Politechnika Wrocławska
Wydział Elektroniki

Projekt Uklady Cyfrowe i Systemy Wbudowane 2

"Kalkulator bcd - PS2"

Autorzy:
Leszek Ostek
Indeks: 249439
E-mail: 249439@student.pwr.edu.pl

Grzegorz Kuchta
Indeks: 248861
E-mail: 248861@student.pwr.edu.pl

Prowadzący zajęcia:
Dr inż. Jaroslaw Sugier

Ocena pracy:

Spis treści

Spis rysunków

Spis tabel

1 Wstęp

1.1 Cel i zakres projektu

Temat: "Kalkulator PS2 w kodzie BCD".
Cel projektu: Implementacja sterownika PS2 oraz modulow sumatora i subtraktora w kodzie BCD. Wspomniane moduły mają być składowymi projektu kalkulatora, który ma być dostępny przy pomocy klawiatury PS2.

1.2 Opis używanych interfejsów, protokołów oraz algorytmów

PS2 (port komunikacyjny) - Do komunikacji z zaprojektowanym układem użyto interfejsu szeregowego PS2 [5]. Użytkownik musi posiadać klawiaturę PS2, która komunikuje się z układem przy pomocy *PS2 Scan Codes* [4].

Sumator i subtraktor - Elementy działają na liczbach 4 bitowych w kodzie BCD. Sumator kodu BCD [2], pobiera na wejściu 2 magistrale 4 bitowe symbolizujące liczby kodu BCD (zakres 0-9) oraz pojedynczy bit symbolizujący przeniesienie z sumatora bitu o 1 młodszego. Wyjściami tego modułu jest magistrala 4 bitowa odwzorowująca sumę na danej pozycji oraz 1 bitowe przeniesienie na starsza o 1 pozycję. Subtraktor [3] działa analogicznie, w tym że zamiast przeniesienia wejściowego pobiera 1 bitową pożyczkę na tej pozycji oraz na wyjściu zamiast przeniesienia na starszą pozycję otrzymujemy pożyczkę na starszej pozycji. Z racji, że jest to subtraktor kodu BCD nie obsługuje on liczb ujemnych, więc korzystanie z subtraktora może się odbywać tylko w przypadku odejmowania liczby mniejszej, bądź równej odejmowanej.

1.3 Krótki opis sprzętu i użytych narzędzi

Środowisko - ISE Design Suite 14.7 [6]. Środowisko pozwala na tworzenie i testowanie układów w języku VHDL.

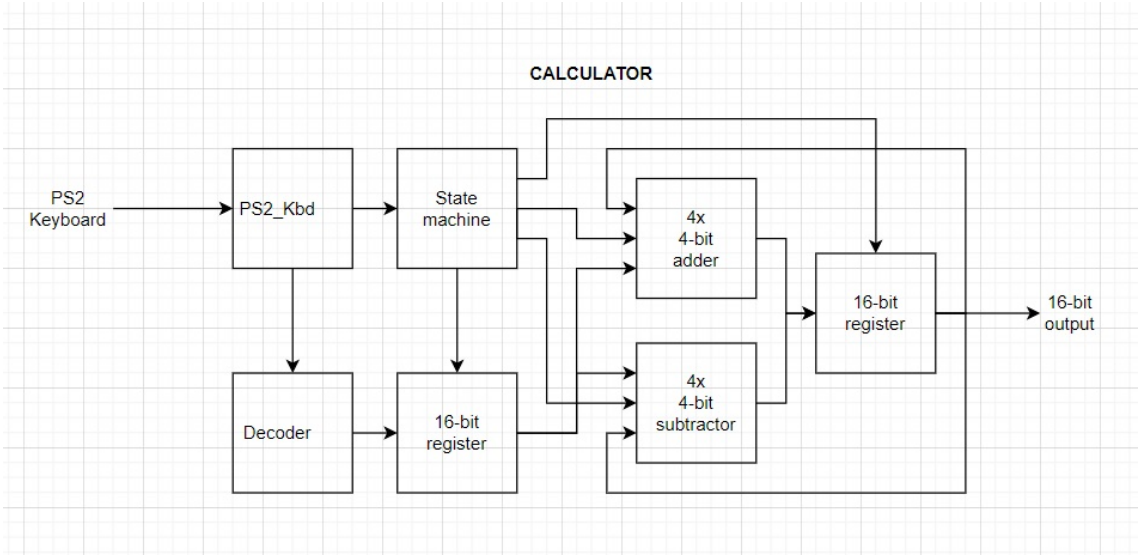
Moduł - Spartan 3E [1]. Płytka pozwalaja na obsługe układów FPGA.

