Układ generujący ciąg Fibonacciego

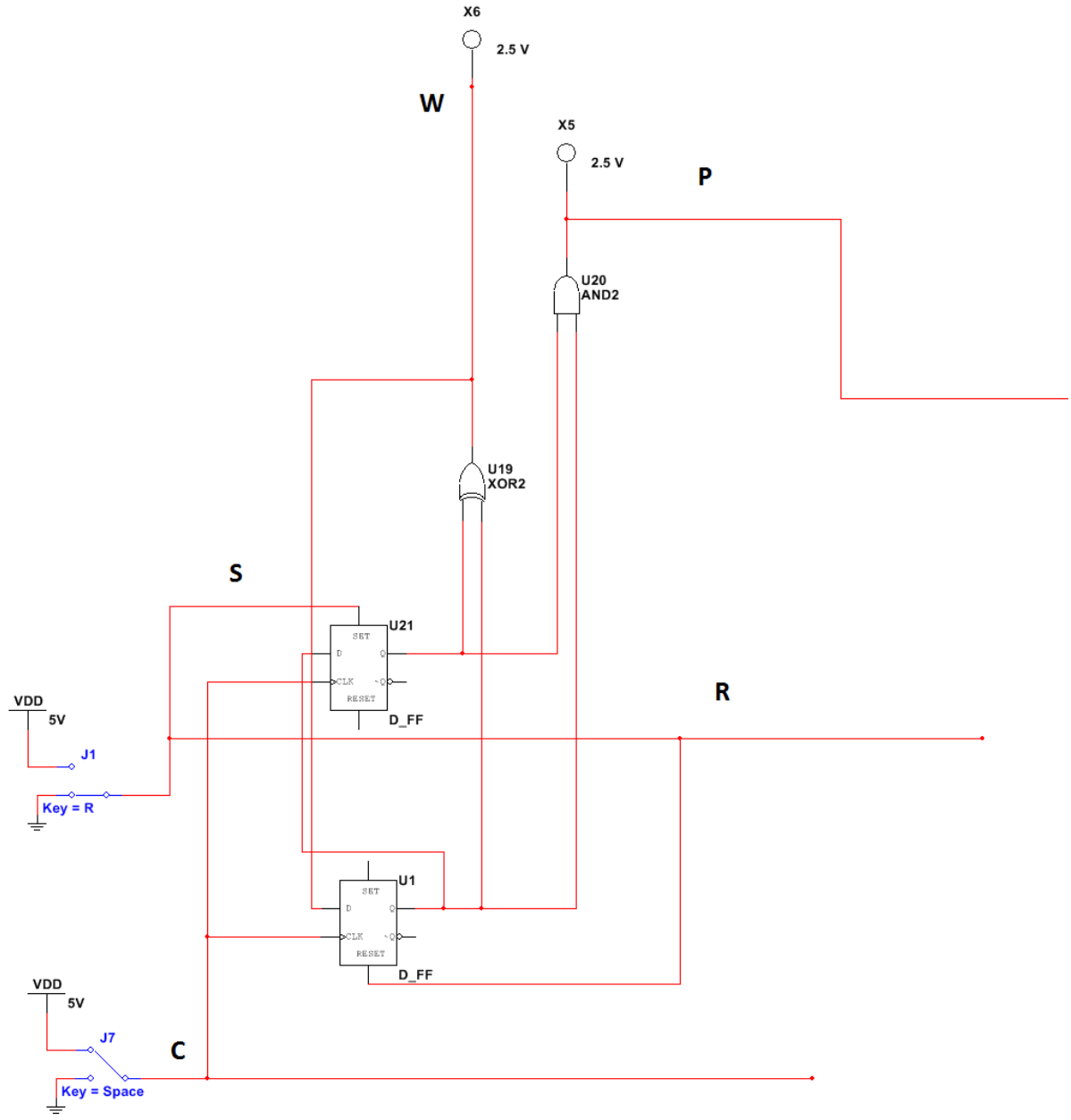
Projekt składa się z dwóch części:

1) Zaprojektowanie układu wyświetlającego kolejne liczby Fibonacciego w programie Multisim.
2) Zaprojektowanie tego samego układu w języku Verilog.

