AGH
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Fizyki i Informatyki Stosowanej

Praca magisterska

Krystian Wojtas

kierunek studiów: **informatyka stosowana**

kierunek dyplomowania: **metody numeryczne**

Oprogramowanie sprzętu laboratoryjnego dedykowanego dla przedmiotu "Projektowanie Systemów Cyfrowych"

Opiekun: **dr inż. Krzysztof Świentek**

Kraków, wrzesień 2014

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....
(czytelny podpis)

Na kolejnych dwóch stronach proszę dołączyć kolejno recenzje pracy popełnione przez Opiekuna oraz Recenzenta (wydrukowane z systemu MISIO i podpisane przez odpowiednio Opiekuna i Recenzenta pracy). Papierową wersję pracy (zawierającą podpisane recenzje) proszę złożyć w dziekanacie celem rejestracji co najmniej na tydzień przed planowaną obroną.

Na kolejnych dwóch stronach proszę dołączyć kolejno recenzje pracy popełnione przez Opiekuna oraz Recenzenta (wydrukowane z systemu MISIO i podpisane przez odpowiednio Opiekuna i Recenzenta pracy). Papierową wersję pracy (zawierającą podpisane recenzje) proszę złożyć w dziekanacie celem rejestracji co najmniej na tydzień przed planowaną obroną.

Spis treści

1 Wstęp **8**
1.1 Cel pracy . . . 8

2 Elektronika cyfrowa **10**
2.1 Logika cyfrowa . . . 10
2.1.1 Negacja . . . 10
2.1.2 Koniunkcja . . . 10
2.1.3 Alternatywa . . . 10
2.1.4 Alternatywa wykluczająca . . . 11
2.1.5 Zasada precedencji . . . 11
2.1.6 Prawa De Morgana . . . 11
2.2 Bramki logiczne . . . 12
2.2.1 Hazard . . . 13
2.3 Systemy liczbowe . . . 13
2.3.1 Liczby binarne . . . 13
2.4 Układy kombinacyjne . . . 14
2.4.1 Jedno-bitowy układ dodający . . . 15
2.4.2 Wielobitowy układ dodający . . . 15
2.4.3 Jedno-bitowy układ porównujący . . . 16
2.4.4 Wielobitowy układ porównujący . . . 17
2.4.5 Multiplexer . . . 17
2.5 Układy sekwencyjne . . . 18
2.5.1 Zatrzask SR . . . 18
2.5.2 Przerzutnik D . . . 18
2.5.3 Rejestr . . . 18
2.5.4 Licznik . . . 19
2.6 FPGA . . . 19
2.6.1 Architektura . . . 19
2.7 Języki Opisu Sprzętu . . . 20
2.7.1 Verilog . . . 20
2.7.2 Procesor . . . 22
2.7.3 Program . . . 24
2.7.4 Symulacja . . . 25

3 Moduły ogólnego przeznaczenia **27**
3.1 Projekty . . . 27
3.1.1 Układ katalogów . . . 27
3.2 Synteza . . . 27
3.2.1 Licznik . . . 27
3.2.2 Drgania styków . . . 28
3.2.3 Spowalniacz zegara . . . 29
3.2.4 Rejestr przesuwny . . . 29
3.2.5 Wykrywacz zbocza . . . 30
3.2.6 Serializacja . . . 31
3.2.7 SPI . . . 32
3.2.8 Generator impulsów . . . 32
3.2.9 Odwracacz bitów . . . 33
3.2.10 Funkcja logarytmiczna . . . 33

3.3 Symulacja . . . 34
3.3.1 Zegar . . . 34
3.3.2 Reset . . . 34
3.3.3 Ustawiacz . . . 34
3.3.4 Monitor . . . 36

4 DAC **41**
4.1 Komunikacja . . . 41
4.1.1 SPI . . . 41
4.1.2 Połączenia . . . 42
4.1.3 Protokół komunikacji . . . 43
4.2 Aplikacja . . . 43
4.3 Synteza . . . 43
4.3.1 Warstwa wierzchnia . . . 43
4.3.2 Kontroler . . . 44
4.3.3 DacSpi . . . 45
4.4 Symulacja . . . 45
4.4.1 Przypadek testowy . . . 45
4.4.2 DAC zachowawczo . . . 45
4.4.3 Przebieg . . . 50

5 Rotor **52**
5.1 Aplikacja . . . 52
5.2 Synteza . . . 53
5.2.1 Warstwa wierzchnia . . . 53
5.2.2 Rotor . . . 53
5.2.3 Kontroler . . . 54
5.3 Symulacja . . . 55
5.3.1 Przypadek testowy . . . 55
5.3.2 Rotor zachowawczo . . . 56
5.3.3 Komunikaty . . . 56

6 Rs232 **58**
6.1 Aplikacja . . . 58
6.2 Synteza . . . 58
6.2.1 Warstwa wierzchnia . . . 58
6.2.2 Transmisja . . . 58
6.2.3 Odbiór . . . 60
6.3 Symulacja . . . 62
6.3.1 Rs232 . . . 62
6.3.2 Rs232 zachowawczo . . . 63
6.3.3 Wyjście . . . 65

7 Klawiatura PS2 **67**
7.1 Interfejs PS2 . . . 67
7.2 Klawiatura . . . 67
7.3 Aplikacja . . . 68
7.4 Synteza . . . 68
7.4.1 Warstwa wierzchnia . . . 68
7.4.2 Kontroler . . . 68

7.4.3 Klawiatura . . . 69
7.5 Symulacja . . . 71
7.5.1 Przypadek testowy . . . 71
7.5.2 Klawiatura zachowawczo . . . 71
7.5.3 Przebieg . . . 74

8 VGA **77**
8.1 Aplikacja . . . 77
8.2 Synteza . . . 77
8.2.1 Połączenia . . . 77
8.2.2 Synchronizacja . . . 78
8.2.3 Controller . . . 79
8.3 Symulacja . . . 80
8.3.1 Przypadek testowy . . . 80
8.3.2 Moduł behawioralny . . . 80
8.3.3 Synchronizacja . . . 81
8.3.4 Zliczanie linii . . . 83
8.3.5 Przebieg . . . 84