Klawiatura PS2 - Aby móc komunikować się z układem należy użyć klawiatury z portem PS2.

2 Przedstawienie układu

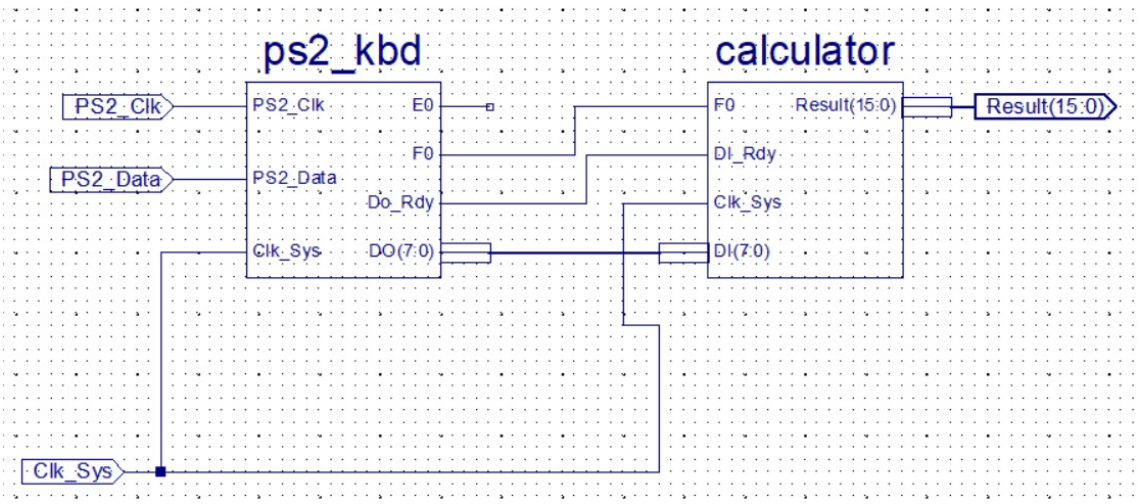
2.1 Struktura ogólna

Rysunek 1 przedstawia uproszczony schemat kalkulatora. Przetwarza on informacje w sposób akumulacyjny, czyli wynikiem aktualnym jest $Result = Result \pm Value$. Dekoder tłumaczy znak z modułu *PS2_Kbd* na 4-bitową liczbe binarną, która trafia do 16-bitowego rejestru przesuwnego. Wartość z rejestru przesuwnego jest dodawana, bądź odejmowana od aktualnego wyniku. Całością zarządza maszyna stanów kalkulatora.



Rysunek 1: Ogólny schemat kalkulatora

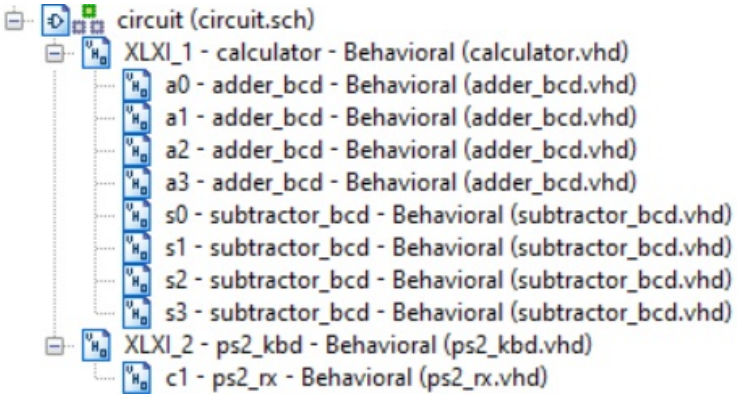
2.2 Schemat szczytowy



Rysunek 2: Moduł szczytowy

Rysunek 2 przedstawia szczytowy schemat projektu. Jest to moduł typu *Schematic*, a w projekcie plik circuit.sch . Wybrano typ *Schematic*, ponieważ bardzo dobrze sprawdza się w łączeniu kluczowych komponentów systemu, gdyż bardzo dobrze widać na nim poszczególne elementy.

