# SYSTEMY CYFROWE I KOMPUTEROWE

Agnieszka Klępka
310712
01158709@pw.edu.pl

PROJEKT INDYWIDUALNY 1
Specyfikacja jednostki wykonawczej exe_unit operującej na danych w kodzie U2

Układ realizuje funkcje jednostki arytmetyczno - logicznej. Potrafi wykonać 12 operacji arytmetycznych i logicznych na dwóch danych m-bitowych lub jednoargumentowe.

W module exe_unit zdefiniowane są trzy m-bitowe wejścia i_argA, i_argB, w tym jedno wyjście n-bitowe sterujące i_oper, wyjście m-bitowe o_result, które określa stan oraz cztery dodatkowe wyjścia jednobitowe o_BF, o_NF, o_PF, o_BF, które są znacznikami wykonanej operacji.

Dane są zapisane w konwencji little endian, startując od najmniej znaczącego bitu i poruszając się w lewo po bitach.

Rozważam jednostkę dla przemyślanych danych, bity są ustawione poprawnie.
W kodzie exe_unit przyjęłam wartości M = 8 oraz N = 8.

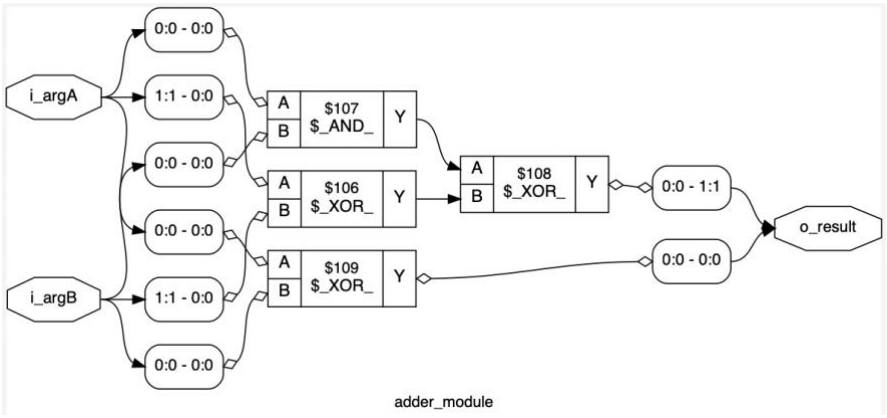
## WEJŚCIA I WYJŚCIA

i_argA -> pierwsze wejście M-bitowe do przetworzenia w jednostce arytmetyczno-logicznej
i_argB -> drugie wejście M-bitowe do przetworzenia w jednostce arytmetyczno-logicznej
i_oper -> wejście sterujące N-bitowe, służące do wyboru operacji wykonywanej przez ALU

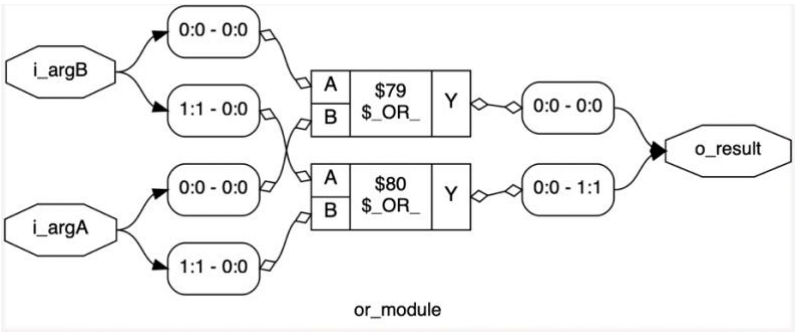
o_result -> M-bitowe wyjście, na które podawany jest wynik operacji wykonywanych w jednostce