9 Bibliografia **86**

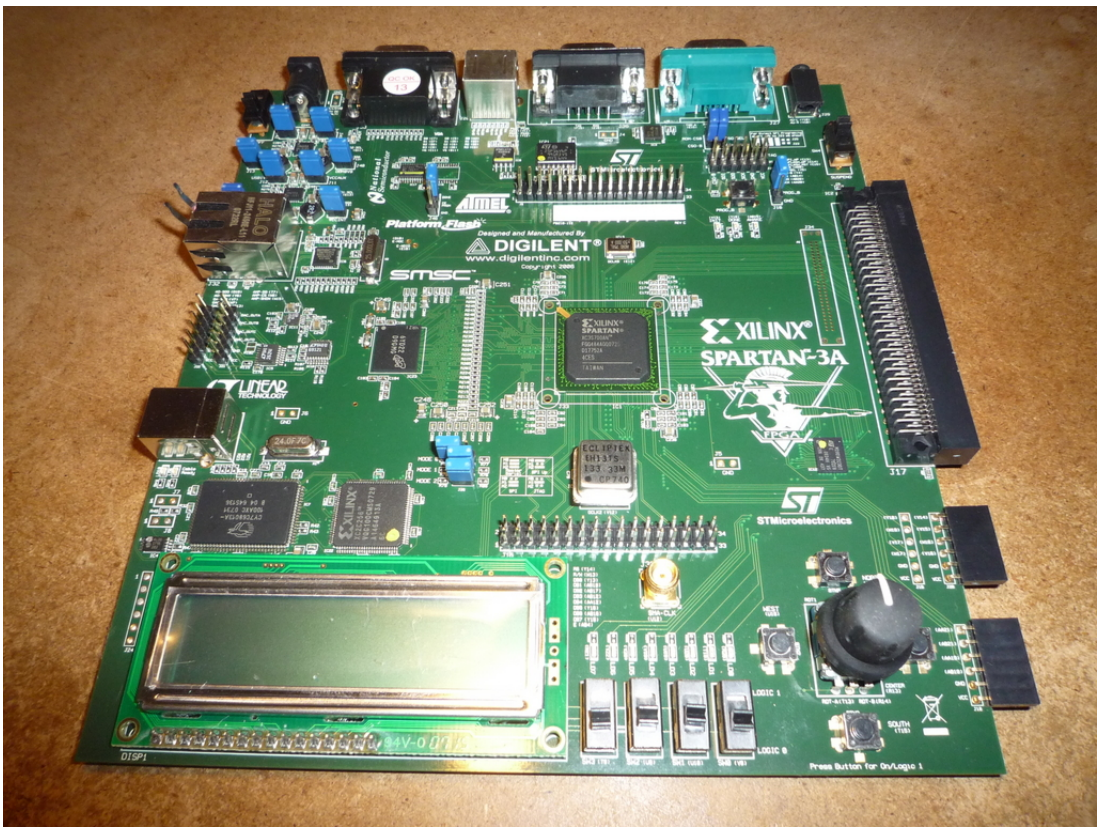
10 Dodatki **87**
10.1 Makefile . . . 87
10.1.1 DAC . . . 90
10.1.2 Rotor . . . 90
10.1.3 Rs232 . . . 91
10.1.4 VGA . . . 91

1 Wstęp

Nie sposób przecenić wynalazku tranzystora z roku 1950 uhonorowanego nagrodą Nobla oraz wszelkich jego późniejszych następstw pozwalających ujrzeć świat w obecnie oglądanej elektronicznej postaci. Droga ekspansji mikroprocesorów wydaje się już być nieodwracalna. Zrewolucjonizowały każdą dziedzinę życia. A to dopiero początek, wciąż jeszcze wiele przed nami. Tradycyjnie kultywowane systemy edukacji, obiegu pieniądza, sprawowania władzy państwowej są przestarzałe. Jednakże chociaż kwestia inwigilacji obywateli jest prężnie rozwijana oraz ukoronowana wszelkimi zdobyczami techniki. Zachodzi pytanie, czy świat opisany przez Orwella w wizji 'Rok 1984' właśnie nie nastaje. Technicznie jest już to możliwe, a skalę zapoczątkowanego zjawiska unaocznił Edward Snowden upubliczniając dokonania Agencji Której Nie Ma.

Powstałe zagadnienia natury etycznej wymagają szerokiej dyskusji. Byłaby ona sensowniejsza przy zrozumieniu działania zewsząd otaczającego nas sprzętu i jego możliwości. Niestety na początku 21 wieku wiedza ta nie jest powszechna, a nawet uważana jest za specjalistyczną. Niniejsza praca jest próbą odczarowania demonów tkwiących w tematyce elektroniki cyfrowej i zmycia z niej znamion nieokiełznanej aury tajemniczości.

1.1 Cel pracy



Rysunek 1: Xilnix Spartan-3AN Starter Kit

Celem pracy jest oprogramowanie za pomocą języka opisu sprzętu (Verliog) układów używanych podczas zajęć laboratoryjnych z Projektowanie Systemów Cyfrowych. Praca polegałaby na przygotowaniu zestawu syntezowalnych bloków HDL do wszystkich elementów płytki „Xilnix Spartan-3AN Starter Kit" (www.xilinx.com/products/devkits/HW-SPAR3AN-SK-UNI-G.htm). Dodatkowo należy opracować modele behawioralne służące do testowania poprawności kodu (poszczególnych modułów sprzętu) tworzonego przez studentów podczas zajęć.

Podsumowaniem całości ma być projekt kompleksowo demonstrujący możliwości wyżej wspomnianego sprzętu.

Utworzone moduły behawioralne muszą jak najwierniej odzwierciedlać zachowania układów występujących na płytce. Znajdą wtedy zastosowanie w uruchamianych symulacjach przebiegów czasowych danej konfiguracji układu programowalnego FPGA. Dzięki nim możliwe będzie stwierdzenie czy dla zsyntetyzowanej konfiguracji stany linii FPGA prowadzące do konkretnego układu płytki przebiegają poprawnie i czy zachodzi pożądana komunikacja poprzez generowanie przez te moduły stosownych komunikatów. W ten sposób studenci będą mogli testować poprawność działania utworzonej przez siebie konfiguracji bez fizycznego dostępu do sprzętu.

2 Elektronika cyfrowa

Elektronika cyfrowa operuje na prądach zmiennych, nieokresowych o stałych napięciach. Odróżniane są jedynie dwa stany logiczne - stan wysokiego potencjału będący logicznym wyrażeniem prawdy oraz stan niskiego potencjału - odpowiednik fałszu. Tworzona jest sieć logiczna operująca na zadanych stanach i dostarczająca rezultat działania na swoich wyjściach. Sieć często pracuje w rytm podsuniętego zegara - okresowego sygnału prostokątnego o 50% wypełnieniu. Sieć może zapamiętać pewne stany wejściowe i brać je pod uwagę w wyliczaniu następnych cykli.

2.1 Logika cyfrowa

Każda sieć zbudowana jest jedynie z kilku typów podstawowych operacji logicznych.

2.1.1 Negacja

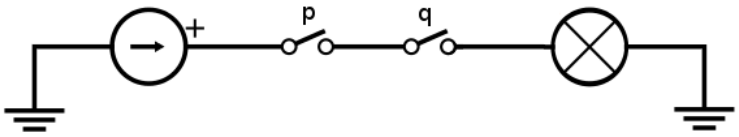
Element negujący zawsze zaprzeczy każdej podanej treści i podmieni jej zawartość na przeciwną.

p	$\neg p$
0	1
1	0

2.1.2 Koniunkcja

Koniunkcja wyznacza prawdę tylko, gdy wszyscy jej doradcy jednogłośnie ją potwierdzają.

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

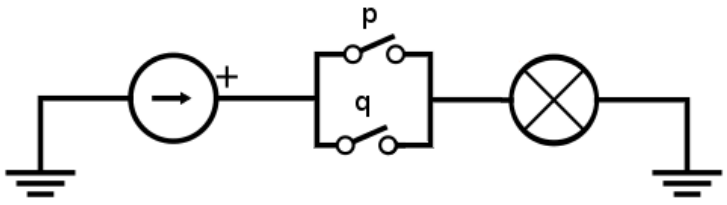


Demonstracja koniunkcji zrealizowanej na połączeniach

2.1.3 Alternatywa

Alternatywa powiela prawdę, jeśli tak stanowi co najmniej jeden jej doradca.

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1



Demonstracja alternatywy zrealizowanej na połączeniach

2.1.4 Alternatywa wykluczająca

Alternatywa wykluczająca przechyli się ku prawdzie, gdy jest ona głoszona przez nieparzystą liczbę doradców.

p	q	$p \oplus q$
0	0	0
0	1	1
1	0	1
1	1	0

2.1.5 Zasada precedencji

Pomiędzy zdaniami logicznymi używa się nawiasowania dla zmiany pierwszeństwa ich ewaluacji. Naturalnym biegiem jest rozpatrywanie ich według przyjętej hierarchii $\neg \wedge \vee \iff$.