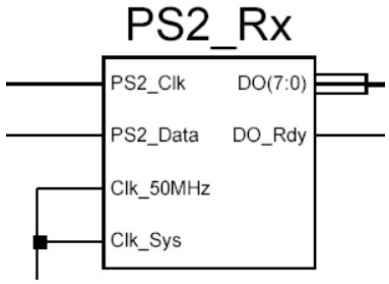
2.3 Hierarchia modułów



Rysunek 3: Drzewo projektu

3 Moduly projektu i ich funkcje

3.1 PS2_rx



Rysunek 4: Moduł PS2_Rx

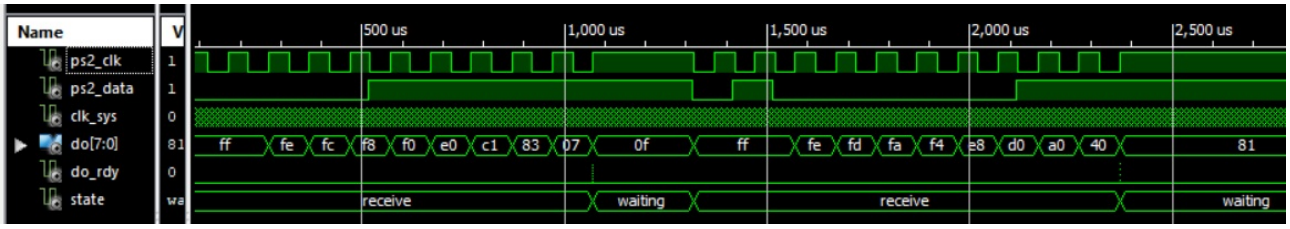
PS2_Rx [7] to moduł znajdujący się wewnątrz odbiornika PS2_Kbd. Jego zadaniem jest przerobić sygnał szeregowy *PS2_Data*, sygnalizowany sygnałem *PS2_Clk*, na 8-bitowy sygnał równoległy, sygnalizowany sygnałem *DO_Rdy*.

3.1.1 kod i omowienie

```
-- Sprawdzamy parzystosc otrzymanego ciagu bitowego,
-- aby upewnic sie czy sygnal jest poprawny.
parity_check <= not(data_reg(8) xor data_reg(7) xor data_reg(6)
                    xor data_reg(5) xor data_reg(4) xor data_reg(3)
                      xor data_reg(2) xor data_reg(1));
```

```
-- Maszyna stanow
process(Clk_Sys, state, data_reg, parity_check, clk_reg)
begin
next_state <= state;
case state is
--  Clk_reg to 2 bitowy rejestr do ktorego jest zapisywany
--  przesuwnie sygnal z ps2_clk.

--  Warunek w operacji if sprawdza zbocze rosnace
when waiting =>
        if clk_reg(0) = '0' and clk_reg(1) = '1' then
                next_state <= reset;
        end if;

-- Po zresetowaniu zatrzynamy obierac
when reset =>
        next_state <= receive;

-- Operacja if sprawdza czy w rejestrze przesuwnym
-- wystapila pelna ramka, jesli tak to przechodzimy
-- do sprawdzenia bitu parzystosci
when receive =>
        if data_reg(10) = '0' and data_reg(0) = '1' then
                next_state <= test;
        end if;

-- Bit 9 nadany w sygnale szeregowym to bit parzystosci
when test =>
        if data_reg(9) = parity_check then
                next_state <= send;
        else
                next_state <= waiting;
        end if;

when send =>
        next_state <= waiting;
end case;
end process;

-- Akcje wykonywane w zaleznosci od stanow
DO_Rdy <= '1' when state = send
   else '0';
DO <= data_reg(9 downto 2);
```

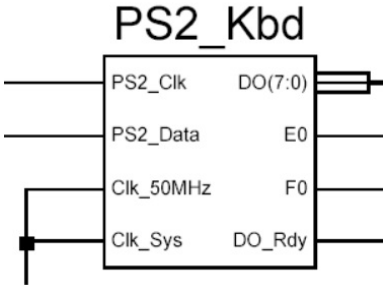
3.1.2 testy



Rysunek 5: PS2_Rx symulacja behawioralna

Celem testu było nadanie dwóch bajtów "F0" oraz "81". Jak widać na *Rysunek 5*, symulacja przebiegła pomyślnie. Bardzo krótkie piki *Do_Rdy*, widać blisko 1100 us i 2400 us.

