

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA

Architektura komputerów 2
Projekt

Implementacja fizyczna układów cyfrowych

AUTORZY:

Adam Jędryka 249443 czwartek TN 18:55 (AK2)
Jakub Pawleniak 248897 środa TP 18:55 (OiAK)

PROWADZĄCY:

Dr inż. Piotr Patronik

Spis treści

1. Wstęp	3
1.1. Cel projektu	3
1.2. Układ full-adder'a	3
2. Verilog	4
2.1. Moduł full adder'a.	4
2.2. Test bench	4
3. Synteza Yosys	6
3.1. Narzędzie	6
3.2. Synteza gate level	6
4. Synteza Qflow	8
4.1. Narzędzie	8
4.2. Przeprowadzenie syntezy	9
4.3. Reprezentacja bramek w układzie fizycznym	9
5. Symulacja cyfrowa	12
5.1. Symulacja bez opóźnień czasowych	12
5.2. Symulacja uwzględniająca opóźnienia czasowe	13
6. Podsumowanie	14
Literatura	15

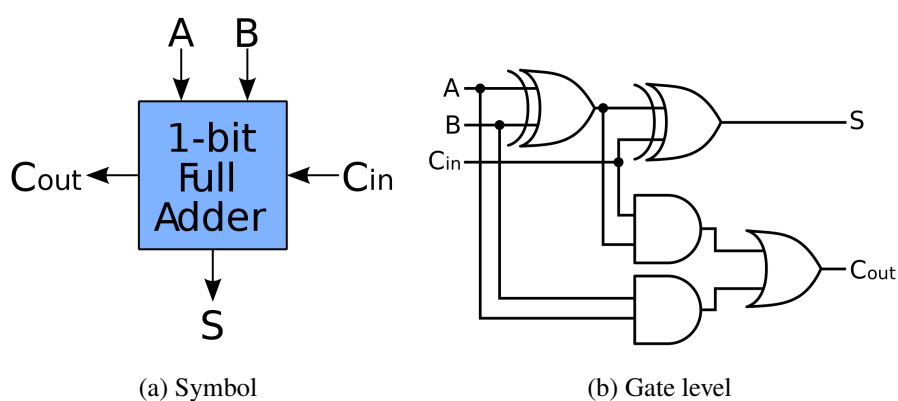
Rozdział 1

Wstęp

1.1. Cel projektu

Celem ogólnym projektu była synteza logiczna i fizyczna modułu Full-Addera z wykorzystaniem narzędzi **Yosys**[1] oraz **Qflow**[2].

1.2. Układ full-adder'a



Rys. 1.1: Full-adder

Powyższy schemat przedstawia układ full-adder'a, który realizuje dodawanie dwóch liczb dwójkowych. Układ wykorzystuje 2 bramki XOR, 2 bramki AND oraz 1 bramka OR. Działanie układu można przedstawić za pomocą wyrażenia boolowskiego:

$$o_sum = i_carry \oplus i_x \oplus i_y$$

$$o_carry = i_x * i_y + i_carry(i_x \oplus i_y)$$

Rozdział 2

Verilog

2.1. Moduł full adder'a.

Poniższy moduł zapisany w języku Verilog realizuje działanie układu Full-adder'a zaprezentowanego w poprzednim rozdziale.

Listing 2.1: Plik *full_adder.v*.

```
1 module full_adder
2 (
3     input wire i_x, i_y, i_carry,
4     output wire o_sum, o_carry
5 );
6
7     assign o_sum = i_x ^ i_y ^ i_carry;
8     assign o_carry = i_x & i_y | (i_x ^ i_y) & i_carry;
9
10 endmodule
```

2.2. Test bench

Poniżej zaprezentowany został test bench modułu full adder'a, testujący wszystkie możliwe kombinacje sygnałów wejściowych w odstępach co 20ns.