2.1.6 Prawa De Morgana

Do formułowania praw należy wprowadzić kolejny symbol logiczny równoważności. W zasadzie jest on zaprzeczeniem poznanej już alternatywy wykluczającej.

p	q	$p \iff q$
0	0	1
0	1	0
1	0	0
1	1	1

I prawo De Morgana to prawo zaprzeczania koniunkcji. Negacja koniunkcji jest równoważna alternatywie negacji.

$\neg(p \wedge q) \iff (\neg p \vee \neg q)$

p	q	$p \wedge q$	$\neg(p \wedge q)$	$\neg p$	$\neg q$	$(\neg p \vee \neg q)$	$\neg(p \wedge q) \iff (\neg p \vee \neg q)$
0	0	0	1	1	1	1	1
0	1	0	1	1	0	1	1
1	0	0	1	0	1	1	1
1	1	1	0	0	0	0	1

II prawo De Morgana to prawo zaprzeczenia alternatywy. Negacja alternatywy jest równoważna koniunkcji negacji

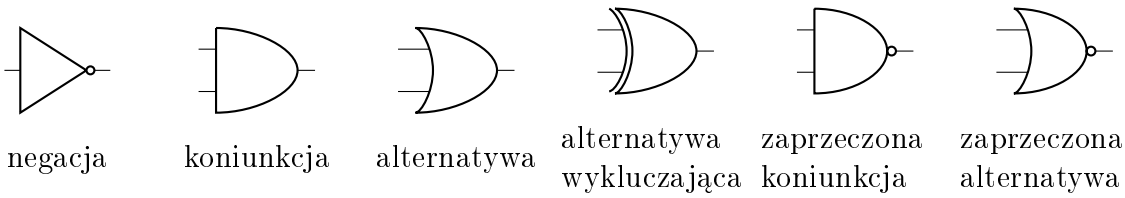
$\neg(p \vee q) \iff (\neg p \wedge \neg q)$

p	q	$p \vee q$	$\neg(p \vee q)$	$\neg p$	$\neg q$	$(\neg p \wedge \neg q)$	$\neg(p \vee q) \iff (\neg p \wedge \neg q)$
0	0	0	1	1	1	1	1
0	1	1	0	1	0	0	1
1	0	1	0	0	1	0	1
1	1	1	0	0	0	0	1

Prawa te są bardzo pomocne do optymalizacji sieci.

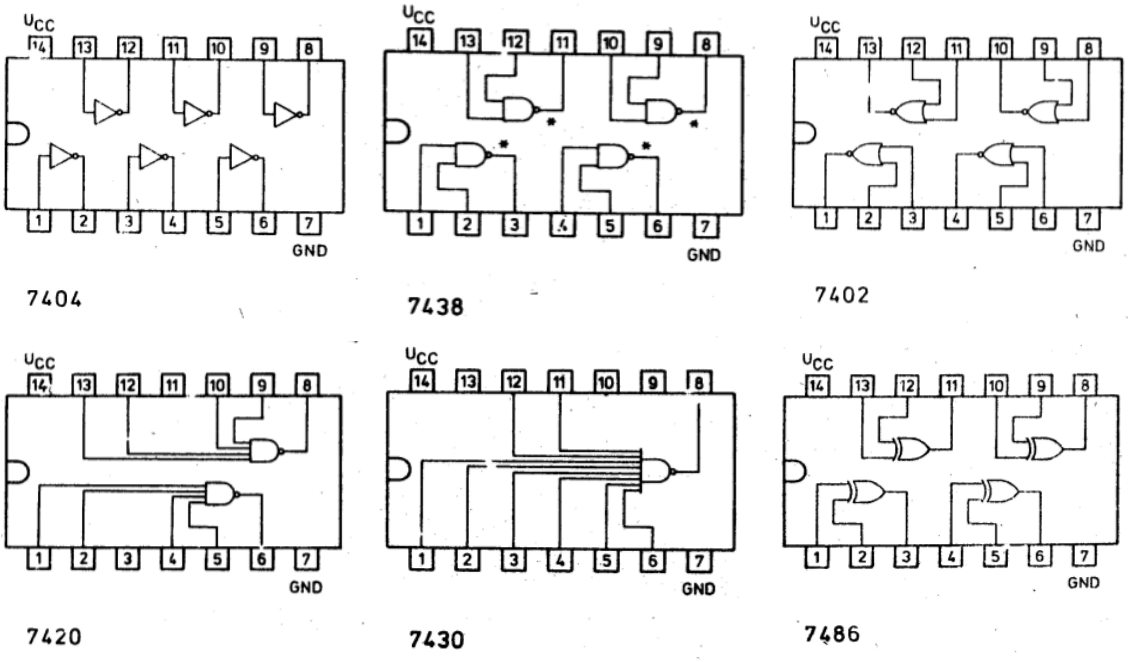
2.2 Bramki logiczne

Są to układy scalone realizujące wyspecyfikowane przez producenta operacje logiczne. Na samych połączeniach przewodów można jedynie osiągnąć logikę koniunkcji lub alternatywy. Nie da się jednak obecnemu w sieci sygnałowi zaprzeczyć. Można by problem obejść używając mechanicznych przekaźników. Właściwsze mogą okazać się tranzystory, tudzież ich zestawy odpowiednio połączone i zatopione w gotowych scalakach nazywanych bramkami logicznymi.

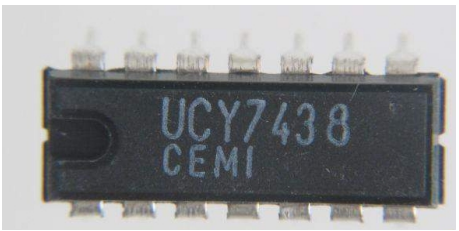


Oznaczenia schematyczne operacji logicznych

Dokumentacja układu określa położenia pinów wejściowych i wyjściowych każdej z dostępnych operacji.



Przykładowe bramki logiczne polskiej PRL-owskiej produkcji firmy CEMI



Egzemplarz bramki logicznej

2.2.1 Hazard

Zadanie nowego stanu wejścia wymaga czasu na przełączenie się wewnętrznych tranzystorów zanim ustali się właściwy wynik na wyjściu. Opóźnienie to nazywane jest czasem propagacji bramki. W tych krótkich momentach stany wyjść są nieustalone.

2.3 Systemy liczbowe

Cyfry są zestawem symboli danego systemu liczbowego ułożonych w określonym porządku według rosnących znaczeń. Przyjęło się uważać za naturalny system o dziesięciu cyfrach.

0_{min} 1 2 3 4 5 6 7 8 9^{max}

Końce porządku wyznaczają symbole najmniejszy i największy.

Liczba jest nieskończonym ciągiem cyfr.

Algorytm nieskończonej inkrementacji liczby zaczyna od jej postaci najmniejszej.

0_{min} ... 0_{min} 0_{min} 0_{min}

Przegląda ciąg zaczynając od najbardziej skrajnego prawego miejsca poszukując pierwszego symbolu niemaksymalnego. Natrafia na niego natychmiast.

Element ten jest sukcesywnie podmieniany na symbol starszy stopniem w szeregu.

