Programmable Digital Systems – Exercise 5

Goals:

- introduction to ALUs (Arithmetic Logic Units),
- hardware implementation of wide-bit multiplier with use of narrow-bit multipliers.

Exercise:

1. Run Active-HDL, create a new workspace and a new design.
2. Enter the program below into editor:

testbench.v file:

```
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
```

adder.v file:

```
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. What do A_WIDTH and B_WIDTH parameters mean? What are advantages of such declaration?
4. Try to explain what is the meaning of the following line:

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

5. Compile and run the simulation. Analyze waveforms of the adder1 module.
6. Does the module work correctly? Check with different numbers and different diagrams of data input (more that one data in testbench etc.).
7. Create a new program with use of adder.v as a starting point. The program should implement a hardware multiplier.

multiplier.v file:

```
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  [XXXXXXXXXXXXXXXX:0] O_DAT,
    input  wire O_ACK
);

....
```

8. What should be typed in place of XXXXXXXXXXXXXXXX symbol, so that the program work well?
9. Patch the testbench program, so that is able to test multiplier1 module with 16-bit depth input data.
10. Compile and run the simulation. Analyze waveforms of the multiplier1 module.
11. Does the module work correctly? Check with different numbers and different diagrams of data input (more that one data in testbench etc.) just like in point 6.

12. Create a new program that implements a hardware 32-bit multiplier with use of hierarchical structure of 16-bit multipliers and some adders. The program should begin as follows:

multiplier_32bit.v file:

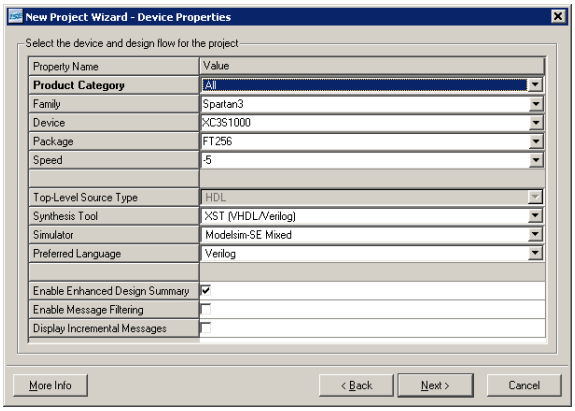
```
module multiplier_32bit
(
    input wire RST,
    input wire CLK,

    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
);

....
```

The rule for multiplication can be visualized as follows:

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. Compile and run the simulation. Analyze waveforms of the multiplier_32bit module.
14. Does the module work correctly? Check with different numbers and different diagrams of data input (more that one data in testbench etc.) just like in point 6.

15. With use of Xilinx ISE, compare synthesis results of two alternative implementations of 32-bit multiplier.
16. Run Xilinx – ISE Project Navigator.
17. Create new project (File/New Project). Give it a name e.g. „bcd_test”. Set type of the project to „HDL”.
18. The settings of the project should be match those on figure above. Do not add new source files. As “existing files”, add files of only analyzed module e.g. *„bin_to_bcd_simple.v”*. Do NOT add testbench files!
19. From left panel select „Synthesize – XST”. Read maximum frequency and HDL synthesis results e.g.




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

Select „Implement design”. Read the real size of the circuit:

```
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%
```