Listing 2.2: Plik *full_adder_tb.v*.

```
11 'include "full_adder.v"
12 'timescale 1 ns/10 ps // time-unit = 1 ns, precision = 10 ps
13
14 module full_adder_tb;
15
16     reg x, y, c;
17     wire sum, o_carry;
18
19     localparam period = 20;
20
21     full_adder adder_inst (
22         .i_x(x),
23         .i_y(y),
24         .i_carry(c),
25         .o_sum(sum),
26         .o_carry(o_carry)
27     );
```

```
28
29  initial
30      begin
31          $dumpfile("test_bench.vcd");
32          $dumpvars(1, full_adder_tb);
33          x = 0; y = 0; c = 0; #period;
34          x = 0; y = 0; c = 1; #period;
35          x = 0; y = 1; c = 0; #period;
36          x = 0; y = 1; c = 1; #period;
37          x = 1; y = 0; c = 0; #period;
38          x = 1; y = 0; c = 1; #period;
39          x = 1; y = 1; c = 0; #period;
40          x = 1; y = 1; c = 1; #period;
41      end
42  endmodule
```

Rozdział 3

Synteza Yosys

3.1. Narzędzie

Yosys [1] to narzędzie *open-source* służące do obsługi syntezy języka Verilog HDL. Jest szeroko używany jako narzędzie, za pomocą którego można wykonywać zadania w dziedzinie syntezy behawioralnej, syntezy RTL oraz syntezy logicznej. W projekcie wykorzystana została wersja yosys-0.9.

3.2. Synteza gate level

W ramach projektu wykonana została synteza na poziomie bramek przygotowanego modułu full-addera, w wyniku której powinien powstać program o strukturze płaskiej (*flatten*), czyli program składający się wyłącznie z bramek logicznych. W celu optymalizacji struktury, do skryptu dodaliśmy zapis wywołujący optymalizację na poziomie wysokim oraz niskim. W celu realizacji tego rodzaju syntezy niezbędne okazało się dołączenie dodatkowej biblioteki **cmos_cells.lib**[3]. W skrypcie znalazła się również komenda odpowiedzialna za podział sieci wielobitowych (*split-nets*). Końcowy fragment skryptu odpowiada za zapis zsyntezowanego modułu, a następnie zapisaniu reprezentacji graficznej logiki układu do pliku `.dot`.

Listing 3.1: Plik *full_adder.ys*.

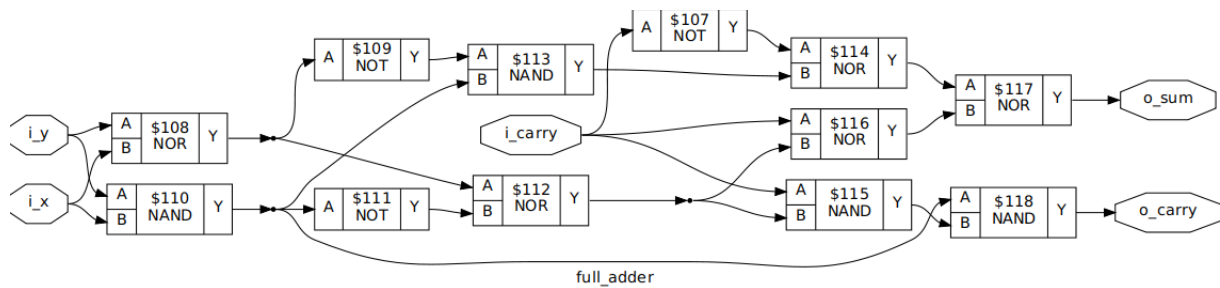
```
44 # read design
45 read_verilog ../verilog/full_adder.v
46 hierarchy -check
47
48 synth -flatten
49
50 # high-level synthesis
51 proc; opt; memory; opt; fsm; opt
52
53 # low-level synthesis
54 techmap; opt
55
56 # map to target architecture
57 read_liberty -lib ../libs/cmos_cells.lib
58 dfflibmap -liberty ../libs/cmos_cells.lib
59 abc -liberty ../libs/cmos_cells.lib
60
61 # split larger signals
62 splitnets -ports; opt
63
64 # cleanup
```

```

65 clean
66
67 # write synthesized design
68 write_verilog ./build/full_adder_synth.v
69
70 # show
71 show -format dot -lib ./build/full_adder_synth.v -prefix ./build/
    ↪ full_adder_cmos

```

Do struktury zsyntezowanego modułu należy zaimportować plik **cmos_cells.v** zawierający moduły bramek użytych w syntezy w pliku.