0_{min} ... 0_{min} 0_{min} 1
0_{min} ... 0_{min} 0_{min} 2
0_{min} ... 0_{min} 0_{min} 3
0_{min} ... 0_{min} 0_{min} 4
0_{min} ... 0_{min} 0_{min} 5
0_{min} ... 0_{min} 0_{min} 6
0_{min} ... 0_{min} 0_{min} 7
0_{min} ... 0_{min} 0_{min} 8
0_{min} ... 0_{min} 0_{min} 9^{max}

Po wyczerpaniu puli na pierwszej pozycji, zajrzy na następną z kolei i tam napotka symbol niemaksymalny do podmiany na większy. Wszystkie symbole na prawo od znalezionej pozycji muszą zostać zminimalizowane.

0_{min} ... 0_{min} 1 0_{min}

Ponownie można podmieniać symbole na pierwszej pozycji zgodnie z porządkiem aż do osiągnięcia symbolu maksymalnego. Wtedy ponownie poszukiwana będzie pozycja pierwsza niemaksymalna i tu algorytm się zapętla.

0_{min} ... 0_{min} 1 1

2.3.1 Liczby binarne

Stosując tą samą zasadę postępowania można odliczać liczby w dowolnym systemie o uznanym porządku symboli. Logika binarna skraca pulę symboli do dwóch możliwości, odróżniając jedynie sybmole najmniejszy i największy.

0_{min} 1^{max}

Odliczanie zawsze startuje od nieskończonego ciągu symboli najmniejszych.

0_{min} ... 0_{min} 0_{min} 0_{min}

Już po pierwszej iteracji pula zostaje wyczerpana.

0_{min} ... 0_{min} 0_{min} 1^{max}

Należy użyć kolejnej, drugiej pozycji i wyzerować poprzednie.

0_{min} ... 0_{min} 1^{max} 0_{min}

Iteracja pierwszej pozycji

0_{min} ... 0_{min} 1^{max} 1^{max}

I znowu przepełnienie, tym razem już dwóch pierwszych pozycji. Należy poruszyć trzecią jako pierwszą niemaksymalną.

0_{min} ... 1^{max} 0_{min} 0_{min}

Na dwóch pierwszych pozycjach zostają powtórzone trzy poprzednie kroki, po czym należy sięgnąć na pozycję czwartą i powtórzyć kroki jej dotychczasowe. I tak do nieskończoności.

dwójkowy	ósemkowy	dziesiętny	szesnastkowy
00000	00	00	00
00001	01	01	01
00010	02	02	02
00011	03	03	03
00100	04	04	04
00101	05	05	05
00110	06	06	06
00111	07	07	07
01000	10	08	08
01001	11	09	09
01010	12	10	0A
01011	13	11	0B
01100	14	12	0C
01101	15	13	0D
01110	16	14	0E
01111	17	15	0F
10000	20	16	10
10001	21	17	11
10010	22	18	12
10011	23	19	13
10100	24	20	14

Tabela odzwierciedla odliczanie w paru systemach liczbowych

2.4 Układy kombinacyjne

Są to układy cyfrowe, które nie zapamiętują stanów pośrednich obliczeń wewnątrz sieci. Stany wyjść są zależne jedynie od stanów bieżących wejść. Są ich funkcją logiczną.

Linie wejść/wyjść układów cyfrowych często są liczbami reprezentowanymi w systemie dwójkowym. Uprzednio zaprezentowane nieskończone ciągi cyfr są modelem matematycznym nie mającym odzwierciedlenia w realnych układach elektronicznych na współczesnym etapie rozwoju techniki. Maszyny przejawiają skończoną precyzję obliczeń wykonywaną w skończonym czasie.

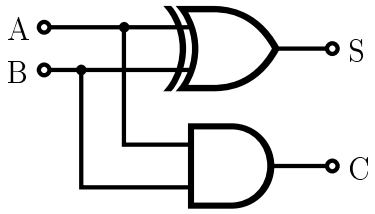
W systemie dwójkowym (binarnym) pojedynczą cyfrę nazywa się jednym bitem informacji. Na jednym bicie można zapisać jedną z dwóch możliwych wartości 0_{min} lub 1^{max}. Na dwóch bitach zapisuje się wszystkie dwie możliwości pierwszego bitu zdwukrotnione dwiema możliwościami bitu drugiej pozycji. Razem 4 możliwości. Dodając trzeci bit, również ilość możliwości się dubluje dając 8. I tu uwidacznia się pewna reguła - na n bitach można zapisać jedną z 2^n możliwych liczb. Odliczając od zera, liczbą największą będzie $2^n - 1$.

2.4.1 Jedno-bitowy układ dodający

Układ ten dodaje dwa jedno-bitowe sygnały nazwane A i B. Jeśli oba wejścia wystawią stan wysoki, wtedy wynik 2 w systemie binarnym $(10)_2$ jest liczbą dwucyfrową wymagającą dwóch wyjściowych linii S i starszej C (informującej też o przepełnieniu).

Po sformułowaniu tablicy prawdy łatwo zauważyć, że linia S jest alternatywą wykluczającą wejść, natomiast C jest ich koniunkcją.

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sieć logiczna jedno-bitowego układu dodającego

2.4.2 Wielobitowy układ dodający

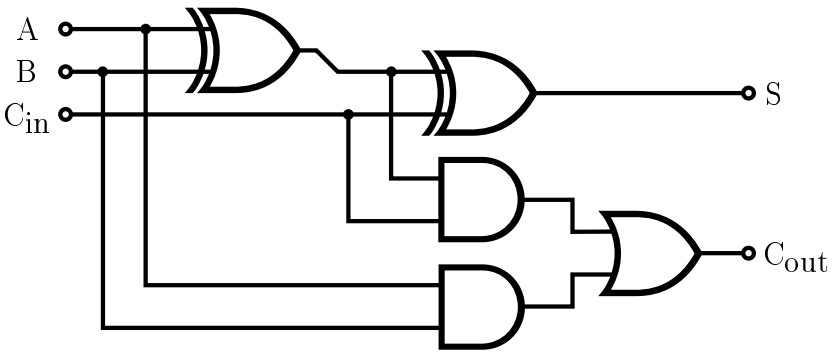
Dodawanie liczb złożonych z większej liczby bitów wymaga właściwego obchodzenia się z bitem przepełnienia. Mechanizm ujawnia dodawanie pisemne, które jest analogiczne jak w systemie dziesiętnym. Dla przykładu użyte zostaną liczby $A = 14 = (1110)_2$ oraz $B = 7 = (0111)_2$. Indeks przy oznaczeniu liczby odnosi się do cyfry na wskazanej pozycji bitu.

	1	1	1		
		1	1	1	0
+		0	1	1	1
	1	0	1	0	1

	C_4	C_3	C_2	C_1	
		A_3	A_2	A_1	A_0
+		B_3	B_2	B_1	B_0
		S_3	S_2	S_1	S_0

Najpierw dodawane są najmłodsze bity $A_0 = 0$ i $B_0 = 1$, a ich wynikiem, zgodnie z tabelą dodawania jedno-bitowego, jest $S_0 = 1$. W następnej kolumnie występują dwa stany wysokie, co zgodnie z tabelą wynosi 0 dla S_1, ale ustawiany jest bit przepełnienia C_2. Dla trzeciej kolumny ponownie oba wejścia są wysokie, jednak należy także wziąść pod uwagę przepełnienie C_2 z poprzedniego kroku. Ta sytuacja dotąd nie została uwzględniona. Należy dla niej zaprezentować ogólniejszą tabelę prawdy uwzględniającą bit przepełnienia z poprzedniego dodawania.