LISTA REALIZOWANYCH FUNKCJI

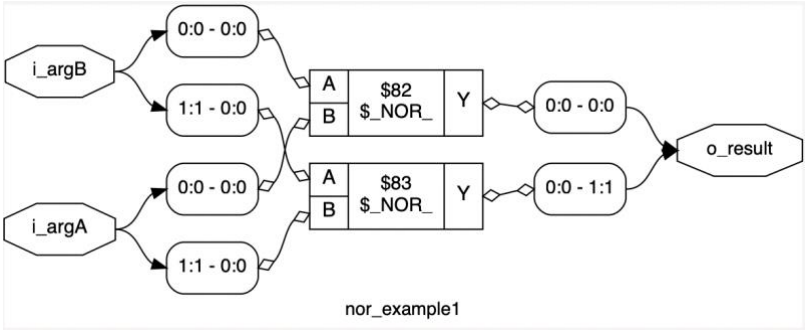
Jednostka realizuje 12 funkcji arytmetyczno – logicznych. Wykonuje na pierwszym miejscu najbardziej podstawowe działanie – dodawanie. Zawiera dwa wejścia oraz wyjście. Jest zrealizowane za pomocą sumatora pełnego. Użyta została tutaj jedna bramka AND oraz trzy bramki XOR. Na wejście i_opera trzeba podać 4’b0000.



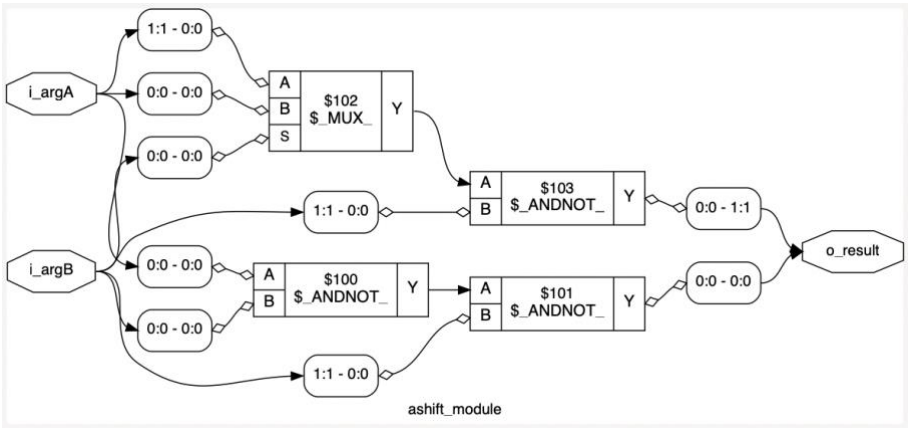

Następną operacją jest or, czyli suma logiczna argumentów. Zawiera dwa wejścia oraz wyjście. Ten moduł został zrealizowany na dwóch bramkach OR. Na wejście i_opera trzeba podać 4’b0001.



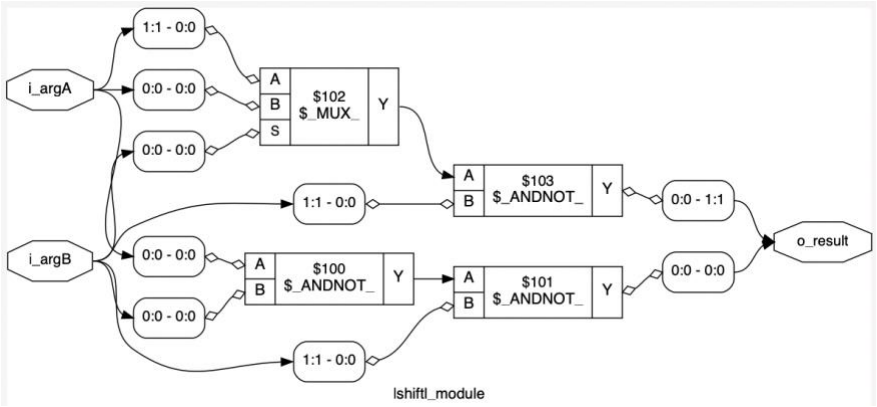

Operacja NOR – zanegowana suma logiczna – została zrealizowana na dwóch bramkach NOR. Zawiera dwa wejścia oraz wyjście. Na wejście i_opera trzeba podać 4’b0010.