3.2 PS2_kbd



Rysunek 6: Moduł PS2_Kbd

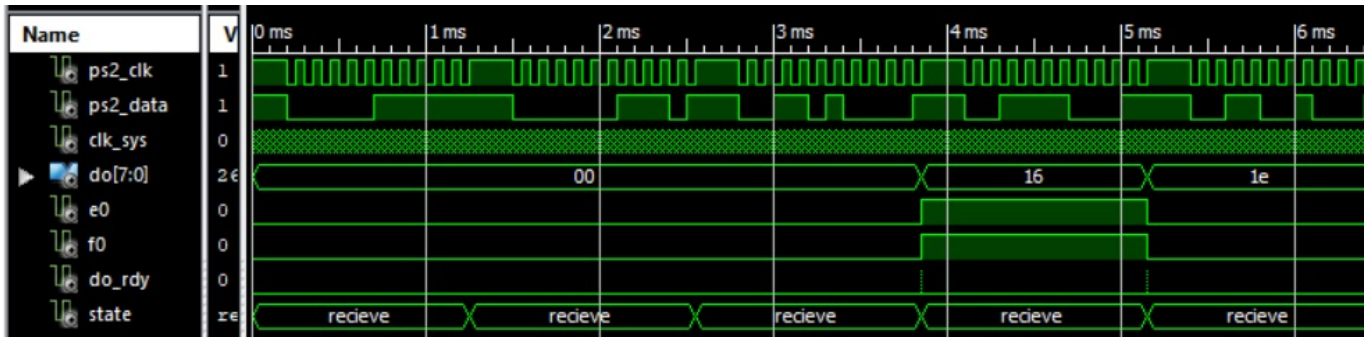
PS2_Kbd [7] to moduł w którego wnętrzu znajduje sie odbiornik PS2_Rx. Jego zadaniem jest wykryce bajtu "F0" lub "E0", które może nadać PS2_Rx i w zależności od tego nadanie 1 bitowej flagi (F0 lub E0). Sygnały *DO* i *DO_Rdy* są propagowane gdy pojawi się inny kod niż F0 i E0.

3.2.1 kod i omowienie

```
-- Case maszny stanow
case state is
        -- Ponizszy if ma na celu wykryc zbocze opadajace
        -- w sygnale DO_Rdy, oraz w zaleznosci od wejscia DO
        -- ustawic odpowiednia flage
        when recieve =>
        if DO_Rdy_reg(0) = '1' and DO_Rdy_reg(1) = '0' then
                if DO_rx = "00001111" then
                        next_state <= set_F0;
                elsif DO_rx = "00000111" then
```

```
                        next_state <= set_E0;
                else
                        next_state <= send;
                end if;
        end if;

        when set_F0 =>
                next_state <= recieve;

        when set_E0 =>
                next_state <= recieve;

        when send=>
                next_state <= reset;

        when reset=>
                next_state <= recieve;
end case;

-- Poniewaz DO_rx jest odwrocone, nalezy jeszcze
-- dopasowac sie do wejscia kalkulatora odwracajac
-- kolejnosc bitow w wyjsciu PS2_Kbd
DO_kbd <= DO_rx(0) & DO_rx(1) & DO_rx(2) & DO_rx(3) &
DO_rx(4) & DO_rx(5) & DO_rx(6) & DO_rx(7);
```

