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

```
odule 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 (dioda) - wynik dodawania wejść a, b i carry_in
   output wire carry_out, // Wyjście przeniesienia (dioda) - 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;

ndmodule
```

ejścia 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.
peracja XOR: sum = a ^ b ^ carry_in; wykonuje dodawanie bitów z uwzględnieniem przeniesienia.
peracja AND i OR: carry_out = (a & b) | (carry_in & (a ^ b)); oblicza przeniesienie, które będzie przeniesione do kolejnego etapu dodawania.

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
```

Syntezowanie 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 icepack

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

```
icepack sum.asc sum.bin
```

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

Programowanie FPGA za pomocą iceprog

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

```
iceprog sum.bin
```

To polecenie ładuje plik binarny na naszą płytę FPGA.