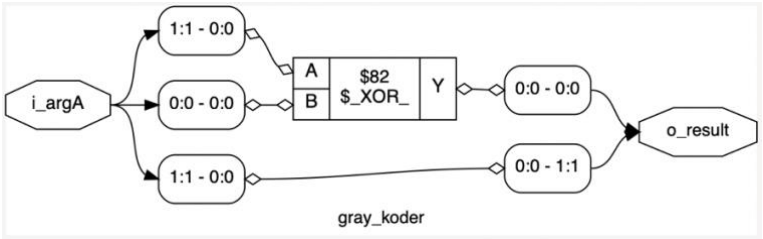
Schemat arytmetycznego przesunięcia argumentów w lewo zawiera trzy bramki NAND oraz multiplexer. Zawiera dwa wejścia oraz wyjście. Na wejście i_opera trzeba podać 4’b0011.



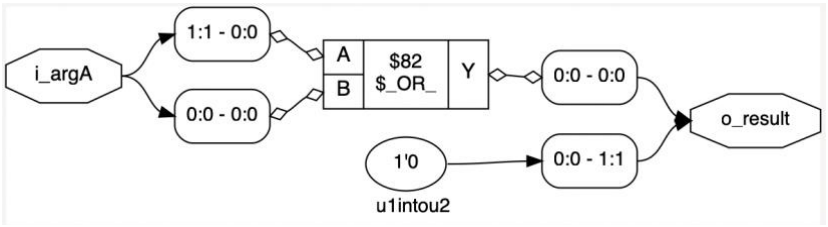

Logiczne przesunięcie argumentów w lewo również zawiera trzy bramki NAND oraz multiplexer. Zawiera dwa wejścia oraz wyjście. Na wejście i_opera trzeba podać 4’b0100.



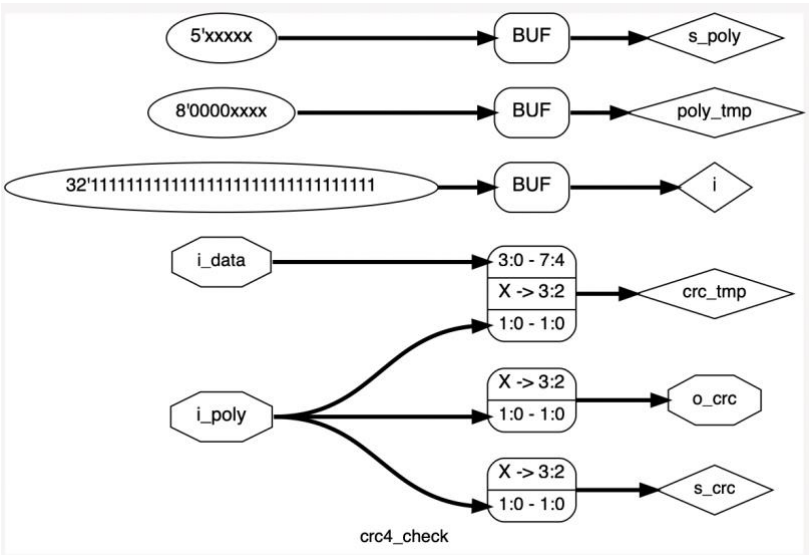

Operacja konwersji danej wejściowej z kodu U2 na kod GRAYA została zrealizowana na jednym XORze. Zawiera jedno wejście oraz wyjście. Na wejście i_opera trzeba podać 4’b0101.