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

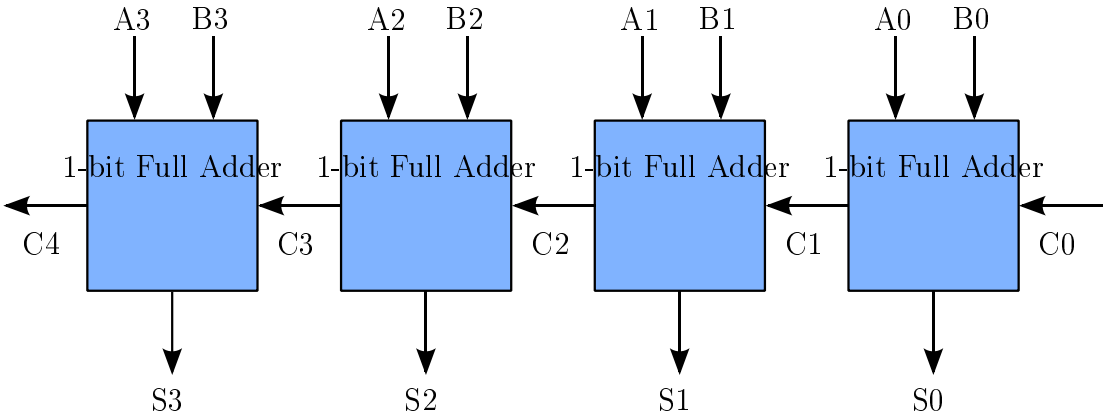


Sieć logiczna pojedynczego ogniwa wielobitowego układu dodającego.

Teraz widać, że ogólna postać dodawania ma trzy wejścia - dwa są bitami z obu składników na odpowiadających sobie pozycjach, trzeci sygnał jest wartością przepełnienia z dodawania na poprzedniej pozycji.

Tak więc w kolumnie trzeciej wystąpiła sytuacja, gdzie oba wejścia są wysokie jak również i bit przepełnienia. Według nowej tablicy trzy jedynki na wejściu dają dwie na wyjściu.

Czwarta kolumna otrzymuje przepełnienie i jedną jedynkę, co oznacza zero dla S_3 oraz ostatnie przepełnienie C_4. Wynikiem jest $21 = (10101)_2$, co wymaga zapisu na 5 bitach. Ostatni bit przepełnienia zapewnia brakujący piąty bit wyniku, dzięki czemu suma jest właściwa w postaci $C_4S_3S_2S_1S_0$.

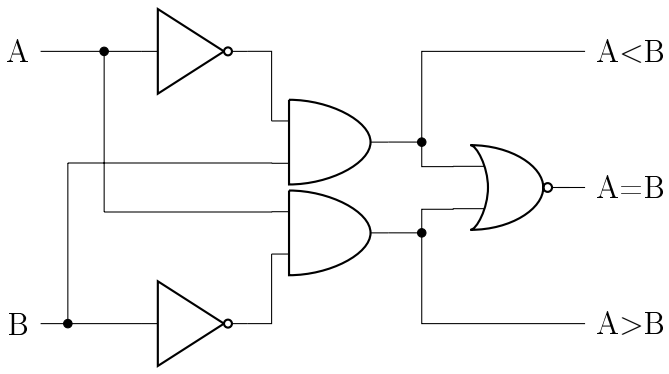


4-bitowy układ dodający zbudowany jest z 4 ogniw

2.4.3 Jedno-bitowy układ porównujący

Układ porównujący stwierdza, czy dwa wejściowe sygnały A i B są sobie równe linią $=$. A jeśli nie są, wtedy określi który z nich reprezentuje większą wartość za pomocą wyprowadzeń $<$ oraz $>$.

A	B	$A < B$	$A = B$	$A > B$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



Sieć logiczna jedno-bitowego układu porównującego

2.4.4 Wielobitowy układ porównujący

Równość dwóch dowolnych wartości n-bitowych stwierdza się, jeśli wszystkie odpowiadające sobie bity są sobie równe.

$x_i = A_i \cdot B_i + \overline{A}_i \cdot \overline{B}_i$.

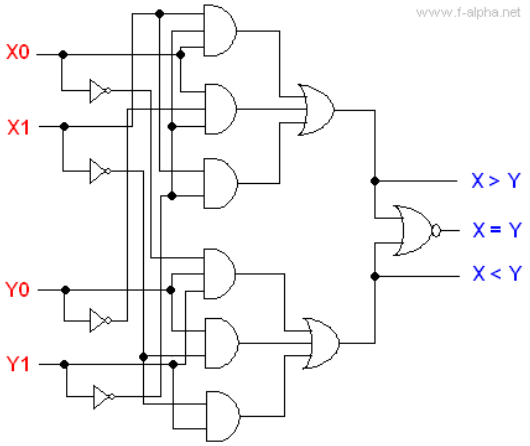
$(A = B) = x_n x_{n-1}(...)x_2 x_1 x_0$

Nierówność stwierdza się porównując odpowiadające sobie bity kolejno poczynając od najstarszego aż do wykrycia pierwszej nieprawidłowości. Znak $<$ lub $>$ nierówności klaruje się według porównania jedno-bitowego na znalezionej pozycji.

$(A > B) = A_3 \cdot \overline{B}_3 + x_3 A_2 \overline{B}_2 + x_3 x_2 A_1 \overline{B}_1 + x_3 x_2 x_1 A_0 \overline{B}_0$

$(A < B) = \overline{A}_3 \cdot B_3 + x_3 \overline{A}_2 B_2 + x_3 x_2 \overline{A}_1 B_1 + x_3 x_2 x_1 \overline{A}_0 B_0$

A_1	A_0	B_1	B_0	$A < B$	$A = B$	$A > B$
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	1
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

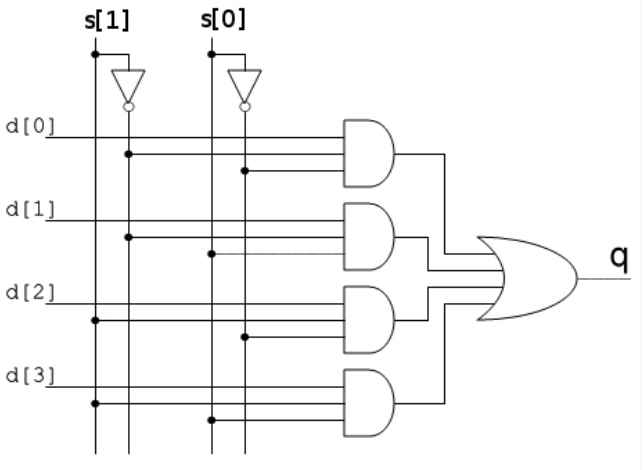


Sieć logiczna dwu-bitowego układu porównującego

2.4.5 Multiplexer

Multiplexer otrzymuje komplet danych, lecz ukazuje jedynie wybraną, interesującą ich część. Preferencji dokonuje się ustawiając bity maski select.

