

# *Projekt: 1-bitowy sumator na FPGA*

## **1. Wprowadzenie**

W tym projekcie tworzymy 1-bitowy sumator na FPGA przy użyciu języka opisu sprzętu Verilog. Sumator ten przyjmuje dwa bity wejściowe oraz bit przeniesienia, wykonuje dodawanie i przekazuje wynik (sumę i bit przeniesienia) na wyjście.

# Tworzenie kodu Verilog dla 1-bitowego sumatora

Zaczynamy od stworzenia kodu Verilog dla 1-bitowego sumatora. Oto przykładowy kod, który należy zapisać w pliku sum.v

```

module full_adder (
input wire a,          // Wejście A (przycisk) - pierwszy bit wejściowy
input wire b,          // Wejście B (przycisk) - drugi bit wejściowy
input wire carry_in,  // Wejście przeniesienia (przycisk) - bit przeniesienia z poprzedniego etapu
output wire sum,      // Wyjście suma (diody) - wynik dodawania wejść a, b i carry_in
output wire carry_out, // Wyjście przeniesienia (diody) - wynik przeniesienia do następnego etapu
output wire l3,       // L3 dla B - pokazuje stan wejścia B
output wire l4,       // L4 dla carry_in - pokazuje stan wejścia przeniesienia
output wire l5        // L5 dla A - pokazuje stan wejścia A
);

// Obliczenie sumy przy użyciu operacji XOR
assign sum = a ^ b ^ carry_in;
// Obliczenie przeniesienia przy użyciu operacji AND i OR
assign carry_out = (a & b) | (carry_in & (a ^ b));

// Przypisanie sygnału l5 do stanu wejścia A
assign l5 = a;

// Przypisanie sygnału l3 do stanu wejścia B
assign l3 = b;

// Przypisanie sygnału l4 do stanu wejścia przeniesienia
assign l4 = carry_in;

endmodule

```

wejścia i wyjścia: a, b, carry\_in to wejścia (przyciski), a sum, carry\_out to wyjścia (diody). Dodatkowe wyjścia l3, l4 i l5 służą do wizualizacji stanów wejść A, B i carry\_in.  
operacja XOR:  $sum = a \oplus b \oplus carry\_in$ ; wykonuje dodawanie bitów z uwzględnieniem przeniesienia.  
operacja AND i OR:  $carry\_out = (a \& b) \vee (carry\_in \& (a \oplus b))$ ; oblicza przeniesienie, które będzie przeniesione do kolejnego etapu dodawania.

# Tabela Prawdy dla 1-Bitowego Sumatora Pełnego

a	b	carry_in	sum	carry_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

a i b to dwa wejściowe bity do dodania.

**przeniesienie wejściowe** to bit przeniesienia z poprzedniego niższego bitu (jeśli dotyczy).

**suma** to wynik dodawania a, b i **przeniesienia wejściowego** (operacja XOR).

**przeniesienie wyjściowe** to bit przeniesienia wygenerowany podczas dodawania, który jest przekazywany do następnego wyższego bitu (operacje AND i OR).

## Jak to działa:

- Gdy **a = 0**, **b = 0** i **przeniesienie wejściowe = 0**, suma wynosi 0 i nie ma przeniesienia, więc **przeniesienie wyjściowe = 0**.
- Gdy **a = 0**, **b = 1** i **przeniesienie wejściowe = 0**, suma wynosi 1 i nie ma przeniesienia, więc **przeniesienie wyjściowe = 0**.
- Gdy **a = 1**, **b = 1** i **przeniesienie wejściowe = 1**, suma wynosi 1 i jest przeniesienie, więc **przeniesienie wyjściowe = 1**.

# Tworzenie pliku konfiguracji pinów (PCF)

Aby skonfigurować piny na FPGA, tworzymy plik konfiguracji pinów (PCF). Oto przykład, jak może wyglądać ten plik:

```
set_io a 12      # Przycisk A (Button 3)
set_io b 11      # Przycisk B (Button 2)
set_io carry_in 10 # Przycisk Carry-in (Button 1)

set_io sum 39    # Wyjście suma na diodzie L1
set_io carry_out 40 # Wyjście carry-out na diodzie L2
set_io l3 41     # Dioda L3 dla wejścia B
set_io l4 42     # Dioda L4 dla carry_in
set_io l5 37     # Dioda L5 dla wejścia A
```

# Synteza z pomocą Yosys

*Z pomocą narzędzia **Yosys** syntezujemy nasz kod Verilog dla FPGA. Aby to zrobić, wykonujemy polecenie:*

```
yosys -p "synth_ice40 -top full_adder -json sum.json" sum.v
```

# Rozmieszczanie i trasowanie z pomocą nextpnr

Następnym krokiem jest rozmieszczanie i trasowanie na FPGA z użyciem narzędzia **nextpnr**. Aby to zrobić, wykonujemy polecenie:

```
nextpnr-ice40 --up5k --package sg48 --pcf sum.pcf --json sum.json --asc
```

To polecenie rozmieszcza elementy na FPGA i wykonuje trasowanie połączeń.

# Tworzenie pliku binarnego za pomocą icепack

Teraz musimy stworzyć plik binarny do załadowania na FPGA. Robimy to za pomocą narzędzia **icепack**:

```
icепack sum.asc sum.bin
```

To polecenie konwertuje plik do formatu, który można załadować na FPGA.



# Programowanie FPGA za pomocą icoprogram

Aby załadować gotowy plik binarny na FPGA, używamy polecenia **icoprogram**:

```
icoprogram sum.bin
```

To polecenie ładuje plik binarny na naszą płytę FPGA.