Projekt przedstawia rozwiązanie ogólne generowania ciągu Fibonacciego.

Projekt w programie Multisim:

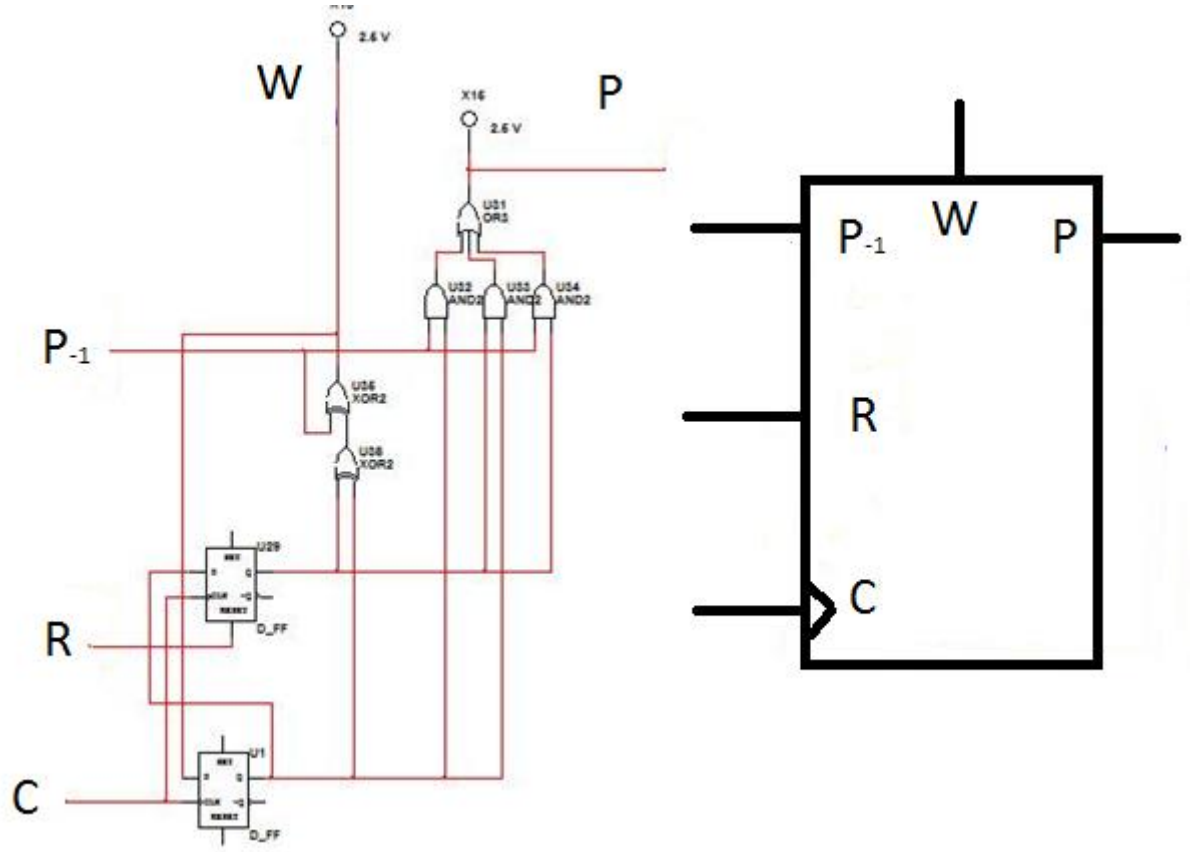
Rysunek 1 przedstawia początek układu. Jest on specyficzny, gdyż nie posiada poprzedniego bitu przeniesienia. Działa na zasadzie półsumatora.



Rysunek 1.

W-wynik, S-set (ustawienie na początku programu, żeby układ „ruszył”), C-clock, R-reset oraz P-przeniesienie.