$s[1]$	$s[0]$	q
0	0	d[0]
0	1	d[1]
1	0	d[2]
1	1	d[3]



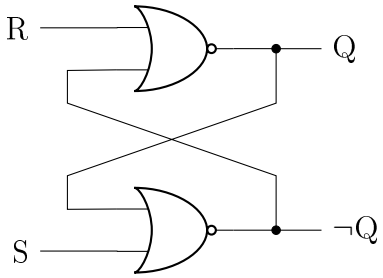
Sieć logiczna multiplexera

2.5 Układy sekwencyjne

W układach sekwencyjnych do pewnych sygnałów wejściowych są przyłączone niektóre wyjścia sprzężeniem zwrotnym. Dzięki temu na bieżący wynik obliczeń wpływają obliczenia uprzednie. Zachodzi efekt pamięci - sieć pamięta swoje poprzednie stany.

2.5.1 Zatrzask SR

$\neg S$	$\neg R$	Q	$\neg Q$
0	0	x	x
0	1	1	0
1	0	0	1
1	1	Q_{n-1}	$\neg Q_{n-1}$

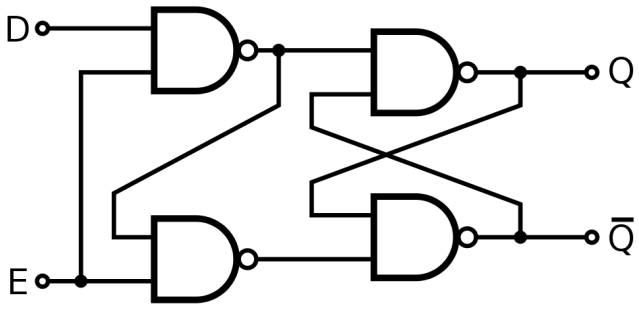


Zatrzask SR

Przerzutnik SR zbudowany jest z dwóch bramek NOR lub NAND. Wynik wyjścia jednej bramki skierowany jest ponownie do wejścia drugiej. Dzięki temu pozostawiając układ w stanie neutralnym, stale utrzymuje on na swoim wyjściu ostatnio wybrany stan. Stan wybiera się wytrącając układ ze stanu neutralnego krótkim zwarciem linii S lub R do masy. Jeśli obie linie naraz zostaną zwarte, wtedy stan wyjściowy uzależniony jest od wewnętrznych hazardów i przyjmuje się go za nieustalony.

2.5.2 Przerzutnik D

D	C	Q
x	0	Q_{n-1}
0	1	0
1	1	1



Przerzutnik D

Przerzutnik D jest rozszerzeniem zatrzasku SR o 2 dodatkowe bramki uniemożliwiające osiągnięcie stanu zabronionego obarczonego nieprzewidywalnymi hazardami. Oczekuje on przyłączenia linii danych D oraz zegarowej C. W chwilach wysokiego stanu zegara, stale przepisuje on daną D na wyjście. Gdy stan zegara stanie się niski, wtedy ostatnia przepisana i 'zapamiętana' wartość D jest utrzymywana na wyjściu.

Dwa przerzutniki D czułe na przeciwne poziomy zegara łączy się kaskadowo. Efektem jest przerzutnik czuły na zbocze zegara. Kierunek zbocza wyznacza kolejność podłączenia przerzutników.

2.5.3 Rejestr

Rejestr jest układem pamięciowym przechowującym bity informacji. Jego budowę można oprzeć na zestawie przerzutników typu D łącząc je równolegle.

2.5.4 Licznik

Licznik to układ inkrementujący liczbę przedstawianą binarnie na wyjściu w takt dostarczanego zegara. Zbudować go można łącząc szeregowo przerzutniki typu D. Do pierwszego dostarcza się sygnał zegarowy, a sygnałem C dla kolejnych jest wyjście Q z poprzedniego. Każdemu przerzutnikowi z osobna zwiera się wyjście $\neg Q$ z wejściem D. Inkrementowana liczba ujawnia się w ciągu $Q_{n-1}Q_{n-2}..Q_0$.

2.6 FPGA

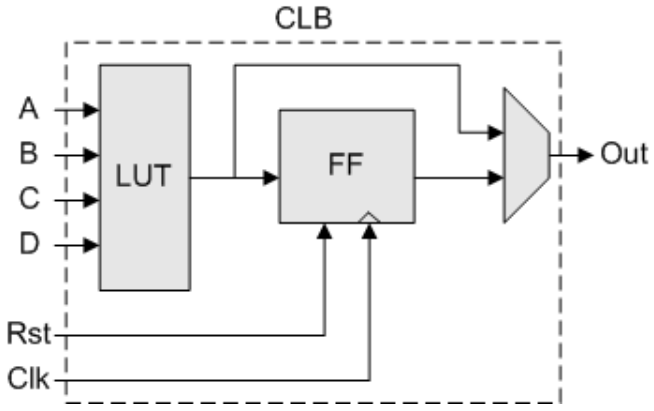
Field Programmable Gate Array jest układem elektronicznym pozwalającym na skonfigurowanie jego zachowania. Konfiguracja taka jest zestawem wygenerowanych danych dostarczonych do FPGA i określa logikę, według której układ ma pracować. Ekwiwalentną logikę przedstawić można układem bramek logicznych i przerzutników.

2.6.1 Architektura

W układzie FPGA wyszczególnić można elementy nazywane blokami logicznymi, które zwarte są ze sobą siecią konfigurowalnych połączeń. Linie wejściowe bloku logicznego poprowadzone poprzez multipleksery do przerzutników* stają się indeksem komórki pamięci. Wartość tam zapisana będzie wynikiem działania sieci. Pamięć ta przechowuje możliwe stany wyjściowe dla wszystkich kombinacji wejść, stąd nazywana jest tablicą przeszukiwań LUT. Tym sposobem odwzorować można dowolną logikę kombinacyjną.

Układy FPGA pozwalają na więcej, także odzwierciedlają logikę sekwencyjną. Wynik z tablicy LUT jest poprowadzony przez multiplekser. Jego konfiguracja powoduje bezpośrednie wyprowadzenie sygnału z bloku logicznego lub przeprowadzenie go przez dedykowany przerzutnik FF (taktowany wspólnym zegarem dla wszystkich bloków). W tym przypadku bloki logiczne spełniają także funkcję pamięciową.

Logika programowalna przechowuje konfigurację swojego działania w rozproszonych w układzie pamięciach RAM. Zapis pamięci o połączeniach między blokami logicznymi, zawartością ich tablic LUT oraz konfiguracją ich multiplekserów dla przerzutników jest konfiguracją układu FPGA. Od zadanej konfiguracji zależy działanie całego układu elektronicznego.



Architektura bloku logicznego FPGA

2.7 Języki Opisu Sprzętu

Modelowanie pojedynczych bitów w poszczególnych pamięciach RAM układu FPGA byłoby niezwykle żmudnym i błędogennym zadaniem, a powstały efekt końcowy byłby nieprzenośny na układy nowsze lub innych producentów. Stąd narosła potrzeba narzędzia do wygenerowania konfiguracji FPGA dla wskazanego chipsetu bazując na dostarczonym modelu pożądanego funkcjonowania sprzętu. Model taki sporządzany jest w języku opisu sprzętu elektronicznego. Najpopularniejsze to VHDL i Verilog oraz wzbogacony o rozszerzenia obiektowe SystemVerilog.

Języki te ze swojej natury modelują procesy współbieżne synchronizowane na zboczach zegara taktującego cały układ.

