Zadanie dodatkowe do projektu 1: weryfikacja operacji z exe_unit

poniżej znajdują się weryfikacje modelu jednostki w postaci tablic prawdy (flagę przepełnienia testuję każdorazowo, jeśli moduł ją obsługuje, dlatego ta flaga nie ma swojej oddzielnej tablicy prawdy w przeciwieństwie do reszty flag)

1) Weryfikacja modułu arytmetycznego przesunięcia w prawo (na 2 bitowych wejść i wyjścia):

```
module \$paramod\arythm_shift\BITS
  wire _0_;
  wire _1_;
  wire _2_;
  wire _3_;
  input [1:0] i_a;
  input [1:0] i_b;
  output [1:0] o_output;
  assign _0_ = i_b[0] | i_b[1];
  assign _1_ = ~_0_;
  assign _2_ = i_a[0] & _1_;
  assign _3_ = i_a[1] & _0_;
  assign o_output[0] = _2_ | _3_;
  assign o_output[1] = i_a[1];
endmodule
```

Tablica prawdy:

				Wire _0_	wire _1_	wire _2_	wire _3_	Output[1]	Output[0]
i_a [1]	i_a [0]	i_b [1]	i_b [0]	i_b[0] | i_b[1]	~_0_	i_a[0] & _1_	i_a[1] & _0_	i_a[1]	_2_ | _3_
0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	0	1
0	1	0	1	1	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	0	0	1	0
1	0	0	1	1	0	0	1	1	1
1	0	1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	1	1	1
1	1	0	0	0	1	1	0	1	1
1	1	0	1	1	0	0	1	1	1
1	1	1	0	1	0	0	1	1	1
1	1	1	1	1	0	0	1	1	1

2) Weryfikacja modułu sumatora (i przy okazji flagi przepełnienia w wypadku jego wywołania; moduł działający na 2 bitach):

```
module \$paramod\fulladder\BITS=s32'000
  wire _00_;
  wire _01_;
  wire _02_;
  wire _03_;
  input [1:0] i_a;
  input [1:0] i_b;
  output o_carry;
  output [1:0] o_sum;
  assign _02_ = i_a[0] & i_b[0];
  assign _03_ = ~_02_;
  assign _00_ = i_a[1] & i_b[1];
  assign _01_ = i_a[1] ^ i_b[1];
  assign o_sum[1] = _02_ ^ _01_;
  assign o_carry = _03_ & _00_;
  assign o_sum[0] = i_a[0] ^ i_b[0];
endmodule
```

Tablica prawdy:

				Wire _0_	wire _1_	wire _2_	wire _3_	o_sum[1]	o_sum[0]	o_carry
i_a [1]	i_a [0]	i_b [1]	i_b [0]	i_a[1] & i_b[1]	i_a[1] ^ i_b[1]	i_a[0] & i_b[0]	~_02_	_02_ ^ _01_	i_a[0] ^ i_b[0]	_03_ & _00_
0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	1	0	1	0
0	0	1	0	0	1	0	1	1	0	0
0	0	1	1	0	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0	1	0
0	1	0	1	0	0	1	0	1	0	0
0	1	1	0	0	1	0	1	1	1	0
0	1	1	1	0	1	1	0	0	0	0
1	0	0	0	0	1	0	1	1	0	0
1	0	0	1	0	1	0	1	1	1	0
1	0	1	0	1	0	0	1	0	0	1
1	0	1	1	1	0	0	1	0	1	1
1	1	0	0	0	1	0	1	1	1	0
1	1	0	1	0	1	1	0	0	0	0
1	1	1	0	1	0	0	1	0	1	1
1	1	1	1	1	0	1	0	1	0	0

3) Weryfikacja operacji or (na 2 bitach):

```
module exe_unit_rtl(i_argA, i_argB, i_oper, o_result);
  input [1:0] i_argA;
  input [1:0] i_argB;
  input [3:0] i_oper;
  output [1:0] o_result;
  assign o_result[0] = i_argB[0] | i_argA[0];
  assign o_result[1] = i_argB[1] | i_argA[1];
endmodule
```

Tablica prawdy:

				o_result[1]	o_result[0]
i_argA [1]	i_argA[0]	i_argB[1]	i_argB[0]	i_argB[1] | i_argA[1]	i_argB[0] | i_argA[0]
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	1