3.2.2 testy

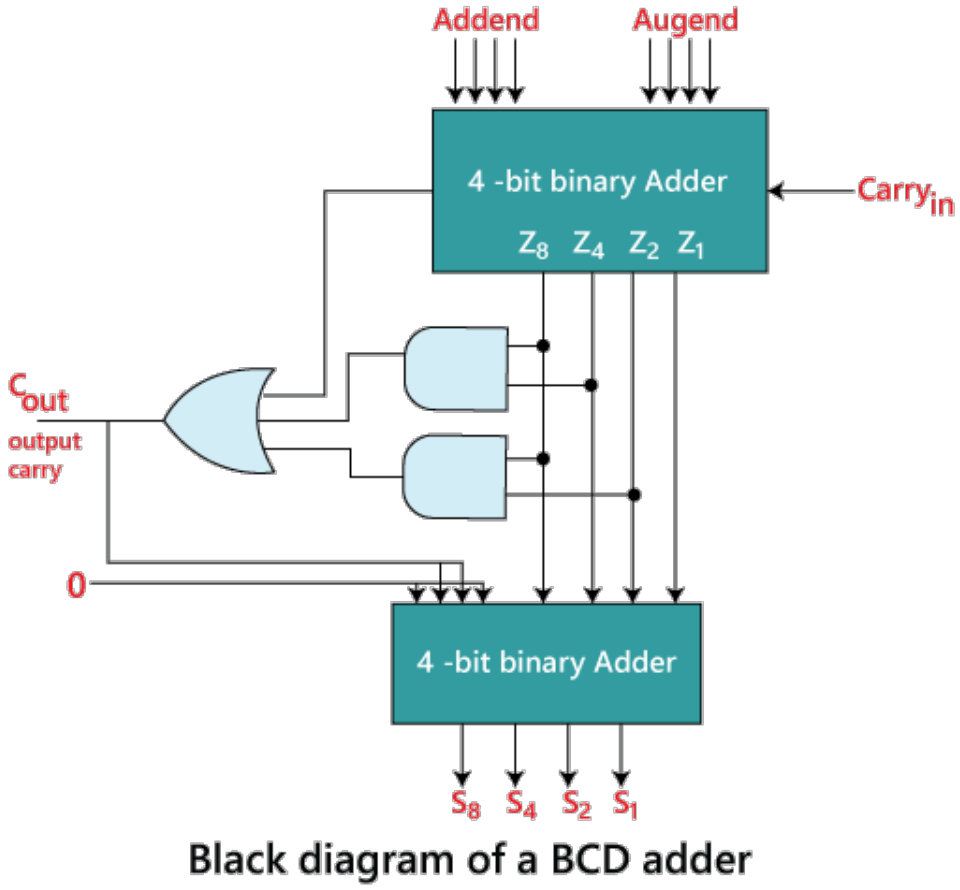


Rysunek 7: PS2_Kbd symulacja behawioralna

Celem testu było nadanie bajtów "16" z flagą "F0" i "E0", a następnie nadanie czystego bajtu "1e".

3.3 Sumator

Schemat sumatora liczb 4 bitowych w kodzie BCD prezentuje sie w sposob nastepujacy :



Rysunek 8: Schemat sumatora BCD

Zaprezentowany powyzej schemat przeniesiono do modulu adderbcd.vhd, ktory pozwala tworzyc uklady dodajace liczby bitowe dowolnej ilosci.

3.3.1 kod i omowienie

Lista wejść :

- 4 bitowa liczba "a" (zakres 0-9)
- 4 bitowa liczba "b" (zakres 0-9)
- 1 bitowe przeniesienie (zakres 0-1)

Lista wyjść :

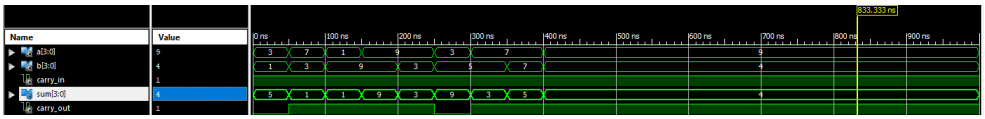
- 4 bitowa liczba symbolizująca sumę na danej pozycji (zakres 0-9)
- 1 bitowe przeniesienie na bicie o 1 starszym (zakres 0-1)

Kod modułu sumatora został zaprezentowany poniżej :