Rys. 3.1: Plik *full_adder_cmos.dot*

Rozdział 4

Synteza Qflow

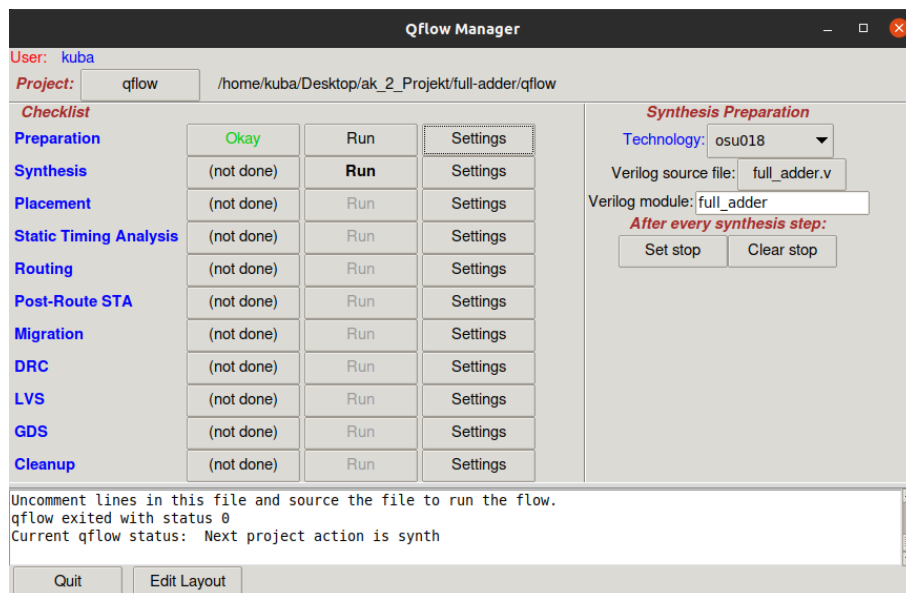
4.1. Narzędzie

Qflow to kompletne narzędzie używane do syntezy obwodów cyfrowych. Na jego działanie składa się szereg komponentów działających na zasadzie *open-source*. Należą do nich m.in.:

- yosys,
- graywolf,
- qrouter,
- magic.

Proces syntezy ułatwia przystępne GUI, dzięki któremu proces jest przejrzysty i prosty w wykonaniu.

W projekcie wykorzystujemy Qflow w wersji 1.3.17.

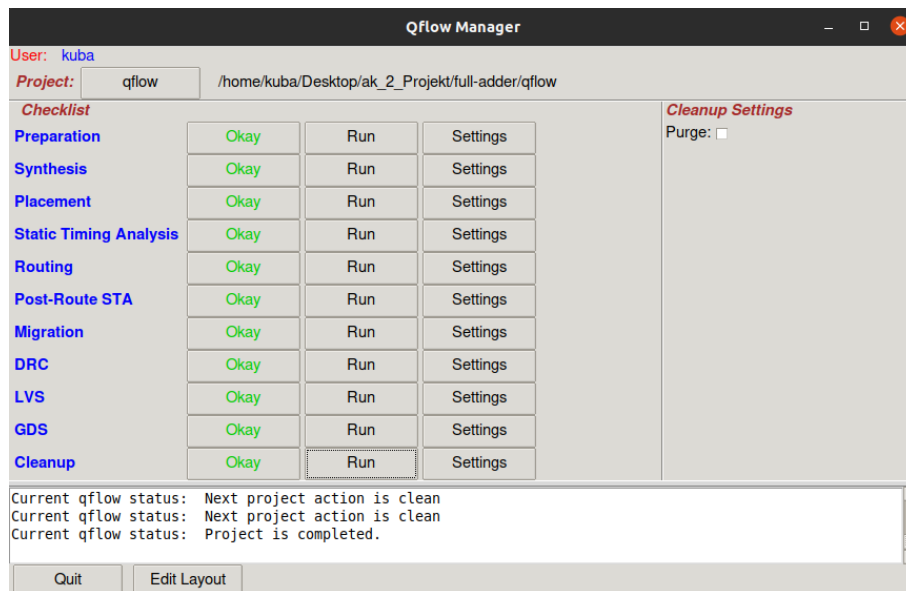


Rys. 4.1: Zrzut ekranu GUI Qflow.

