// Plik double\_dabble.v  
  
`default\_nettype none  
  
module **DoubleDabble** #(  
 parameter INPUT\_BITS = 16,  
 parameter OUTPUT\_DIGITS = 5,  
 parameter OUTPUT\_BITS = OUTPUT\_DIGITS \* 4  
)(  
 input wire [ INPUT\_BITS-1:0] Binary\_i,  
 output reg [OUTPUT\_BITS-1:0] BCD\_o  
);  
   
 // Iteratory pętli for  
 integer i; // 1  
 integer j; // 2  
   
 // Blok kombinacyjny  
 always @(\*) begin // 3  
 BCD\_o = 0; // 4  
   
 // Dla każdego bitu na wejściu  
 for(i=0; i<INPUT\_BITS; i=i+1) begin // 5  
   
 // Dla każdej cyfry na wyjściu  
 for(j=0; j<OUTPUT\_BITS; j=j+4) begin // 6  
   
 // Jeżeli cyfra jest >= 5  
 if(BCD\_o[j+:4] >= 4'd5) begin // 7  
   
 // To dodaj 3 do tej cyfry  
 BCD\_o[j+:4] = BCD\_o[j+:4] + 4'd3; // 8  
 end  
 end  
   
 // Przesuń bity w rejestrze  
 // i dodaj kolejny bit z wejścia danych Binary\_i  
 BCD\_o = {BCD\_o[OUTPUT\_BITS-2:0], // 9

Binary\_i[(INPUT\_BITS-1)-i]};  
 end  
 end  
  
endmodule  
  
`default\_nettype wire

Listing 1. Kod pliku double\_dabble.v

**// Plik double\_dabble\_tb.v**

**// (...)  
 // Testuj wszystkie wartości od zera do maksimum  
 for(i=0; i<=MaxInput; i=i+1) begin // 1  
 @(posedge Clock); // 2  
 Binary <= i; // 3  
 Verify(Binary, BCD); // 4  
 end  
   
 // Testuj wszystkie wartości od maksimum do zera  
 for(i=MaxInput; i>=0; i=i-1) begin // 5  
 @(posedge Clock);  
 Binary <= i;  
 Verify(Binary, BCD);  
 end  
// (...)**

Listing 2. Fragment pliku double\_dabble\_tb.v

@echo off

iverilog -o double\_dabble.o ^

double\_dabble.v ^

double\_dabble\_tb.v

vvp double\_dabble.o

del double\_dabble.o

Listing 3. Kod pliku double\_dabble.bat

VCD info: dumpfile double\_dabble.vcd opened for output.

===== START =====

INPUT\_BITS: 8

OUTPUT\_BITS: 12

OUTPUT\_DIGITS: 3

MaxInput: 255

Pass: 512

Fail: 0

====== END ======

double\_dabble\_tb.v:100: $finish called at 1025000 (1ns)

Listing 4. Log po symulacji

// Plik top.v  
  
`default\_nettype none  
  
module **top** #(  
 parameter CLOCK\_HZ = 25\_000\_000  
)(  
 input wire Clock, // Pin 20  
 input wire Reset, // Pin 17  
 input wire EncoderA\_i, // Pin 68  
 input wire EncoderB\_i, // Pin 67  
 output wire [7:0] Cathodes\_o, // Pin 40 41 42 43 45 47 51 52  
 output wire [7:0] Segments\_o // Pin 39 38 37 36 35 34 30 29  
);  
  
 // Zmienna na potrzeby enkodera obrotowego   
 wire Increment;  
 wire Decrement;

// Instancja modułu enkodera obrotowego   
 **Encoder** Encoder\_inst(  
 .Clock(Clock),  
 .Reset(Reset),  
 .AsyncA\_i(EncoderA\_i),  
 .AsyncB\_i(EncoderB\_i),  
 .AsyncS\_i(1'b1),  
 .Increment\_o(Increment),  
 .Decrement\_o(Decrement),  
 .ButtonPress\_o(),  
 .ButtonRelease\_o(),  
 .ButtonState\_o()   
 );  
   
 // Licznik liczący w górę/dół w zakresie 0...9999 (dziesiętnie)  
 reg [15:0] Counter; // 1

// Logika licznika  
 always @(posedge Clock, negedge Reset) begin  
 if(!Reset) begin  
 Counter <= 0;  
 end else if(Increment) begin  
 if(Counter == 16'd9999)  
 Counter <= 16'd0;  
 else  
 Counter <= Counter + 1'b1;  
 end else if(Decrement) begin  
 if(Counter == 16'd0)  
 Counter <= 16'd9999;  
 else  
 Counter <= Counter - 1'b1;  
 end  
 end  
   
 // Zmienna do przekazywania wyniku z modułu Double Dabble

// do modułu wyświetlacza  
 wire [15:0] Decimal;

**// Instancja modułu konwertera BIN/BCD  
 DoubleDabble #(  
 .INPUT\_BITS(16),  
 .OUTPUT\_DIGITS(4)  
 ) DoubleDabble\_inst(  
 .Binary\_i(Counter), // 2  
 .BCD\_o(Decimal) // 3  
 );  
   
 // Instancja modułu 8-cyfrowego wyświetlacza 7-segmentowego  
 DisplayMultiplex #(  
 .CLOCK\_HZ(CLOCK\_HZ),  
 .SWITCH\_PERIOD\_US(1000),  
 .DIGITS(8)  
 ) DisplayMultiplex\_inst(  
 .Clock(Clock),  
 .Reset(Reset),  
 .Data\_i({Decimal, Counter}), // 4  
 .DecimalPoints\_i(8'b00010000),  
 .Cathodes\_o(Cathodes\_o),  
 .Segments\_o(Segments\_o),  
 .SwitchCathode\_o()  
 );   
   
endmodule  
  
`default\_nettype wire**

Listing. 5. Kod pliku top.v