2.7.1 Verilog

Verilog dysponuje dwoma podstawowymi typami danych. Pierwszym jest rejestr, oto jego deklaracja

Listing 1: Rejestr

```
1 reg rejestr_jednobitowy;
2 reg rejestr_jednobitowy_z_przypisana_wartoscia = 1'b1;
3 reg [7:0] rejestr_8bitowy = 8'haa;
4 reg [7:0] pamiec_256_elementow_1bajtowych [255:0];
```

Rejestry dane przechowują. Można je wpisywać i odczytywać. Prawa strona przypisania zawiera liczbę bitów przedstawianej liczby, po apostrofie definiuje jej system liczbowy zapisu (z dostępnych binarnego b, heksadecymalnego h i dziesiętnego d), na końcu przedstawia samą liczbę.

Drugim typem danych jest połączenie tworzące więź z rejestrem macierzystym.

Listing 2: Polaczenie

```
1 wire polaczenie_jednobitowe = rejestr_jednobitowy;
2 wire [2:0] ekstrakcja_3bitow_z_rejestru = rejestr_8bitowy[3:1];
3 wire [7:0] konkatenacja_rejestrow = {rejestr_8bitowy[6:0], rejestr_jednobitowy};
4 wire kombinatoryka = (rejestr_jednobitowy) ? rejestr_8bitowy[4] : 1'b1;
```

Połączenia mogą jedynie wskazywać dane zamieszczone w rejestrach. Nie można do nich danych zapisywać, do tego celu potrzebny jest bezpośredni dostęp do wskazywanego rejestru. Połączenia mogą ukazywać jedynie interesującą część rejestru dzięki operatorowi ekstrakcji []. Mogą też wskazywać części danych zapisanych w kilku innych rejestrach poprzez ich konkatenację {}. Operator trójargumentowy ? : umożliwia zapisanie wyrażeń kombinacyjnych warunkowych.

Dostępne są podstawowe bramki logiczne i poprzez połączenie między nimi modeluje się logikę kombinacyjną, na przykład sumatora jedno bitowego.

Listing 3: Sum1.v

```
1 wire A, B, Cin, Y, Cou;
2 wire g1_o, g2_o, g3_o, g4_o;
3
4 and g1(g1_o, A, B);
5 xor g4(g4_o, A, B);
6 and g2(g2_o, Cin, g4_o);
7 or  g3(Cout, g1_o, g2_o);
8 xor g5(Y, Cin, g4_0);
```

Jednakże właściwsze jest modelowanie na wyższym poziomie abstrakcji zwanym zachowawczym. Syntezator zdolny jest sam podstawić bramki pod logikę, którą się mu przedstawi opisowo.

Listing 4: Sum1.v

```
wire A, B, Cin, Y, Cout;

{ Cout, Y } = A + B + Cin;
```

Do zamodelowania logiki sekwencyjnej posłużyć się należy rejestrem i blokiem *always* czułym na zbocze zegara. Zapisywanie wartości do konkretnego rejestru dozwolone jest tylko w jednym, dedykowanym bloku *always*. Przykładowy timer zlicza takty zegara głównego i w równych odstępach czasu informuje o swoich przepełnieniach.

Listing 5: Timer.v

```
reg [2:0] counter = 3'd0;
wire full = reg == 3'b111;

always @(posedge clk)
    reg <= reg + 1;
```

Zadeklarowany został 3-bitowy rejestr *counter* o początkowej zerowej wartości. Wyjście $full$ sformułowane jest kombinacyjnie i osiąga stan wysoki tylko, gdy licznik jest pełny. Na narastających zboczach dostarczonego z zewnątrz zegara *clk* wykonuje się blok *always* inkrementujący licznik. W rezultacie sygnał $full$ generowany jest co 8 takt zegara.

Odpowiednikiem obudów układów elektronicznych z wyprowadzonymi pinami są w Verilogu moduły. W ich opisie wyspecyfikowana jest lista wejść i wyjść, a wewnątrz deklarowanego modułu powoływać można instancje innych i modelować połączenia między nimi. Struktura jest hierarchiczna. Zewnętrzne połączenia modułu najwyższego poziomu są wyprowadzeniami pinów FPGA.

Listing 6: Top.v

```
module Top
(
    input wire clk,
    input wire rst,
    input wire [3:0] buttons,
    output wire [7:0] leds
);

wire blink_led;
wire light_all_leds;

Controller controller_(
    .buttons(buttons)
    .blink_led(blink_led),
    .light_all_leds(light_all_leds)
);

Leds leds_(
    .leds(leds),
    .blink_led(blink_led),
    .light_all_leds(light_all_leds)
);

endmodule
```

2.7.2 Procesor

Na FPGA można zaimplementować logikę arytmetyczną operującą na tych samych rejestrach. Operacje następowałyby pojedynczo w kolejności uzależnionej od ciągu dostarczonych instrukcji nazywanych programem. Co więcej, specjalne instrukcje skoku warunkowego sprawdzałyby stan bieżących obliczeń i od ich zależności mogłyby zakłócić porządek wykonywanego programu zmieniając aktualną pozycję w wykonywanym ciągu instrukcji. Taką konstrukcję nazywa się procesorem.

Przykładowy prosty procesor operuje na jednym 8-bitowym rejestrze zwanym akumulatorem. Ma on dostęp do pamięci RAM również o 8-bitowej szerokości szyny danych. Wszystkie instrukcje są jednakowej długości 2 bajtów i ulokowane są w oddzielnej pamięci PROM. Doprecyzowując procesor ten jest 8-bitowym potokowym przedstawicielem rodziny RISC o architekturze harwardzkiej.

Jego możliwości przedstawia tabela

Kod maszynowy	Mnemonik	Opis
0000 0000 $\{OP1\}_8$	LDI $\{OP1\}_8$	Ładuje do akumulatora wartość $\{OP1\}_8$ podaną w instr
0001 $\{ADDR\}_{12}$	LD $\{ADDR\}_{12}$	Ładuje do akumulatora wartość dostępną pod adresem $\{ADDR\}_{12}$
0010 $\{\{ADDR\}\}_{12}$	ST $\{\{ADDR\}\}_{12}$	Zapisuje wartość z akumulatora pod adresem $\{ADDR\}_{12}$
0011 0000 $\{OP1\}_8$	ADD $\{OP1\}_8$	Dodaje do akumulatora wartość podaną w instrukcji $\{OP1\}_8$
0100 0000 $\{OP1\}_8$	SUB $\{OP1\}_8$	Odejmuje od akumulatora wartość podaną w instrukcji $\{OP1\}_8$
0101 0000 $\{OP1\}_8$	AND $\{OP1\}_8$	Wykonuje na akumulatorze funkcję logiczną AND z operandem $\{OP1\}_8$
0110 0000 $\{OP1\}_8$	OR $\{OP1\}_8$	Wykonuje na akumulatorze funkcję logiczną OR z operandem $\{OP1\}_8$
0111 0000 $\{OP1\}_8$	XOR $\{OP1\}_8$	Wykonuje na akumulatorze funkcję logiczną XOR z operandem $\{OP1\}_8$
1000 0000 00000000	NOT	Wykonuje na akumulatorze funkcję logiczną NOT
1001 0000 00000000	LR	Przesuwa wartość akumulatora w lewo
1010 0000 00000000	RR	Przesuwa wartość akumulatora w prawo
1011 $\{ADDR\}_{12}$	JMP $\{ADDR\}_{12}$	Następne instrukcje procesor wykonana spod adresu $\{ADDR\}_{12}$
1100 $\{ADDR\}_{12}$	JMPZ $\{ADDR\}_{12}$	Jeśli w akumulatorze jest wartość zero, to następne instrukcje procesor wykonana spod adresu $\{ADDR\}_{12}$

Występują tu instrukcje bezargumentowe jak zaprzeczenie NOT oraz przesunięcia LR i RR. Reszta wymaga argumentu w postaci 12-bitowego adresu pamięci $\{ADDR\}_{12}$ lub 8-bitowego operandu $\{OP1\}_8$. Wartości te zakodowane są bezpośrednio w instrukcji na dalszych bitach.