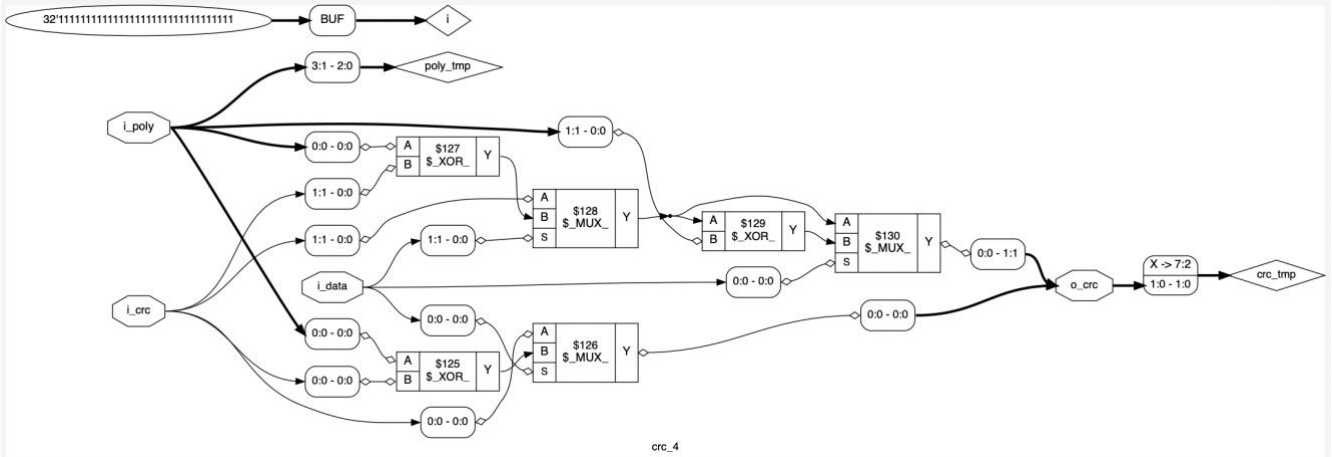


Następną operacją jest konwersja danej wejściowej z U1 na kod U2. W tym przypadku użyto jednej bramki OR. Zawiera jedno wejście oraz wyjście. Na wejście i_opera trzeba podać 4’b0110.

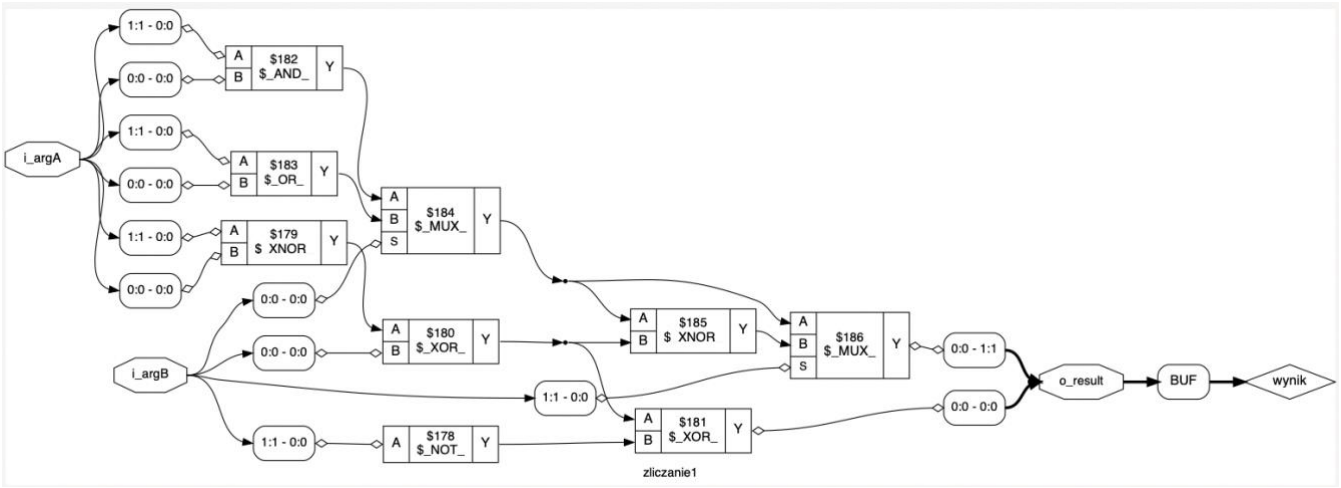



Następnymi operacjami są sprawdzanie zgodności i wyznaczanie kodu CRC-4. Wyznaczanie kodu zostało zrealizowane na trzech XORach oraz trzech multiplexerach. Zawiera dwa wejścia oraz wyjście. Na wejście i_opera trzeba podać odpowiednio 4’b1001 oraz 4’b1010.