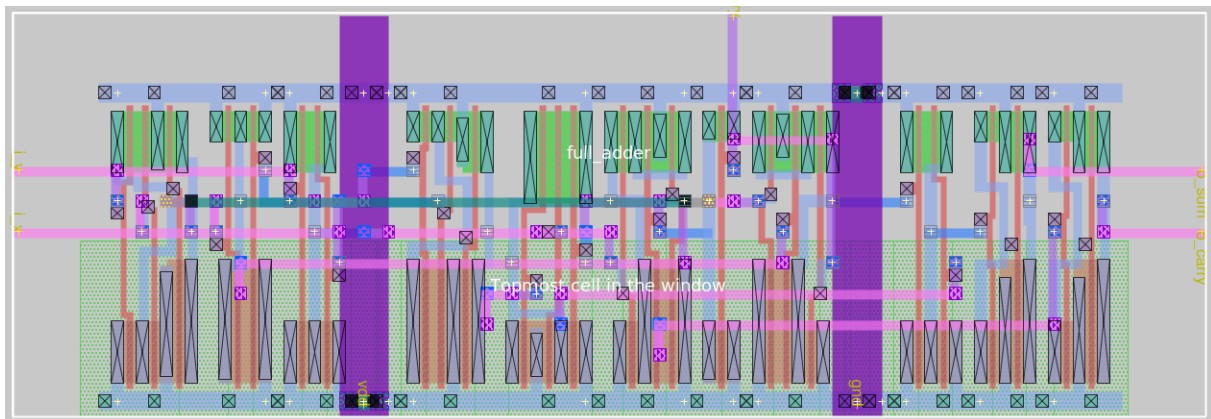
Wadą użycia GUI jest brak dostępu do wszystkich możliwości narzędzia. Zaawansowane opcje dostępne są poprzez linię komend.

4.2. Przeprowadzenie syntezy

Cały proces syntezy podzielony jest na podprocesy, których status możemy śledzić z poziomu GUI, a szczegółowe informacje można odczytać z plików wyjściowych podprocesów utworzonych w folderze projektu. Pierwszym z etapów jest *Preparation*, w którym wybierany jest moduł (w naszym przypadku *full_adder.v*) oraz technologię docelową. Domyślną technologią jest *osu018* i właśnie z niej skorzystaliśmy w procesie syntezy. Domyślnie wszystkie etapy po procesie *Preparation* wykonują się bez zatrzymania (jeżeli nie napotkamy po drodze żadnego problemu). Po rozpoczęciu syntezy przyciskiem *Run* wykonuje się etap po etapie.



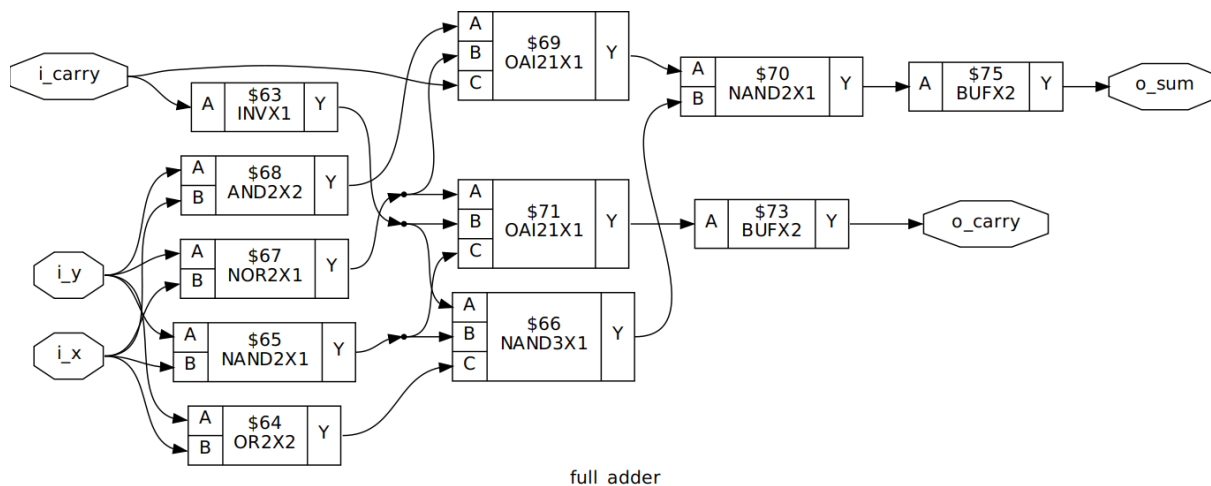
Rys. 4.2: Widok programu Qflow Manager po poprawnie zakończonej syntezie.