Na rysunku 2 przedstawiliśmy „cegiełkę” z jakiej składa się cały układ. Użyliśmy razem szesnastu takich części do wyświetlania kolejnych liczb Fibonacciego na czterech wyświetlaczach.



Rysunek 2.

P_{-1} jest bitem przeniesienia z poprzedniego sumowania. W to wynik sumowania, a P to aktualne przeniesienie. R-reset, C-clock. Układ składa się z przerzutników typu D, bramek XOR i bramek AND. Układ ten działa tak jak algorytm na kolejne liczby Fibonacciego. Na kilku rejestrach przechowujemy wartość Fib_{n-1}, na kolejnych wartość Fib_{n-2}. Po cyklu zegara, zamiast Fib_{n-1} ustawiamy wartość $Fib_{n-1}+ Fib_{n-2}$. Cztery wyświetlacze są podłączone do odpowiednio $W_1,\ldots,W_{15}$ oraz do ostatniego bitu przeniesienia. Liczby Fibonacciego są wyświetlane w kodzie szesnastkowym.

Tabela prawdy dla P:

P_{-1}	Q_1Q_0	00	01	11	10
0		0	0	1	0
1		0	1	1	1

Funkcja P:

$P=Q_0P_{-1}+Q_1Q_0+Q_1P$

P-bieżące przeniesienie

P_{-1}-poprzednie przeniesienie

Q_1-górny bit

Q_0-dolny bit

Tabela prawdy dla W:

P_{-1}	Q_1Q_0	00	01	11	10
0		0	1	0	1
1		1	0	1	0

Funkcja W:
$W=P_{-1}xorQ1xorQ0$
W-wynik
P-bieżące przeniesienie
P_{-1}-poprzednie przeniesienie
Q_1-górny bit
Q_0-dolny bit

Schemat układu jest dołączony do sprawozdania.

Obsługa:

Należy użyć przełącznika „R”(użyć klawisza „R” dwa razy), żeby rozpocząć wyświetlanie liczb. Następnie wystarczy używać przełącznika „C”(można używać spacji), żeby wyświetlać kolejne liczby. Do resetowania użyć przełącznika pod klawiszem „R”.

Projekt w języku Verilog:

Moduł „dek7seg” służy za dekodowanie wyświetlacza 7 segmentowego. Dzięki temu możliwe będzie wypisywanie odpowiednich cyfr na wyświetlaczu.

```
module dek7seg (
    input [3:0] data_in ,
    output reg [6:0] seg
);

always@( data_in )
begin
    case ( data_in )
            4'h0 : seg <= 7'b0111111;
            4'h1 : seg <= 7'b0000110;
            4'h2 : seg <= 7'b1011011;
            4'h3 : seg <= 7'b1001111;
            4'h4 : seg <= 7'b1100110;
            4'h5 : seg <= 7'b1101101;
            4'h6 : seg <= 7'b1111101;
            4'h7 : seg <= 7'b0000111;
            4'h8 : seg <= 7'b1111111;
            4'h9 : seg <= 7'b1101111;
            4'hA : seg <= 7'b1110111;
            4'hB : seg <= 7'b1111100;
            4'hC : seg <= 7'b0111001;
            4'hD : seg <= 7'b1011110;
            4'hE : seg <= 7'b1111001;
```

```
            4'hF : seg <= 7'b1110001;
    endcase
end

endmodule
```

W kolejnym module „regD" jest określone przypisywanie do rejestru jedynki albo zera reagujące na każde narastające zbocze odpowiednio sygnału clock, reset oraz set. Reset powoduje ustawienie zera, set-ustawienie jedynki. W innym przypadku zostaje zapisana odpowiednia wartość do rejestru q.

```
module regD( clk, reset, set, d, q);
 input clk, reset, d,set;
 output q;

 reg   q;
 wire clk, reset, d;

 always @ (posedge clk or posedge reset or posedge set)
 if (reset) begin
  q <= 1'b0;
 end
else begin
    if (set) begin
            q <= 1'b1;
    end
    else begin
            q <= d;
    end
end

endmodule
```