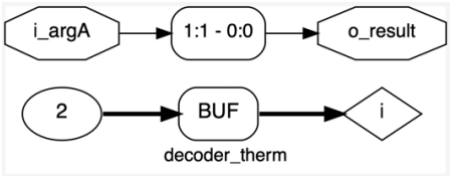


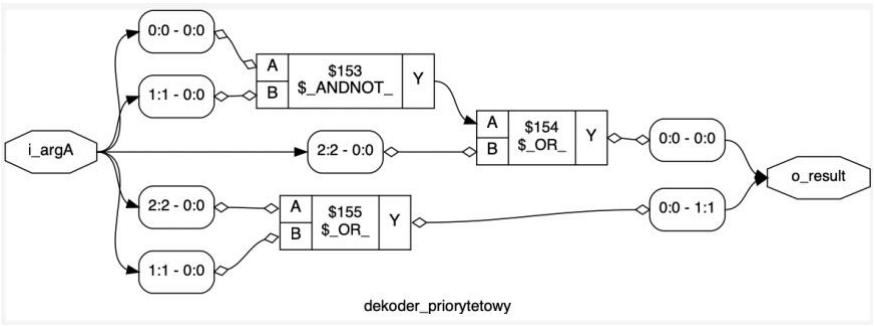
Realizacja zliczania sumarycznej liczby jedynek w obu argumentach wejściowych zawiera jedną bramkę AND, NOT, OR oraz po dwie bramki XNOR, XOR oraz dwa multiplexery. Zawiera dwa wejścia oraz wyjście.  Na wejście i_opera trzeba podać 4’b1000.