0	1	1	1	1	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

4) Weryfikacja operacji nor (na 2 bitach):

```
module exe_unit_rtl(i_argA, i_argB, i_oper, o_result);
  wire _0_;
  wire _1_;
  input [1:0] i_argA;
  input [1:0] i_argB;
  input [3:0] i_oper;
  output [1:0] o_result;
  assign _0_ = i_argB[0] | i_argA[0];
  assign o_result[0] = ~_0_;
  assign _1_ = i_argB[1] | i_argA[1];
  assign o_result[1] = ~_1_;
endmodule
```

Tablica prawdy:

				Wire _0_	wire _1_	o_result[1]	o_result[0]
i_argA [1]	i_argA[0]	i_argB[1]	i_argB[0]	i_argB[0] | i_argA[0]	i_argB[1] | i_argA[1]	~_1_	~_0_
0	0	0	0	0	0	1	1
0	0	0	1	1	0	1	0
0	0	1	0	0	1	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	1	1	0	0
1	0	1	0	0	1	0	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	0
1	1	0	1	1	1	0	0
1	1	1	0	1	1	0	0
1	1	1	1	1	1	0	0

5) Weryfikacja logicznego przesunięcia w prawo (na 2 bitach):

```
module exe_unit_rtl(i_argA, i_argB, i_oper, o_result);
  wire _00_;
  wire _01_;
  wire _02_;
  wire _03_;
  wire _04_;
  wire _05_;
  input [1:0] i_argA;
  input [1:0] i_argB;
  input [3:0] i_oper;
  output [1:0] o_result;
  assign _03_ = ~i_argB[0];
  assign _04_ = ~i_argB[1];
  assign _05_ = i_argA[0] & _03_;
  assign _00_ = i_argA[1] & i_argB[0];
  assign _01_ = _05_ | _00_;
  assign _02_ = i_argA[1] & _04_;
  assign o_result[0] = _04_ & _01_;
  assign o_result[1] = _03_ & _02_;
endmodule
```

Tablica prawdy:

				Wire _0_	wire _1_	wire _2_	wire _3_	Wire _4_	Wire _5_	o_result[1]	o_result[0]
i_argA [1]	i_argA[0]	i_argB[1]	i_argB[0]	i_argA[1] & i_argB[0]	_05_ | _00_	i_argA[1] & _04_	~i_argB[0]	~i_argB[1]	i_argA[0] & _03_	_03_ & _02_	_04_ & _01_
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	1	1	1	0	1
0	1	0	1	0	0	0	0	1	0	0	0
0	1	1	0	0	1	0	1	0	1	0	0
0	1	1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	0	1	0
1	0	0	1	1	1	1	0	1	0	0	1
1	0	1	0	0	0	0	1	0	0	0	0
1	0	1	1	1	1	0	0	0	0	0	0
1	1	0	0	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	0	1	0	0	1
1	1	1	0	0	1	0	1	0	1	0	0
1	1	1	1	1	1	0	0	0	0	0	0

6) Weryfikacja modułu licznika zer w obu argumentach wejściowych (na 2 bitach):

```
module \$paramod\count_0\BITS=s32'00000000000000
  wire _00_;
  wire _01_;
  wire _02_;
  wire _03_;
  wire _04_;
  wire _05_;
  wire _06_;
  input [1:0] i_input1;
  input [1:0] i_input2;
  output [1:0] o_output;
  output o_overflow;
  assign _05_ = i_input2[1] | i_input2[0];
  assign _06_ = i_input1[1] | i_input1[0];
  assign _00_ = _05_ | _06_;
  assign o_overflow = ~_00_;
  assign _01_ = i_input1[1] ^ i_input1[0];
  assign _02_ = i_input2[1] ^ i_input2[0];
  assign _03_ = _01_ & _02_;
  assign _04_ = _05_ ^ _06_;
  assign o_output[1] = _03_ | _04_;
  assign o_output[0] = _01_ ^ _02_;
endmodule
```

Tablica prawdy:

				Wire _0_	wire _1_	wire _2_	wire _3_	Wire _4_	Wire _5_	Wire _6_	o_output[1]	o_output[0]	o_overflow
i_input1 [1]	i_input1[0]	i_input2[1]	i_input2[0]	_05_ | _06_	i_input1[1] ^ i_input1[0]	i_input2[1] ^ i_input2[0]	_01_ & _02_	_05_ ^ _06_	i_input2[1] | i_input2[0]	i_input1[1] | i_input1[0]	_03_ | _04_	_01_ ^ _02_	~_00_
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	1	0	1	1	0	1	1	0
0	0	1	0	1	0	1	0	1	1	0	1	1	0
0	0	1	1	1	0	0	0	1	1	0	1	0	0
0	1	0	0	1	1	0	0	1	0	1	1	1	0
0	1	0	1	1	1	1	1	0	1	1	1	0	0
0	1	1	0	1	1	1	1	0	1	1	1	0	0
0	1	1	1	1	1	0	0	0	1	1	0	1	0
1	0	0	0	1	1	0	0	1	0	1	1	1	0
1	0	0	1	1	1	1	1	0	1	1	1	0	0
1	0	1	0	1	1	1	1	0	1	1	1	0	0
1	0	1	1	1	1	0	0	0	1	1	0	1	0
1	1	0	0	1	0	0	0	1	0	1	1	0	0
1	1	0	1	1	0	1	0	0	1	1	0	1	0
1	1	1	0	1	0	1	0	0	1	1	0	1	0
1	1	1	1	1	0	0	0	0	1	1	0	0	0

7) Weryfikacja modułu realizującego flagi pf oraz nf (na 3 bitach).

```
module \$paramod\pf_nf\BITS=s32'00000000000000000000000000000011 (i_input, o_pf, o_nf);
  wire _0_;
  wire _1_;
  input [2:0] i_input;
  output o_nf;
  output o_pf;
  wire [2:0] s_counter;
  assign _0_ = ~i_input[1];
  assign _1_ = i_input[0] ^ i_input[2];
  assign o_pf = i_input[1] ^ _1_;
  assign o_nf = _0_ ^ _1_;
  assign s_counter[0] = o_pf;
endmodule
```

Tablica prawdy:

			Wire _0_	Wire _1_	Wire s_counter[0]	o_pf	o_nf
i_input [2]	i_input [1]	i_input[0]	~i_input[1]	i_input[0] ^ i_input[2]	o_pf	i_input[1] ^ _1_	_0_ ^ _1_
0	0	0	1	0	0	0	1
0	0	1	1	1	1	1	0
0	1	0	0	0	1	1	0
0	1	1	0	1	0	0	1
1	0	0	1	1	1	1	0
1	0	1	1	0	0	0	1
1	1	0	0	1	0	0	1
1	1	1	0	0	1	1	0

Yosys dość ciekawie poradził sobie ze zliczaniem liczby jedynek, flagi pf i nf działają poprawnie.

8) Weryfikacja modułu konwersji z kodu termometrycznego na kod nkb (3 bity)

```
module \$paramod\thermo2bin\LEN=s32'00000000000
  wire _0_;
  input [2:0] i_thermo;
  output [2:0] o_nkb;
  assign _0_ = ~i_thermo[2];
  assign o_nkb[0] = i_thermo[1] & _0_;
  assign o_nkb[2:1] = { 1'h0, i_thermo[2] };
endmodule
```

Tablica prawdy:

			Wire _0_	o_nkb[2:1]	o_nkb[0]
i_thermo [2]	i_thermo [1]	i_thermo[0]	~i_thermo[2]	{ 1'h0, i_thermo[2] }	i_thermo[1] & _0_
0	0	0	1	0 0	0
0	0	1	1	0 0	0
0	1	0	1	0 0	1
0	1	1	1	0 0	1
1	0	0	0	0 1	0
1	0	1	0	0 1	0
1	1	0	0	0 1	0
1	1	1	0	0 1	0

9) Weryfikacja modułu konwersji kodu binarnego na kod gorącej jedynki:

```
module \$paramod\binary2onehot\BITS=s32'00000000000
  wire _00_;
  wire _01_;
  wire _02_;
  wire _03_;
  wire _04_;
  input [2:0] i_nkb;
  output [2:0] o_onehot;
  output o_overflow;
  assign _03_ = i_nkb[2] | i_nkb[0];
  assign _04_ = ~_03_;
  assign o_onehot[2] = i_nkb[1] & _04_;
  assign _00_ = i_nkb[2] | i_nkb[1];
  assign _01_ = ~_00_;
  assign _02_ = i_nkb[0] | _00_;
  assign o_onehot[0] = ~_02_;
  assign o_onehot[1] = i_nkb[0] & _01_;
  assign o_overflow = _03_ & _00_;
endmodule
```