W module „start" mamy rejestry: sum-suma oraz carry-przeniesienie. Będziemy w tych rejestrach zapisywać wynik dodawania Q1 i Q0. Sumowanie reaguje na narastające zbocze zegara. Moduł jest wykonywany na starcie programu.

```
module start(
    input clock,
    input set,
    output reg sum,
    output reg carry
);

wire outQ1, outQ0;
```

```
regD Q1(.clk(clock), .set(set),.d(outQ0), .q(outQ1));
regD Q0(.clk(clock), .reset(set), .d(outQ0 ^ outQ1), .q(outQ0));

always@(posedge clock)
begin
    sum <= outQ1 ^ outQ0;
    carry <= outQ1 & outQ0;

end

endmodule
```

W module „tile” określamy już kolejne liczby Fibonacciego. Uwzględniamy tu poprzednie przeniesienie i sumujemy podobnie jak w module „start”.

```
module tile(
    output reg sum,
    output reg carry,
    input reset,
    input clk,
    input prevCarry
);

wire outQ0, outQ1;

regD Q1(.clk(clk), .reset(reset), .d(outQ0), .q(outQ1) );
regD Q0(.clk(clk), .reset(reset), .d(sum), .q(outQ0));

always@(posedge clk or posedge prevCarry) // to prevent signal prevCarry from delaying
begin
    sum <= outQ1 ^ outQ0 ^ prevCarry;
    carry <= (prevCarry & outQ0) | (outQ1 & outQ0) | (prevCarry & outQ1);
end

endmodule
```

Główny moduł „UP2_TOP” łączy wszystkie moduły ze sobą.
Kod tego modułu został przedstawiony na kolejnych dwóch stronach:

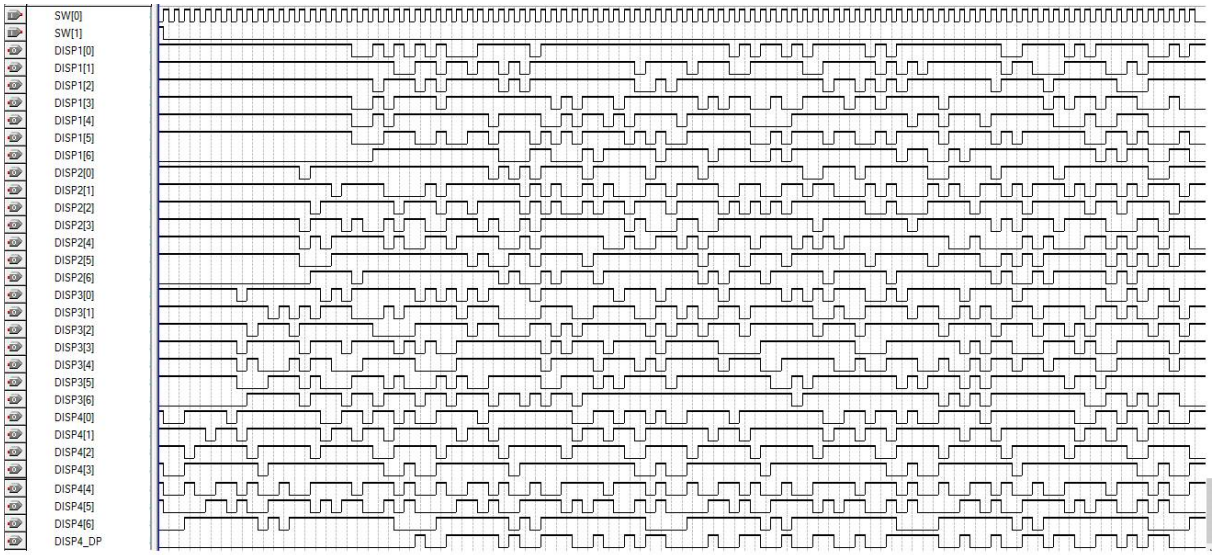
```
//Cz. 1
module UP2_TOP(
        input [1:0] SW,
        output [6:0] DISP1,
        output [6:0] DISP2,
        output [6:0] DISP3,
        output [6:0] DISP4,
        output DISP4_DP
);
wire carry0,sum0;

start s1(
        .clock(SW[0]),
        .set(SW[1]),
        .sum(sum0),
        .carry(carry0)
);

wire carry1,sum1;

tile t1(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry0),
        .sum(sum1),
        .carry(carry1),
);

wire carry2,sum2;

tile t2(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry1),
        .sum(sum2),
        .carry(carry2),
);

wire carry3,sum3;

tile t3(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry2),
        .sum(sum3),
        .carry(carry3),
);

dek7seg d1(
        .data_in({sum3,sum2,sum1,sum0}),
        .seg(DISP4[6:0])
);
```

```
//Cz. 2
wire carry4, sum4;

tile t4(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry3),
        .sum(sum4),
        .carry(carry4),
);

wire carry5,sum5;

tile t5(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry4),
        .sum(sum5),
        .carry(carry5),
);

wire carry6,sum6;

tile t6(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry5),
        .sum(sum6),
        .carry(carry6),
);

wire carry7,sum7;

tile t7(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry6),
        .sum(sum7),
        .carry(carry7),
);

dek7seg d2(
        .data_in({sum7,sum6,sum5,sum4}),
        .seg(DISP3[6:0])
);
```

```
//Cz. 3
wire carry8,sum8;

tile t8(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry7),
        .sum(sum8),
        .carry(carry8),
);

wire carry9,sum9;

tile t9(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry8),
        .sum(sum9),
        .carry(carry9),
);

wire carry10,sum10;

tile t10(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry9),
        .sum(sum10),
        .carry(carry10),
);

wire carry11,sum11;

tile t11(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry10),
        .sum(sum11),
        .carry(carry11),
);

dek7seg d3(
        .data_in({sum11,sum10,sum9,sum8})
,
        .seg(DISP2[6:0])
);
```

```
//Cz. 4
wire carry12,sum12;

tile t12(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry11),
        .sum(sum12),
        .carry(carry12),
);

wire carry13,sum13;

tile t13(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry12),
        .sum(sum13),
        .carry(carry13),
);

wire carry14,sum14;

tile t14(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry13),
        .sum(sum14),
        .carry(carry14),
);

wire sum15;

tile t15(
        .clk(SW[0]),
        .reset(SW[1]),
        .prevCarry(carry14),
        .sum(sum15),
        .carry(DISP4_DP),
);

dek7seg d4(
        .data_in({sum15,sum14,sum13,sum12})
,
        .seg(DISP1[6:0])
);

endmodule
```

Wyniki symulacji:



SW[0]-przełącznik służący za clock.
SW[1]-reset
DISP4_DP-lampka zapalająca się, gdy liczby Fibonacciego wyszły poza zakres wyświetlaczy.
DISP1[0]-odpowiada segmentowi „a” w wyświetlaczu nr 1.
DISP1[1]-odpowiada segmentowi „b” w wyświetlaczu nr 1.
DISP1[2]-odpowiada segmentowi „c” w wyświetlaczu nr 1.
DISP1[3]-odpowiada segmentowi „d” w wyświetlaczu nr 1.
DISP1[4]-odpowiada segmentowi „e” w wyświetlaczu nr 1.
DISP1[5]-odpowiada segmentowi „f” w wyświetlaczu nr 1.
DISP1[6]-odpowiada segmentowi „g” w wyświetlaczu nr 1.
Podobnie z wyświetlaczami nr 2, 3 i 4.
Obsługa:

Po wciśnięciu start w Program Device należy używać przełącznika SW[0]-pierwszy switch po prawej. Pełne przełączenie(powrót do ustawienia początkowego przełącznika) generuje jedną liczbę z ciągu Fibonacciego. Do użycia resetu należy użyć SW[1]-drugi switch licząc od prawej strony. Jeśli zapali się kropka przepełnienia na wyświetlaczu to oznacza, że wyszliśmy poza zakres i trzeba użyć reset, żeby wyświetlać ciąg od początku.