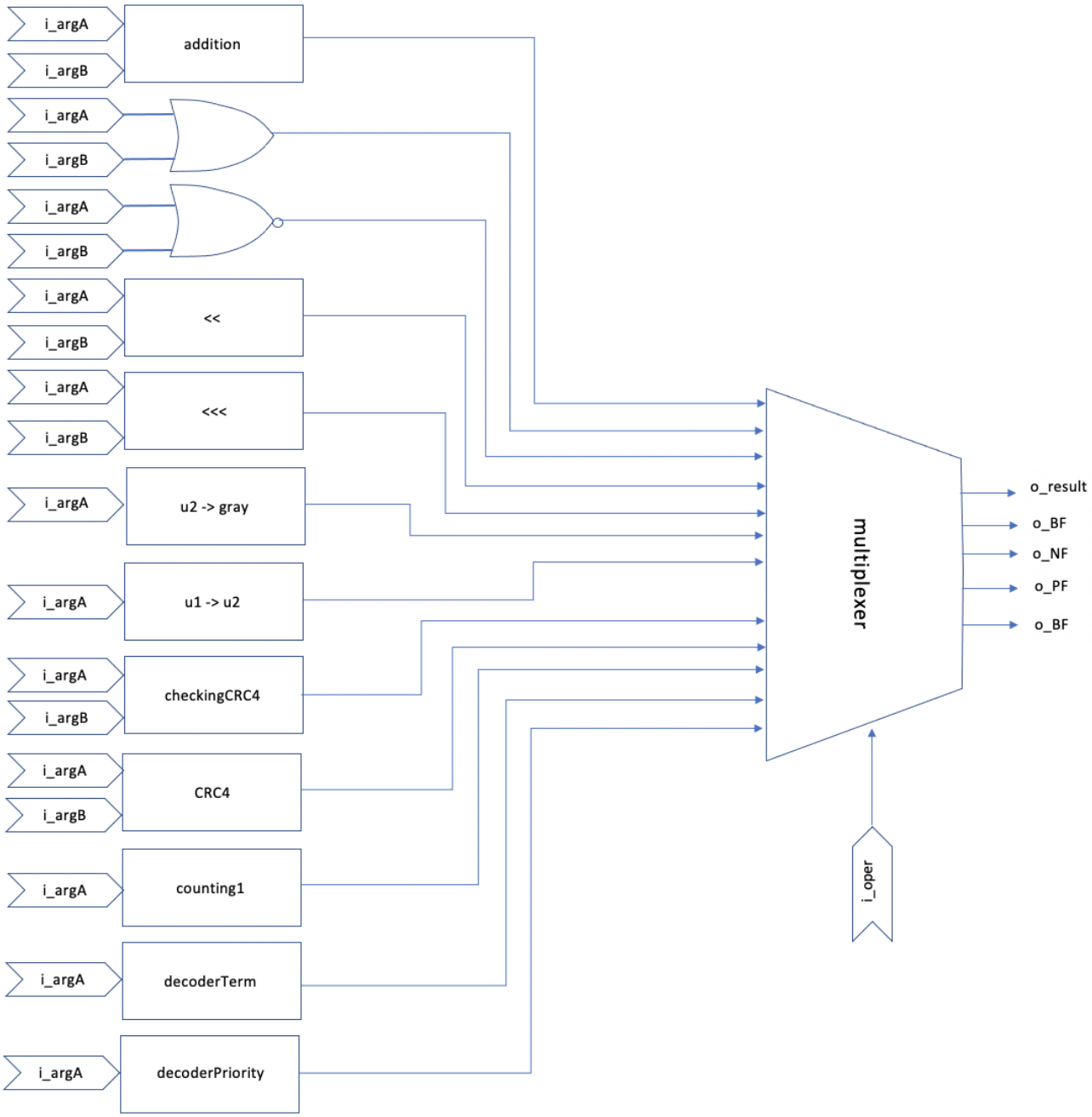

Przedostatnią operacją jest dekoder termometrowy, czyli konwersja termometru do kodu binarnego. Zawiera wejście oraz wyjście. Na wejście i_opera trzeba podać 4’b0111.




Ostatnią operacją jest dekoder priorytetowy, zrealizowany na jednej bramce NAND oraz na dwóch bramkach OR. Zawiera jedno wejście oraz wyjście. Na wejście i_opera trzeba podać 4’b1011.



SCHEMAT BLOKOWY STRUKTURY JEDNOSTKI



FLAGI

W jednostce znajdują się cztery jednobitowe znaczniki wykonanej operacji. Pierwszym z nim jest BF, który informuje o tym, czy w wyniku jest jedna jedynka. Problem w module został rozwiązany za pomocą pętli for, która sprawa ilość jedynek w argumencie wyjściowym i na swoim wyjściu (wyjściu znacznika) pokazuje 1‘b1, jeśli liczba jedynek jest równa 1.

Drugim jest BF, który informuje o tym, czy w wyniku jest jedno zero. Moduł został skonstruowany analogicznie do poprzedniego wskaźnika. Pętla analizuje wyjście i wyświetla wartość wskaźnika 1’b1, gdy jest w nim jedno zero.

Znacznik PF realizuje zadanie uzupełnienia do nieparzystej liczby jedynek. Jest to realizowane za pomocą jednego XORa, który analizuje wynik i gdy w wyniku jest parzysta liczba jedynek, na wyniku ma zero. Jeśli jest nieparzysta to wynikiem jest 1, co stanowi uzupełnienie.

Znacznik NF realizuje uzupełnienie do parzystej liczby jedynek. Jest to zrobione z użyciem XOR, analogicznie do PF, i zaprzeczenia wyniku.

Flagi NF i PF zostały zaimplementowane w module exe_unit, w always_comb.
Natomiast dwa pozostałe podobnie do modułów, dołączając plik do exe_unit.

LISTA ZDEFINIOWANYCH MODUŁÓW – wybrane opisy rozumowania

1. Dodawanie adder.sv
2. Or or_module.sv
3. Nor nor_module.sv
4. Logiczne przesunięcie argumentów w lewo lshift_left.sv
5. Arytmetyczne przesunięcie argumentów w lewo ashift_left.sv
6. Konwersja danej wejściowej z kodu U1 na kod U2 u1intou2.sv

Moduł zawierający konwersje U1 na U2 zawiera pętle if else. Dla liczb dodatnich ciąg bitów jest przepisywany. Kod wygląda tak samo. Dla ujemnych wartości (else) następuje przesunięcie o jeden bit w prawo. Podobnie wyglądała konwersja U2 na kod GRAYA.