Tablica prawdy:

			Wire _0_	Wire _1_	Wire _2_	Wire _3_	Wire _4_	o_onehot[2]	o_onehot[1]	o_onehot[0]	o_overflow
i_nkb [2]	i_nkb [1]	i_nkb[0]	i_nkb[2] | i_nkb[1]	~_00_	i_nkb[0] | _00_	i_nkb[2] | i_nkb[0]	~_03_	i_nkb[1] & _04_	i_nkb[0] & _01_	~_02_	_03_ & _00_
0	0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	1	1	1	0	0	1	0	0
0	1	0	1	0	1	0	1	1	0	0	0
0	1	1	1	0	1	1	0	0	0	0	1
1	0	0	1	0	1	1	0	0	0	0	1
1	0	1	1	0	1	1	0	0	0	0	1
1	1	0	1	0	1	1	0	0	0	0	1
1	1	1	1	0	1	1	0	0	0	0	1

10) Weryfikacja modułu konwersji z kodu U2 na kod U1 (na 3 bitach):

```
module \$paramod\u2tou1\BITS=s32'00000000000000000
  wire _00_;
  wire _01_;
  wire _02_;
  wire _03_;
  input [2:0] i_input;
  output [2:0] o_output;
  output o_overflow;
  assign _02_ = ~i_input[0];
  assign _03_ = ~i_input[1];
  assign _00_ = i_input[2] & _02_;
  assign o_output[0] = i_input[2] ^ i_input[0];
  assign _01_ = i_input[0] | i_input[1];
  assign o_overflow = _03_ & _00_;
  assign o_output[1] = i_input[1] ^ _00_;
  assign o_output[2] = i_input[2] & _01_;
endmodule
```

Tabela prawdy:

			Wire _0_	Wire _1_	Wire _2_	Wire _3_	o_output[2]	o_output[1]	o_output[0]	o_overflow
i_input [2]	i_input [1]	i_input[0]	i_input[2] & _02_	i_input[0] | i_input[1]	~i_input[0]	~i_input[1]	i_input[2] & _01_	i_input[1] ^ _00_	i_input[2] ^ i_input[0]	_03_ & _00_
0	0	0	0	0	1	1	0	0	0	0
0	0	1	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	0	1	0	0
0	1	1	0	1	0	0	0	1	1	0
1	0	0	1	0	1	1	0	1	1	1
1	0	1	0	1	0	1	1	0	0	0
1	1	0	1	1	1	0	1	0	1	0
1	1	1	0	1	0	0	1	1	0	0

11) Weryfikacja modułu konwersji kodu znak-moduł na kod U2:

```
module \$paramod\sm_to_u2\BITS=s32'00000000000000000000000000000011
  wire _0_;
  input [2:0] i_input;
  output [2:0] o_output;
  assign _0_ = i_input[0] & i_input[2];
  assign o_output[1] = i_input[1] ^ _0_;
  assign { o_output[2], o_output[0] } = { i_input[2], i_input[0] };
endmodule
```

Tablica prawdy:

			Wire _0_	o_output[2]	o_output[1]	o_output[0]
i_input [2]	i_input [1]	i_input[0]	i_input[0] & i_input[2]	i_input[2]	i_input[1] ^ _0_	i_input[0]
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	0	0	0	1	0	0
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	1	1	1	0	1

12) Weryfikacja flagi of (of=1 gdy o_result=’1) na przykładzie operacji or na 2 bitach:

```
module exe_unit_rtl(i_argA, i_argB, i_oper, o_result, o_of);
  input [1:0] i_argA;
  input [1:0] i_argB;
  input [3:0] i_oper;
  output o_of;
  output [1:0] o_result;
  assign o_result[1] = i_argB[1] | i_argA[1];
  assign o_result[0] = i_argB[0] | i_argA[0];
  assign o_of = o_result[1] & o_result[0];
endmodule
```

Tablica prawdy:

				o_result[1]	o_result[0]	o_of
i_argA [1]	i_argA [0]	i_argB [1]	i_argB [0]	i_argB[1] | i_argA[1]	i_argB[0] | i_argA[0]	o_result[1] & o_result[0]
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	1
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	1	1
1	0	1	0	1	0	0
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1