Programowalne Układy Cyfrowe – Ćwiczenie 3

Cele:

- zapoznanie z arytmetyką BCD,
- realizacja sprzętowa układu konwersji BIN na BCD,
- realizacja sprzętowa prostego układu dodawania BCD.

Wstęp:

BCD (ang. Binary-Coded Decimal czyli dziesiętny zakodowany dwójkowo) – sposób zapisu liczb polegający na zakodowaniu kolejnych cyfr dziesiętnych liczby dwójkowo przy użyciu czterech bitów stosowany w elektronice i informatyce. Taki zapis pozwala na łatwą konwersję liczby do i z systemu dziesiętnego, jest jednak nadmiarowy (wykorzystuje tylko 10 czterobitowych układów z 16 możliwych). Przykładowo, liczba 127 w podstawowym wariancie BCD wygląda tak: 0001 0010 0111

Źródło: Wikipedia

Przebieg ćwiczenia:

1. Uruchomić program Active-HDL, stworzyć nowy projekt.
2. W postaci trzech 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 [31:0] i_dat;
reg i_stb;

initial
begin
    i_dat <= 0;
    i_stb <= 0;
    #1000;
    i_dat <= 8;
    i_stb <= 1;
    #100;
    i_stb <= 0;

end

bin_to_bcd_simple test
(
.CLK(CLK),
.RST(RST),

.I_DAT(i_dat),
.I_STB(i_stb),

.O_DAT(),
.O_STB()
);

endmodule
```

Plik bin_to_bcd_simple.v

```
module bin_to_bcd_simple
(
input wire CLK,
```

```
input wire RST,

input wire  [31:0] I_DAT,
input wire         I_STB,

output wire [39:0] O_DAT,
output wire        O_STB
);

assign O_DAT[ 3: 0] = (I_DAT)%10;
assign O_DAT[ 7: 4] = (I_DAT/10)%10;
assign O_DAT[11: 8] = (I_DAT/100)%10;
assign O_DAT[15:12] = (I_DAT/1000)%10;
assign O_DAT[19:16] = (I_DAT/10000)%10;
assign O_DAT[23:20] = (I_DAT/100000)%10;
assign O_DAT[27:24] = (I_DAT/1000000)%10;
assign O_DAT[31:28] = (I_DAT/10000000)%10;
assign O_DAT[35:32] = (I_DAT/100000000)%10;
assign O_DAT[39:36] = (I_DAT/1000000000)%10;

assign O_STB = I_STB;

endmodule
```

3. Skompilować i uruchomić symulację. Wykreślić przebiegi czasowe modułu bin_to_bcd_simple.
4. Czy układ działa poprawnie? Sprawdzić z innymi wartościami liczb poddawanych konwersji: 16, 127…..
5. Dlaczego moduł w takiej postaci nie jest dobrą implementacją sprzętową?
6. Wprowadzić poniższe programy:

Plik bin_to_bcd.v

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

input wire  [31:0] I_DAT,
input wire         I_STB,

output wire [39:0] O_DAT,
output wire        O_STB
);
reg [31:0] bin;
reg [32:0] bst;

always @(posedge CLK or posedge RST)
if (RST) begin
    bin <= 0;
    bst <= 0;
end else begin
    if (I_STB)
    begin
           bin <= I_DAT;
           bst <= 1;
    end else begin
           bin <= (bin<<1);
           bst <= (bst<<1);
    end
end

wire [3:0] bcd9,bcd8,bcd7,bcd6,bcd5,bcd4,bcd3,bcd2,bcd1,bcd0;
wire [9:0] ovr;

bcd_shl_1 b0 (.CLK(CLK), .RST(RST), .ADD1(bin[31]), .DAT(bcd0), .OVERFLOW(ovr[0]) );
bcd_shl_1 b1 (.CLK(CLK), .RST(RST), .ADD1(ovr[0]), . DAT(bcd1), .OVERFLOW(ovr[1]) );
bcd_shl_1 b2 (.CLK(CLK), .RST(RST), .ADD1(ovr[1]), . DAT(bcd2), .OVERFLOW(ovr[2]) );
. . . .
bcd_shl_1 b9 (.CLK(CLK), .RST(RST), .ADD1(ovr[8]), . DAT(bcd9), .OVERFLOW(ovr[9]) );

assign O_DAT = {bcd9,bcd8,bcd7,bcd6,bcd5,bcd4,bcd3,bcd2,bcd1,bcd0};
assign O_STB = bst[32];

endmodule
```

Plik bcd_shl_1.v

```
module bcd_shl_1
(
    input wire CLK,
    input wire RST,

    input wire       ADD1,
    output reg [3:0] DAT,
    output reg       OVERFLOW
);

always @(posedge CLK or posedge RST)
if (RST) begin
    DAT <= 0;
    OVERFLOW <= 0;
end else begin
    DAT <=
           (DAT==0) ? {3'd0, ADD1} : // 0-1        no overflow
           (DAT==1) ? {3'd1, ADD1} : // 2-3        no overflow
           (DAT==2) ? {3'd2, ADD1} : // 4-5        no overflow
           (DAT==3) ? {3'd3, ADD1} : // 6-7        no overflow
           (DAT==4) ? {3'd4, ADD1} : // 8-9        no overflow
           (DAT==5) ? {3'd0, ADD1} : // 0-1
           (DAT==6) ? {3'd1, ADD1} : // 2-3
           (DAT==7) ? {3'd2, ADD1} : // 4-5
           (DAT==8) ? {3'd3, ADD1} : // 6-7
           (DAT==9) ? {3'd4, ADD1} : // 8-9
           {3'd7, ADD1};   // when error
    OVERFLOW <= ((DAT>=0) && (DAT<=4)) ? 1'b0 : 1'b1;
end
endmodule
```

