ns, '0' after 600ns, '1' after 650ns, '0' after 700ns, '1' after 750ns;
   WeC <= '0', '1' after 100ns, '0' after 200ns, '1' after 300ns, '0' after 400ns, '1' after 500ns, '0' after 600ns, '1' after 700ns;
   WeB <= '0', '1' after 200ns, '0' after 400ns, '1' after 600ns;
   WeA <= '0', '1' after 400ns;


-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      WAIT; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Przebieg symulacji

Na symulacji widać, że układ został zaprojektowany i zaprogramowany poprawnie. Przykładowo – w 400ns nasze wejście są odpowiednio: 1000 (czyli 8 dziesiętna), a dla wyjścia mamy – 1110, co znaczy, że symulacja przebiegła poprawnie. Kod Aikena działa dla cyfr 0-9, więc to, co się dzieje po 500ns nie jest w tym przypadku istotne.

Wnioski

Wykonanie zadań laboratoryjnych odbyło się bez większych komplikacji. Wykorzystaliśmy swoją wiedzę z III semestru z *Logiki układów cyfrowych*. Po drodze napotkaliśmy problem techniczny wynikający z pierwszorazowego używania środowiska *Xilinx*. Nie potrafiliśmy stworzyć pliku *.jed*, problem rozwiązało zmienienie zakładki z *Simulation* na *Implementation* i znalezienie odpowiedniej funkcji. Rozwiązanie tego problemu zajęło nam bardzo dużo czasu przez co nie zdążyliśmy wykonać do końca zadania 3. Schemat i symulacja dla zadania 3 powstały na zajęciach, jednakże nie zdążyliśmy przetestować na realnym układzie. Zadania 1 i 2 w pełni powstały i zostały przetestowane w trakcie zajęć.