Rys. 4.3: Schemat fizyczny układu powstały w wyniku syntezy.

4.3. Reprezentacja bramek w układzie fizycznym

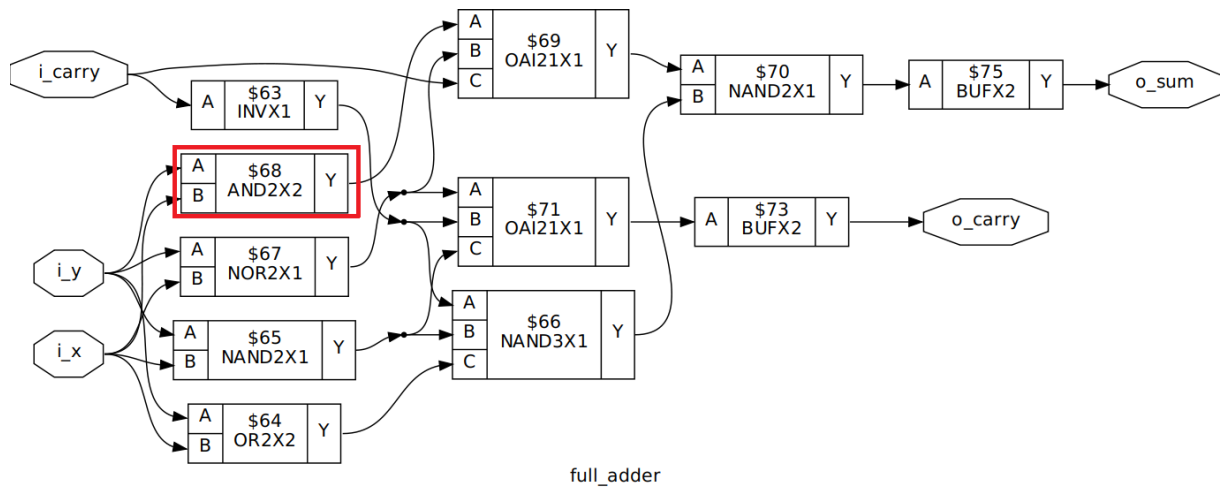
Z racji korzystania z technologii *osu18* [5] w programie Qflow, ponownie wygenerowaliśmy plik schematu logicznego na poziomie bramek korzystając tym razem z domyślnego skryptu *yosys* wygenerowanego w Qflow dla tej właśnie technologii.

Rys. 4.4: Plik *full_adder_osu18.dot*Listing 4.1: Fragment pliku *full_adder.v* po syntezie.