7. Poprawić plik testbench.v aby wykorzystywać moduł bin_to_bcd zamiast bin_to_bcd_simple.
8. Skompilować i uruchomić symulację. Wykreślić przebiegi czasowe modułu bin_to_bcd.
9. Czy układ działa poprawnie? Sprawdzić z innymi wartościami liczb poddawanych konwersji: 16, 127…..
10. Przyjrzeć się przebiegom czasowych linii „bcd0”….”bcd9” i znaleźć na nich przyczynę nieprawidłowego działania modułu. Co należy zmienić?

11. Skorygowany program uzupełnić o sterowanie w stylu „handshake” z liniami STB i ACK. Nagłówek modułu w pliku bin_to_bcd powinien posiadać następujące porty:

```
input CLK,
input RST,

input [31:0] I_DAT,
input        I_STB,
output       I_ACK,

output [39:0] O_DAT,
output        O_STB,
input         O_ACK
);
```

Najprostszym schemat dokonania takiej modyfikacji, jest wprowadzenie rejestru zajętości/gotowości:

```
reg ready;
```

który będzie domyślnie „resetowany” na wartość „1”, ustawiany na „0” przy nowej danej i przywracany do „1” po zakończeniu pracy:

```
always @(posedge CLK or posedge RST)
if (RST) ready <= 1;
else if (O_ACK&O_STB) ready <= 1;
else if (I_STB) ready <= 0;

assign I_ACK = I_STB && ready;
```

jak również zmodyfikować mechanizm przyjmowania danych z zapisu:
`if (I_STB)` na zapis: `if (I_STB && ready)` lub inaczej `if (I_ACK)`

a)
Aby układ czekał na wyjściowy O_ACK, konieczne jest zapamiętanie wyniku, przerabiając linie O_DAT i O_STB na rejestry (reg), oraz dopisując poniższy proces zamiast „assign O_...=”

```
always @(posedge CLK or posedge RST)
if (RST) begin
    O_STB <= 0;
    O_DAT <= 0;
end else if (bst[32]) begin
    O_STB <= 1;
    O_DAT <= {bcd9,bcd8,bcd7,bcd6,bcd5,bcd4,bcd3,bcd2,bcd1,bcd0};
end else if (O_ACK) O_STB <= 0;
```

b) alternatywnie, można zatrzymać działanie układu wprowadzając poprawki do bcd_shl_1:

```
    input wire       ENABLE,
```

oraz

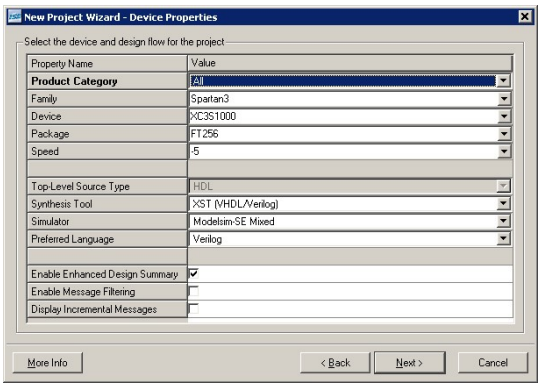
```
    if (ENABLE) DAT <=
```

a także podłączyć linie ENABLE tych modułach do nowej linii „en”:

```
    wire en = ~O_STB || ready;
```

które również sterowałaby przesuwanie bitowym.

```
    end else if (en) begin
           bin <= (bin<<1);
           bst <= (bst<<1);
    end
```



12. W testbench’u dodać brakujące linie. Wysterować liniami I_STB, I_ACK, O_STB i O_ACK:

```
initial
begin
    i_dat <= 0;
    i_stb <= 0;
    #1000;
    i_dat <= 16;
    i_stb <= 1;
    #100;
    i_stb <= 0;

end
```

13. Skompilować i uruchomić symulację. Wykreślić przebiegi czasowe modulu bin_to_bcd.
14. Z wykorzystaniem Xilinx ISE, porównać wyniki syntezy dwóch implementacji: a i b, oraz sprawdzić, że czy program z punktu „2” się syntezuje.
15. Uruchomić środowisko Xilinx – ISE Project Navigator.
16. Utworzy nowy projekt (File/New Project). Nadać nazwę np. „bcd_test”. Jako typ projektu wybrać „HDL”.
17. Ustawienia projektu wybrać jak na rysunku powyżej. Nie dodawać nowych plików źródłowych. Jako istniejące pliki źródłowe (existing sources) dodać pliki badanego modułu np. „*bin_to_bcd_simple.v”.* Nie dodawać pliku testbench’a!
18. 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%
```

19. Podpowiedzi do punktów 9:
    - czy linia OVERFLOW jest ustawiana na czas?
    - czy moduł bcd_shl_1 jest odpowiednio reset-owany gdy nadchodzą nowe dane?