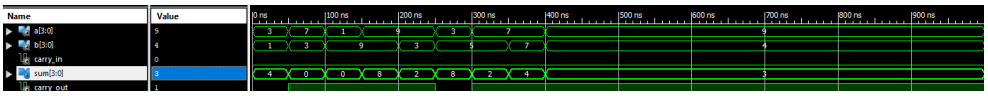
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  use IEEE.NUMERIC_STD.ALL;
5
6  entity adder_bcd is
7      Port (
8             A : in  STD_LOGIC_VECTOR(3 downto 0);
9            B : in  STD_LOGIC_VECTOR(3 downto 0);
10           CARRY_IN : in  STD_LOGIC;
11           SUM : out  STD_LOGIC_VECTOR(3 downto 0);
12           CARRY_OUT : out  STD_LOGIC
13         );
14 end adder_bcd;
15
16 architecture Behavioral of adder_bcd is
17
18 begin
19
20 process(A,B)
21
22 variable sum_temp : unsigned(4 downto 0);
23 variable temp_a : unsigned(3 downto 0);
24 variable temp_b : unsigned(3 downto 0);
25
26 begin
27    temp_a := unsigned(A);
28    temp_b := unsigned(B);
29     sum_temp := ('0' & temp_a) + ('0' & temp_b) + ("0000" & CARRY_IN);
30     if(sum_temp > 9) then
31         CARRY_OUT <= '1';
32         SUM <= std_logic_vector(resize((sum_temp + "00110"),4));
33     else
34         CARRY_OUT <= '0';
35         SUM <= std_logic_vector(sum_temp(3 downto 0));
36     end if;
37 end process;
38
39 end Behavioral;
```

Rysunek 9: Kod modułu sumatora

3.3.2 Testy



Rysunek 10: Test dla przeniesienia wejściowego równego 1



Rysunek 11: Test dla przeniesienia wejściowego równego 0

3.4 Subtraktor

Subtraktor kodu BCD, został zaprojektowany strickte z myślą o implementacji w języku vhdl oraz nie posiada schematu w przeciwieństwie do sumatora kodu BCD.

3.4.1 kod i omowienie

Lista wejść :

- 4 bitowa liczba "a" (zakres 0-9)
- 4 bitowa liczba "b" (zakres 0-9)
- 1 bitowa pożyczka (zakres 0-1)

Lista wyjść :

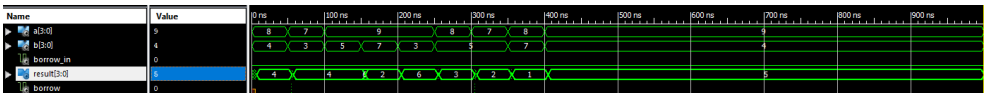
- 4 bitowa liczba symbolizująca różnicę na danej pozycji (zakres 0-9)
- 1 bitowa pożyczka na bicie o 1 starszym (zakres 0-1)

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  use IEEE.NUMERIC_STD.ALL;
5
6  entity subtractor_bcd is
7      Port (
8              A : in  STD_LOGIC_VECTOR (3 downto 0):= X"0";
9             B : in  STD_LOGIC_VECTOR (3 downto 0):= X"0";
10            BORROW_IN: in  STD_LOGIC;
11            RESULT : out  STD_LOGIC_VECTOR (3 downto 0):= X"0";
12            BORROW : out  STD_LOGIC:= '0'
13            );
14 end subtractor_bcd;
15
16 architecture Behavioral of subtractor_bcd is
17
18 begin
19 process(A,B,BORROW_IN)
20
21 variable sum_temp : unsigned(4 downto 0);
22 variable temp_a : unsigned(4 downto 0);
23 variable temp_a_borrow : unsigned(4 downto 0);
24 variable temp_b : unsigned(4 downto 0);
25
26 begin
27     temp_a := unsigned('0' & A);
28     temp_b := unsigned('0' & B);
29     temp_a_borrow := unsigned('0' & A) + 10;
30     if(temp_a < temp_b ) then
31          sum_temp := temp_a_borrow - temp_b - ("0000" & BORROW_IN);
32         BORROW <= '1';
33     else
34          sum_temp := temp_a - temp_b - ("0000" & BORROW_IN);
35         BORROW <= '0';
36     end if;
37     RESULT <= std_logic_vector(sum_temp(3 downto 0));
38
39 end process;
40
41 end Behavioral;
```

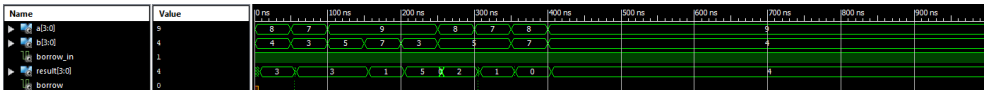
Rysunek 12: Kod modułu subtraktora

Kod modułu subtraktora został zaprezentowany powyżej.