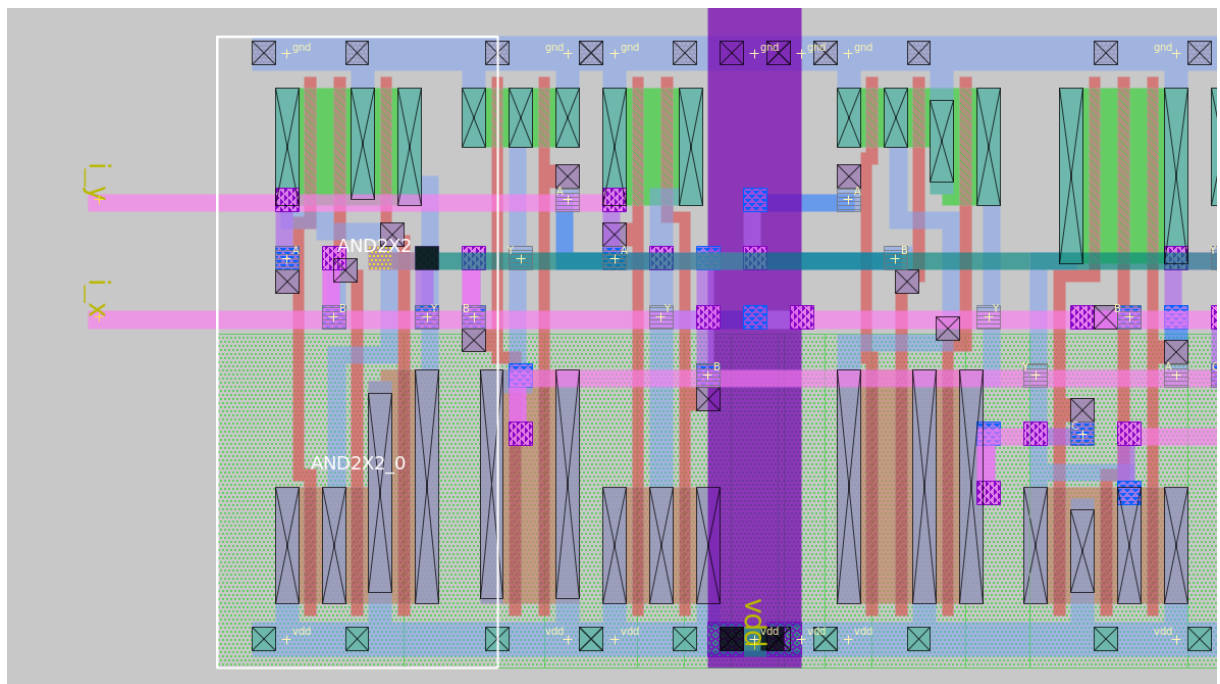
```

73 /* Verilog module written by vlog2Verilog (qflow) */
74
75 module full_adder(
76     input i_carry,
77     input i_x,
78     input i_y,
79     output o_carry,
80     output o_sum
81 );
82
83 wire vdd = 1'b1;
84 wire gnd = 1'b0;
85
86 wire _7_ ;
87 wire i_x ;
88 wire i_y ;
89 wire _4_ ;
90 wire _1_ ;
91 wire o_sum ;
92 wire i_carry ;
93 wire _6_ ;
94 wire _3_ ;
95 wire _0_ ;
96 wire o_carry ;
97 wire _8_ ;
98 wire _5_ ;
99 wire _2_ ;
100
101 /***/
102
103 AND2X2 _14_ (
104     .A(i_y),
105     .B(i_x),
106     .Y(_1_)
107 );
108
109 /***/
110
111 endmodule

```



(a) Przykładowa bramka AND



(b) Fragment układu odpowiedzialny za bramkę AND

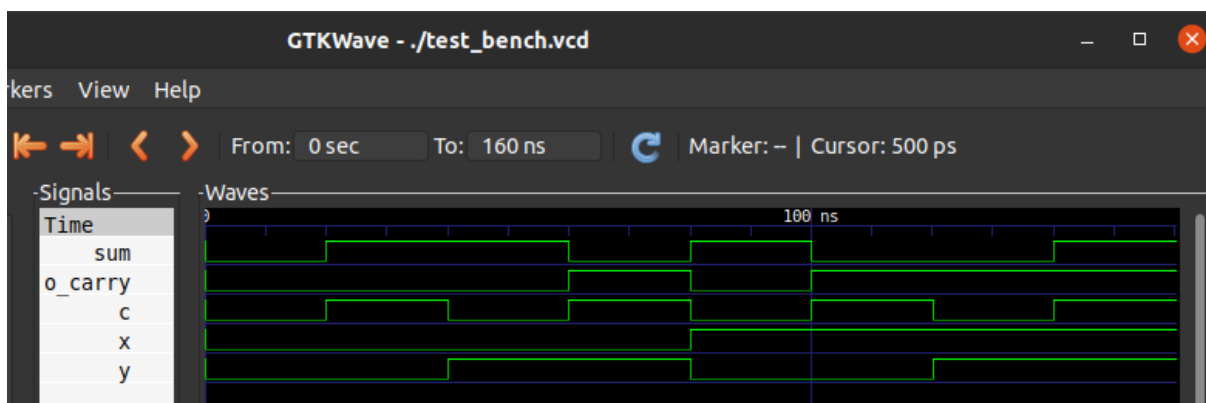
Rys. 4.5: Implementacja bramki AND

Jak możemy zauważyć wygenerowany plik schematu logicznego na poziomie bramek odpowiada wygenerowanemu układowi fizycznemu. Na zdjęciu A zaznaczono bramkę AND oraz na zdjęciu B zaznaczono jej odpowiednik w układzie fizycznym.