7. Konwersja danej wejściowej z kodu U2 na kod GRAY koder_gray.sv
8. Wyznaczenie kodu CRC-4 (0.2 pkt) crc4.sv

Wyznaczanie kodu crc4 polega na dzieleniu (analogicznie do dzielenia wielomianów). Dla 4-bitowego CRC (WPOLY = 4) dzielnik będzie 5-bitowy (WCODE = 5). Jeżeli nad najstarszą pozycją dzielnika jest wartość 0, to przesuwa się dzielnik w prawo o jedną pozycję, aż do napotkania 1, po czym wykonywana jest operacja XOR, co zostało zaimplementowane w kodzie za pomocą pętli for oraz pętli if w for.

9. Sprawdzenie zgodności kodu CRC-4 crc4_check.sv

10. Zliczanie sumarycznej liczby jedynek w obu argumentach wejściowych counting.sv

Zliczanie jedynek w dwóch argumentach odbyło się za pomocą dwóch pętli for, która analizowała wszystkie bity i zliczała jedynki do zmiennej lokalnej.

11. Dekoder termometrowy (thermometer to binary encoder) dekoder_term.sv
12. Dekoder piorytetowy dekoder_prioryt.sv

PROBLEMY

Niestety problemem u mnie była synteza całego exe_unit. Syntezował się tylko z pięcioma pierwszymi modułami oraz ze znacznikami, które nie były w oddzielnym pliku.
Implementację zrobiłam zgodnie z materiałami wykładowymi oraz materiałami, które znalazłam w Internecie.
Nie potrafię znaleźć błędu w rozwiązaniu.
Utworzyłam plik makefile oraz run.ys, w którym zostały wskazane pliki z modułami potrzebnymi do uruchomienia exe_unit.

```
module exe_unit(i_argA, i_argB, i_oper, o_PF, o_result);

    parameter M = 8;
    parameter N = 8;

    input logic signed [N-1:0] i_argA;
    input logic signed [N-1:0] i_argB;
    input logic [7:0]          i_oper;
    output logic [M-1:0]       o_result;
    output logic o_PF;

    logic [M-1:0] s_koder_gray, s_u1intou2, s_dekoder_term, s_counting1, s_crc4_check, s_crc4, s_dekoder_prioryt;

    koder_gray#(.LEN(N)) gray_koder(.i_argA(i_argA), .o_result(s_koder_gray));
    u1intou2#(.BITS(N)) u1intou2(.i_argA(i_argA), .o_result(s_u1intou2));
    dekoder_term#(.LEN(N)) decoder_therm(.i_argA(i_argA), .o_result(s_decoder_therm));
    counting1#(.LEN(N)) decoder_therm(.i_argA(i_argA), .o_result(s_counting1));
    crc4_check#(.WCODE(N)) crc4_check(.i_data(i_argA), .i_poly(i_argB), .o_crc(s_crc4_check));
    crc4#(.WCODE(5), .WPOLY(4)) crc_4(.i_data(i_argA), .i_poly(i_argB[3:0]), .i_crc(i_argB[4'b0]), .o_crc(s_crc4));
    dekoder_prioryt#(.BITS(N)) dekoder_priorytetowy(.i_argA(i_argA), .o_result(s_decoder_prioryt));
```

makefile ●

makefile

```
1   SYNTH = yosys
2
3   rtl:
4       ${SYNTH} -s run.yosys
5
6   run:
7       iverilog -g2012 ./WORK/exe_unit.sv tb.sv.WORK/a.out
8
9   run_rtl:
10      iverilog -g2005-sv ./RTL/exe_unit_rtl.sv tb.sv./RTL/a.out
```

run.ys ×

run.ys

```
1   read_verilog -sv exe_unit.sv
2   read_verilog -sv ./WORK/koder_gray.sv
3   read_verilog -sv ./WORK/u1intou2.sv
4   read_verilog -sv ./WORK/dekoder_term.sv
5   read_verilog -sv ./WORK/counting1.sv
6   read_verilog -sv ./WORK/crc4_check.sv
7   read_verilog -sv ./WORK/crc4.sv
8   read_verilog -sv ./WORK/dekoder_prioryt.sv
9
10  synth
11
12  abc -g AND,OR,XOR
13  opt_clean -purge
14
15  show exe_unit
16
17  write_verilog -noattr exe_unit_rtl.sv
```