3.4.2 testy

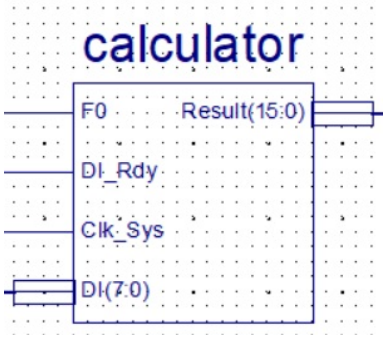


Rysunek 13: Test dla pożyczki wejściowej równej 0



Rysunek 14: Test dla pożyczki wejściowej równej 1

3.5 Kalkulator



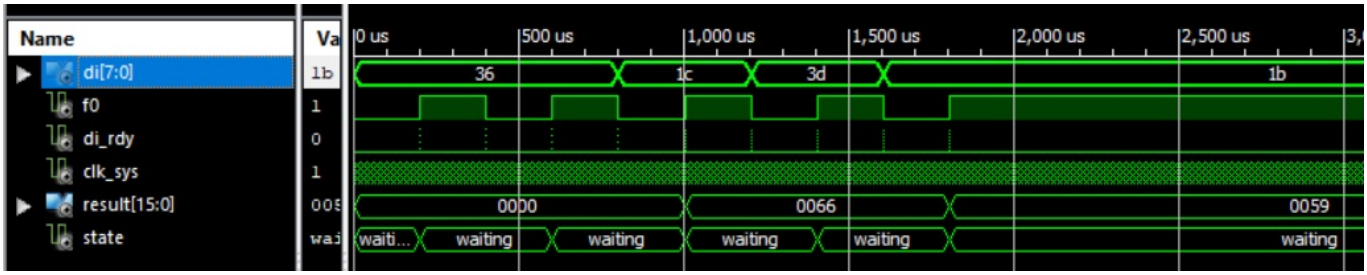
Rysunek 15: Moduł kalkulatora

3.5.1 kod i omowienie

```
-- Dekoder, jego zadaniem jest przetlumaczyc kod PS2 na liczbe binarna
DECODE : process(DI)
begin
    case DI is
                    when X"45" => number <= X"0";
                    when X"16" => number <= X"1";
                    when X"1E" => number <= X"2";
                    when X"26" => number <= X"3";
                    when X"25" => number <= X"4";
                    when X"2E" => number <= X"5";
                    when X"36" => number <= X"6";
                    when X"3D" => number <= X"7";
                    when X"3E" => number <= X"8";
                    when X"46" => number <= X"9";
                    when others => number <= X"F";
```

```
        end case;
end process;

ACTIONS : process(Clk_Sys)
begin
if(rising_edge(Clk_Sys)) then
case state is

        when waiting =>
        -- Przesuwny rejestr, dopisujacy aktualna liczbe
        when input =>
        if number < X"A" then
                current_value(15 downto 4) <= current_value(11 downto 0);
                current_value(3 downto 0) <= number;
        end if;

        when add =>
        final_value <= sum;
        current_value <= X"0000";

        when sub =>
        if current_value <= final_value then
                final_value <= subtract;
        end if;
        current_value <= X"0000";

        when reset =>
        final_value <= X"0000";
        current_value <= X"0000";

        end case;
end if;
end process;
```