Rozdział 5

Symulacja cyfrowa

5.1. Symulacja bez opóźnień czasowych



Rys. 5.1: Symulacja modułu full-adder'a bez uwzględnienia opóźnień czasowych

Input			Output	
x	y	carry	sum	carry
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

Tab. 5.1: Tabela prawdy dla prawidłowego układu full-adder'a

Powyższy zrzut ekranu z przebiegów sygnałów w czasie poprawnie pokrywa się z oczekiwaniami zapisanymi w tabeli prawdy dla tego samego układu.

5.2. Symulacja uwzględniająca opóźnienia czasowe

Dla potrzeby wykonania takiej symulacji wymagany jest PDK (process design kit), czyli zestaw plików opisujących modele fizyczne np. bramek wraz z opisami opóźnień na bramkach. Następnie w procesie Post-Route STA należy wygenerować plik .sdf opisujący m.in. opóźnienia na ścieżkach oraz bramkach. Przy pomocy tego plik oraz pliku modułu verilog'owego po syntezie można przeprowadzić testbench'a, który to w efekcie pozwoli nam na zarejestrowanie zmian sygnałów w czasie uwzględniająca opóźnienia czasowe. Następnie będzie w stanie wyświetlić to na wykresie w czasie.

Symulację tą próbowaliśmy uzyskać modyfikując procesy programu Qflow. Ręczne użycie komend do poszczególnych procesów mogłoby być bardzo czasochłonne, aby je zrozumieć i poprawnie uruchomić. Jednak po wielu godzin próbach symulacja uwzględniająca opóźnienia czasowe nie została wykonana. Nie byliśmy w stanie wygenerować pliku .sdf z zapisanymi opóźnieniami dla bramek, zapisane tam były tylko opóźnienia dla ścieżek. Problem ten może leżeć w konfiguracji poszczególnych etapów w Qflow.

Rozdział 6

Podsumowanie

Przygotowanie prostego modułu w języku Verilog oraz sprawdzenie jego działania przebiegło bezproblemowo. Natomiast podczas procedury syntezy układu zaczęły pojawiać się niedogodności.

Pierwsze problemy pojawiły się podczas instalacji narzędzia Qflow. Najnowsza wersja programu zainstalowana z oficjalnego repozytorium powodowała wiele błędów, natomiast wersja 1.3.17 zainstalowana poprzez standardowe repozytoria package Ubuntu działała poprawnie i była w stanie sama poprawnie skonfigurować swoje środowisko. Jednak w przypadku każdej wersji Qflow wykazywał brak pliku `osu018_stdcells.gds2`.

Problemem okazała się również biblioteka *OpenSTA* przy użyciu której próbowaliśmy przeprowadzić analizę opóźnień układu. Oficjalna strona wydawcy informuje bowiem, że dokumentacja jest przestarzała. Po wielu godzinach prób nie udało się nam przeprowadzić analizy opóźnień bramek oraz ich połączeń. I na koniec nie jesteśmy w stanie do końca stwierdzić gdzie leży problem.

Ostatecznie można stwierdzić, że społeczność związana z tymi technologiami nie jest zbyt duża, co powoduje, że wsparcie ciężko uzyskać w danym problemie.

Poprzez przygotowanie projektu udało się nam poznać podstawy implementacji fizycznej układów cyfrowych. Dowiedzieliśmy się, z jakich etapów składa się taki proces, jakich narzędzi możemy do tego użyć oraz jakie elementy są nam potrzebne do przeprowadzenia tego procesu.

Literatura

- [1] C. Wolf: *Yosys open synthesis suite* - <http://www.clifford.at/yosys/>
- [2] R. Timothy Edwards: *Qflow 1.3: An Open-Source Digital Synthesis Flow* - <http://opencircuitdesign.com/qflow/>
- [3] C. Wolf: *CMOS Gate-Level Netlist* - <https://github.com/YosysHQ/yosys/tree/master/examples/cmos>
- [4] dr inż. Cezary Maj: *Kurs Verilog* - <https://fiona.dmcs.pl/~cmaj/Verilog/Kurs%20verilog.pdf>
- [5] *PKD osu018* - <https://download.libresilicon.com/technologies/OSU018/>
- [6] Wikipedia: *Full-adder* - [https://pl.wikipedia.org/wiki/Sumator_\(uk%C5%82ad_logiczny\)](https://pl.wikipedia.org/wiki/Sumator_(uk%C5%82ad_logiczny))