Programowalne Układy Cyfrowe – Ćwiczenie 5

Cele:

- zapoznanie z układami arytmetycznymi,
- realizacja mnożenia szerokiego za pomocą modułów o węższej szerokości bitowej.

Przebieg ćwiczenia:

1. Uruchomić program Active-HDL, stworzyć nowy projekt.
2. W postaci dwóch plików źródłowych wprowadzić następujący program:

Plik testbench.v

```
module mytestbenchmodule();

reg CLK;
initial CLK  <= 0;
always #50  CLK <= ~CLK;

reg RST;
initial
begin
    RST <= 0;
    RST <= #100 1;
    RST <= #500 0;
end

reg [15:0] i_dat_a;
reg [15:0] i_dat_b;
reg i_stb;
reg o_ack;

initial
begin
    o_ack <= 0;
    i_stb <= 0;
    #1000;
    i_dat_a <= 15;
    i_dat_b <= 22;
    i_stb <= 1;
    #100;
    i_stb <= 0;
    #4050;
    o_ack <= 1;

end

adder
#(
.A_WIDTH(16),
.B_WIDTH(16)
)
adder1
(
.CLK(CLK),
.RST(RST),

.I_DAT_A(i_dat_a),
.I_DAT_B(i_dat_b),
.I_STB(i_stb),
.I_ACK(),

.O_DAT(),
.O_STB(),
.O_ACK(o_ack)
);

endmodule
```

Plik adder.v

```
module adder
#(
parameter A_WIDTH = 32,
parameter B_WIDTH = 32
)
(
    input wire RST,
    input wire CLK,

    input  wire I_STB,
    output wire I_ACK,
    input  wire [A_WIDTH-1:0] I_DAT_A,
    input  wire [B_WIDTH-1:0] I_DAT_B,

    output reg  O_STB,
    output reg  [(A_WIDTH>B_WIDTH?A_WIDTH:B_WIDTH) : 0] O_DAT,
    input  wire O_ACK
);

assign I_ACK = I_STB & ~O_STB;

always @(posedge CLK or posedge RST)
if (RST) O_DAT <= 0; else if (I_ACK) O_DAT <= I_DAT_A+I_DAT_B;

always @(posedge CLK or posedge RST)
if (RST) O_STB <= 0; else if (O_ACK) O_STB <= 0; else if (I_ACK) O_STB <= 1;

endmodule
```

3. Co oznaczają parametry A_WIDTH i B_WIDTH? Jakie zalety ma ich zdefiniowanie?
4. Zastanowić się, z czego wynika zapis:

```
output reg  [(A_WIDTH>B_WIDTH?A_WIDTH:B_WIDTH) : 0] O_DAT,
```

5. Skompilować i uruchomić symulację. Wykreślić przebiegi czasowe modułu adder1.
6. Czy układ działa poprawnie? Sprawdzić z innymi wartościami liczb. Sprawdzić inne sposoby podawania danych (kilka liczb w jednym testbench'u itp).
7. Stworzyć nowy program wykorzystując jako podstawę plik adder.v. Program ten ma realizować funkcję mnożenia.

Plik multiplier.v

```
module multiplier
#(
parameter A_WIDTH = 32,
parameter B_WIDTH = 32
)
(
    input wire RST,
    input wire CLK,

    input  wire I_STB,
    output wire I_ACK,
    input  wire [A_WIDTH-1:0] I_DAT_A,
    input  wire [B_WIDTH-1:0] I_DAT_B,

    output reg  O_STB,
    output reg  [xxxxxxxxxxxxxxxxx:0] O_DAT,
    input  wire O_ACK
);

....
```

8. Co powinno się znaleźć w miejscu symbolu xxxxxxxxxxxxxxxxx aby układ działał poprawnie?
9. Poprawić testbench, aby testował moduł multiplier1 dla szerokości bitowej 16 danych wejściowych.
10. Skompilować i uruchomić symulację. Wykreślić przebiegi czasowe modułu multiplier1.
11. Czy układ działa poprawnie? Sprawdzić z innymi wartościami liczb poddawanych konwersji. Sprawdzić inne sposoby podawania danych (kilka liczb w jednym testbench'u itp) podobnie jak w punkcie 6.

12. Napisać nowy program realizujący funkcję mnożenia 32-bitowego z wykorzystaniem podrzędnych modułów mnozenia 16-bitowego i dodawania. Program powinien zaczynać się następująco:

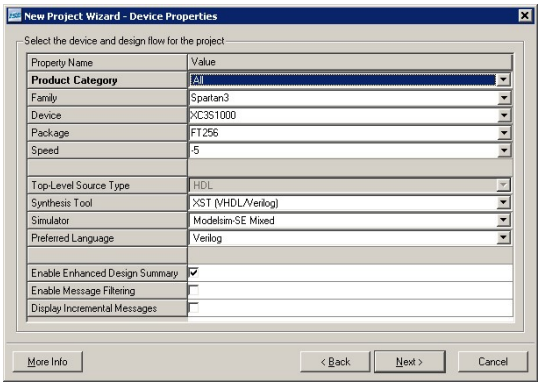
Plik multiplier_32bit.v

```
module multiplier_32bit
(
    input wire RST,
    input wire CLK,

    input  wire I_STB,
    output wire I_ACK,
    input  wire [31:0] I_DAT_A,
    input  wire [31:0] I_DAT_B,

    output wire  O_STB,
    output wire [63:0] O_DAT,
    input  wire O_ACK
);

....
```

Zasada mnożenia jest jak przy mnożeniu "pisemnym":

Liczba A			A_H	A_L
Liczba B	*		B_H	B_L
				B_L* A_L
	+		B_L* A_H	
	+		B_H * A_L	
	+	B_H * A_H		

13. Skompilować i uruchomić symulację. Wykreślić przebiegi czasowe modułu multiplier_32bit.
14. Czy układ działa poprawnie? Sprawdzić z innymi wartościami liczb poddawanych konwersji. Sprawdzić inne sposoby podawania danych (kilka liczb w jednym testbench'u itp) podobnie jak w punkcie 6.



15. Z wykorzystaniem Xilinx ISE, porównać wyniki syntezy dwóch implementacji mnożenia 32-bitowego.
16. Uruchomić środowisko Xilinx – ISE Project Navigator.
17. Utworzy nowy projekt (File/New Project). Nadać nazwę np. „mul_test”. Jako typ projektu wybrać „HDL”.
18. Ustawienia projektu wybrać jak na rysunku obok. Nie dodawać nowych plików źródłowych. Jako istniejące pliki źródłowe (existing sources) dodać pliki badanego modułu np. *„multiplier.v” (a później "multiplier_32bit.v"*. Nie dodawać pliku testbench’a!
19. Z panelu po lewej stronie wybrać „Synthesize – XST”. Odczytać maksymalną częstotliwość oraz wyniki syntezy HDL np.

```
Minimum period: 4.174ns (Maximum Frequency: 239.572MHz)
 . . .
Macro Statistics
# Registers                                          : 106
 Flip-Flops                                          : 106
# Comparators                                        : 10
 4-bit comparator lessequal                          : 10
=========================================================
```

Wybrać opcję „Implement design”. Odczytać rzeczywistą wielkość układu:

```
Logic Utilization:
Number of Slice Flip Flops:          106 out of  15,360    1%
Number of 4 input LUTs:              110 out of  15,360    1%
Logic Distribution:
  Number of occupied Slices:            56 out of   7,680    1%
```