3.5.2 testy

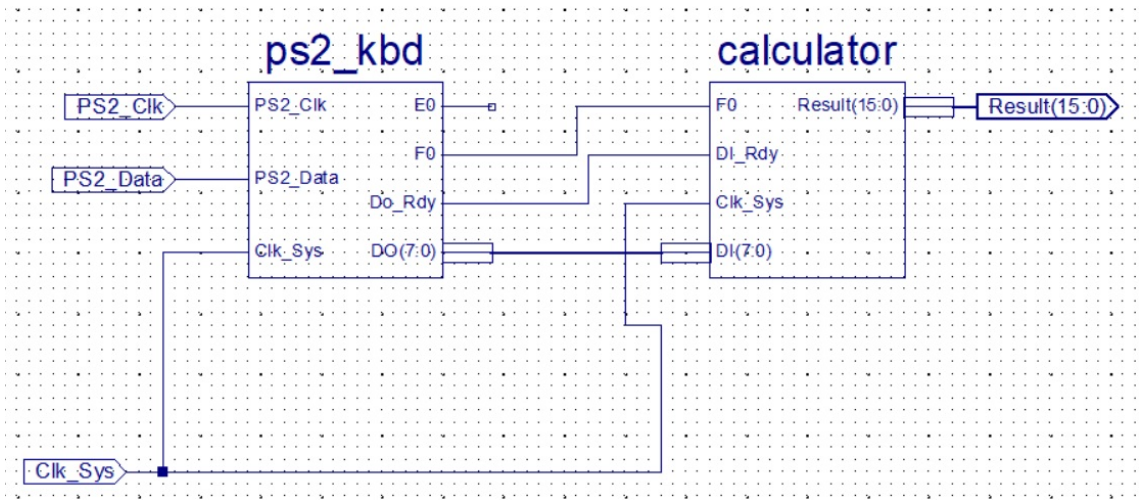


Rysunek 16: Symulacja behawioralna kalkulatora

Celem testu było dodanie do zera liczby 66, oraz odjęcie od niej 7. Wynik testu jest poprawny, należy zwrócić uwage że kalkulator reaguje na puszczenie klawisza. Kod "1c" oznacza dodawanie, kod "1b" oznacza odejmowanie.

4 Implementacja systemu

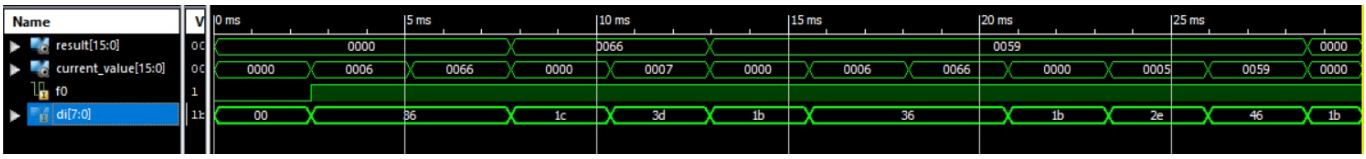
4.1 Połączenie modułów



Rysunek 17: circuit.sch

System został połączony przy pomocy metody *Schematic*.

4.2 Test systemu



Rysunek 18: Test systemu

Celem testu było dodanie liczby 66 do wyniku, następnie odjęcie 7. Ponieważ kalkulator jest zablokowany na liczby ujemne, sprawdzamy odjęcie 66 od 59. Prawidłowo system nie powinien wykonać tej operacji (i tak sie stało). Na końcu odejmujemy 59 od 59 i otrzymujemy 0.

4.3 Implementacja

circuit Project Status (06/02/2021 - 00:02:39)			
Project File:	Calculator.xise	**Parser Errors:**	
Module Name:	circuit	**Implementation State:**	Placed and Routed
Target Device:	xc3s100e-5vq100	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	5 Warnings (5 new)
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Device Utilization Summary [-]				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	69	1,920	3%	
Number of 4 input LUTs	222	1,920	11%	
Number of occupied Slices	135	960	14%	
Number of Slices containing only related logic	135	135	100%	
Number of Slices containing unrelated logic	0	135	0%	
Total Number of 4 input LUTs	226	1,920	11%	
Number used as logic	222			
Number used as a route-thru	4			
Number of bonded IOBs	19	66	28%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	3.54			

Rysunek 19: Wynik implementacji

Ostrzeżenia dotyczą jedynie, niepodłączonych do niczego wyjść/wejść różnych modułów. Ponieważ nie wpływa to na pracę układu, ostrzeżenia można zignorować.

5 Posdumowanie

Finalnie udało się zaprojektować sterowniki dla interfejsu PS2 oraz prosty kalkulator obsługujący dodawanie oraz odejmowanie liczb dodatnich. Celem rozbudowania projektu można by dodać moduły wykonujące operacje mnożenia, dzielenia, zmiana kodowania liczb np. uzupełnieniowy dwójkowy (możliwość obsługi liczb dodatnich oraz ujemnych). Definitywnie poprawie należy poddać moduł ps2_rx, który zwraca wynik w złej kolejności bitów. Należy odwrócić kolejność wyjściowych bitów, i zlikwidować odwracanie wejścia w ps2_kbd.

Literatura

[1] https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[2] https://www.geeksforgeeks.org/bcd-adder-in-digital-logic/

[3] https://www.electrical4u.com/bcd-or-binary-coded-decimal-bcd-conversion-addition-subtraction/

[4] https://techdocs.altium.com/display/FPGA/PS2+Keyboard+Scan+Codes

[5] https://en.wikipedia.org/wiki/PS/2_port

[6] https://www.xilinx.com/products/design-tools/ise-design-suite.html

[7] http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/