Listing 7: CPU.v

```
1  module CPU
2  (
3      input clk,
4      input rst,
5      output [7:0] outport
6  );
7
8  reg [7:0]    ACC = 8'd0;
9  reg [11:0]   PC = 12'd0;
10 reg [11:0]   PCtmp = 12'd0;
11 reg [15:0]   IR = 16'd0;
12 wire [3:0]   OPCODE = IR[15:12];;
13 wire [7:0]   OP1 = IR[7:0];
14 wire [11:0] ADDR = IR[11:0];
15 wire [11:0] addr2 = { 1'b0, ADDR[11:1] };
16
17 reg [15:0] PROM [0:3];
18 initial $readmemb("out.bindump", PROM);
19 reg [7:0] RAM [0:31];
20 initial RAM[0] = 8'd0;
21 assign outport = RAM[0][7:0];
22
23 always @(posedge clk)
24     if(rst) PC <= 12'hfff;
25     else begin
26         PC <= PCtmp;
27         IR <= PROM[PCtmp];
28     end
29
30 always @*
31     case(OPCODE)
32         4'b1011: PCtmp = addr2;
33         4'b1100: if(ACC == 7'd0) PCtmp = addr2; else PCtmp = PC + 1;
34         default: PCtmp = PC + 1;
35     endcase
36
37 always @(posedge clk)
38     case(OPCODE)
39         // Load immediate value
40         4'b0000: ACC <= OP1;
41         // Load from / store to RAM
42         4'b0001: ACC <= RAM[ADDR];
43         4'b0010: RAM[ADDR] <= ACC;
44         // ALU
45         4'b0011: ACC <= ACC + OP1;
46         4'b0100: ACC <= ACC - OP1;
47         4'b0101: ACC <= ACC & OP1;
48         4'b0110: ACC <= ACC | OP1;
49         4'b0111: ACC <= ACC ^ OP1;
50         4'b1000: ACC <= ~ ACC;
51         4'b1001: ACC <= { ACC[6:0], ACC[7] };
52         4'b1100: ACC <= { ACC[0], ACC[7:1] };
53     endcase
54
55 endmodule
```

Moduł rozpoczyna się od zadeklarowania pamięci danych RAM, instrukcji PROM oraz niezbędnych rejestrów akumulatora ACC, wskaźników bieżącej instrukcji PC, PCtmp oraz rejestru

z pobraną bieżącą instrukcją IR. Poprzez połączenia wyekstrahowane są z rejestru instrukcji pola mnemonika OPCODE, adresu ADDR oraz operandu OP1.

W każdym takcie zegarowym procesor uaktualnia sobie wskaźnik następnej wykonywanej instrukcji oraz pobiera ją z pamięci. Następną instrukcją zazwyczaj jest kolejną w programie, chyba że zdekodowana została instrukcja skoku JMP. Wtedy procesor przeniesie wykonania programu pod instrukcję zadaną w argumencie adresu. Jeśli zdekodowany został skok warunkowy $JMPZ$, to najpierw sprawdzona zostaje zawartość akumulatora i skok nastąpi tylko przy jego zerowej wartości.

Ostatni blok zawiera jednostkę arytmetyczno-logiczną i wykonuje operacje na akumulatorze według zadawanych instrukcji.

2.7.3 Program

Zestaw instrukcji w przedstawionym kodzie maszynowym należy procesorowi dostarczyć, a wcześniej je wygenerować. Język nazywany asemblerem zamienia mnemoniki instrukcji czytelniejsze dla człowieka na właściwy kod maszynowy konkretnego procesora.

Zestaw dostępnych instrukcji zapisany jest makrami asemblera. Pewne makra przyjmują parametr adresu *addr* lub operandu *op*1 dokładnie jak odpowiadająca generowana instrukcja.

Listing 8: Instructions.asm

```
%define LDI(op1)     db 00000000b, op1
%define LD(addr)     db 00101000b, addr
%define ST(addr)     db 00110000b, addr
%define ADD(op1)     db 10000000b, op1
%define SUB(op1)     db 10001000b, op1
%define OR(op1)      db 10011000b, op1
%define XOR(op1)     db 10100000b, op1
%define NOT          db 10101000b, 0
%define LR           db 10111000b, 0
%define RR           db 11000000b, 0
%define JMP(addr)    db 00001000b, addr
%define JMPZ(addr)   db 00010000b, addr
```

Definicje makr wykorzystuje się do zapisania kodu czytelniejszego dla człowieka, a gotowego do tłumaczenia na kod maszynowy.

Listing 9: Diods.asm

```
%include "instructions.asm"

LDI(00110011b) ; zaladowanie wartosci do akumulatora
rotate:
ST(0)          ; przepisanie aktualnego stanu akumulatora do pamieci pod adres
    0, zapalenie odpowiadajacych sobie diod
LR             ; obrocenie rejestru akumulatora w lewo
JMP(rotate)    ; ponowne obroty wykonywany nieskonczenie
```

Program najpierw załącza definicję makr dostępnych instrukcji, po czym bazując na nich zapisuje swój przebieg działania. Pierwsze bajty programu to instrukcja ładująca wartość 00110011_2 do rejestru akumulatora. Następna instrukcja $ST(0)$ zaetykietowana *rotate* przepisuje zawartość akumulatora do pamięci pod adres 0. Wartość spod tego specjalnego adresu jest wyprowadzana na piny wyjściowe procesora. Jeśli do procesora przyłączone są diody, spowoduje to zaświecenie 4 diod. Dalej akumulator jest obracany w lewo, po czym następuje skok programu pod etykietę *rotate*. Nowa wartość zostaje wyprowadzona, co zapali 4 inne diody i wygasi resztę. I ponowny skok, program się zapętla rytmicznie przesuwając świecenie zestawu 4 diod.

Translacja źródeł programu na kod maszynowy dokonana jest asemblerem yasm.

Listing 10: Translacja

```
$ yasm -o out.bin program.asm
```

Źródło veriloga wczytuje obraz pamięci z zewnętrznego pliku, jednak zadanie $*readmemb* spodziewa się postaci liczb binarnych zapisanych w ASCII, co zaspokojone zostaje skryptem perla.

Listing 11: Zrzut

```
$ perl -ne '$\="\n" and map { print } unpack "(B16)*"' \
  < out.bin > project/out.bindump
```

Dla rozwiania wątpliwości przedstawiony jest zrzut wynikowego kodu maszynowego.

Listing 12: Wydruk

```
$ cat project/out.bindump
0000000000110011
0010000000000000
1001000000000000
1011000000000010
```

2.7.4 Symulacja

Moduły syntezowalne dla FPGA dobrze jest przesymulować dla zweryfikowania poprawności działania zamodelowanej logiki. Moduły symulacyjne instancjonują wierzchnią warstwę syntezy i zwykle podłączają im linie zegarowe i resetu. Linie te należy zdefiniować, a w tym celu używa się niesyntezowalnych instrukcji spowolnienia czasu. W przykładzie sygnał resetu ustawiany jest wysoko przez krótki czas, po czym wraca do stanu niskiego.

Listing 13: Reset

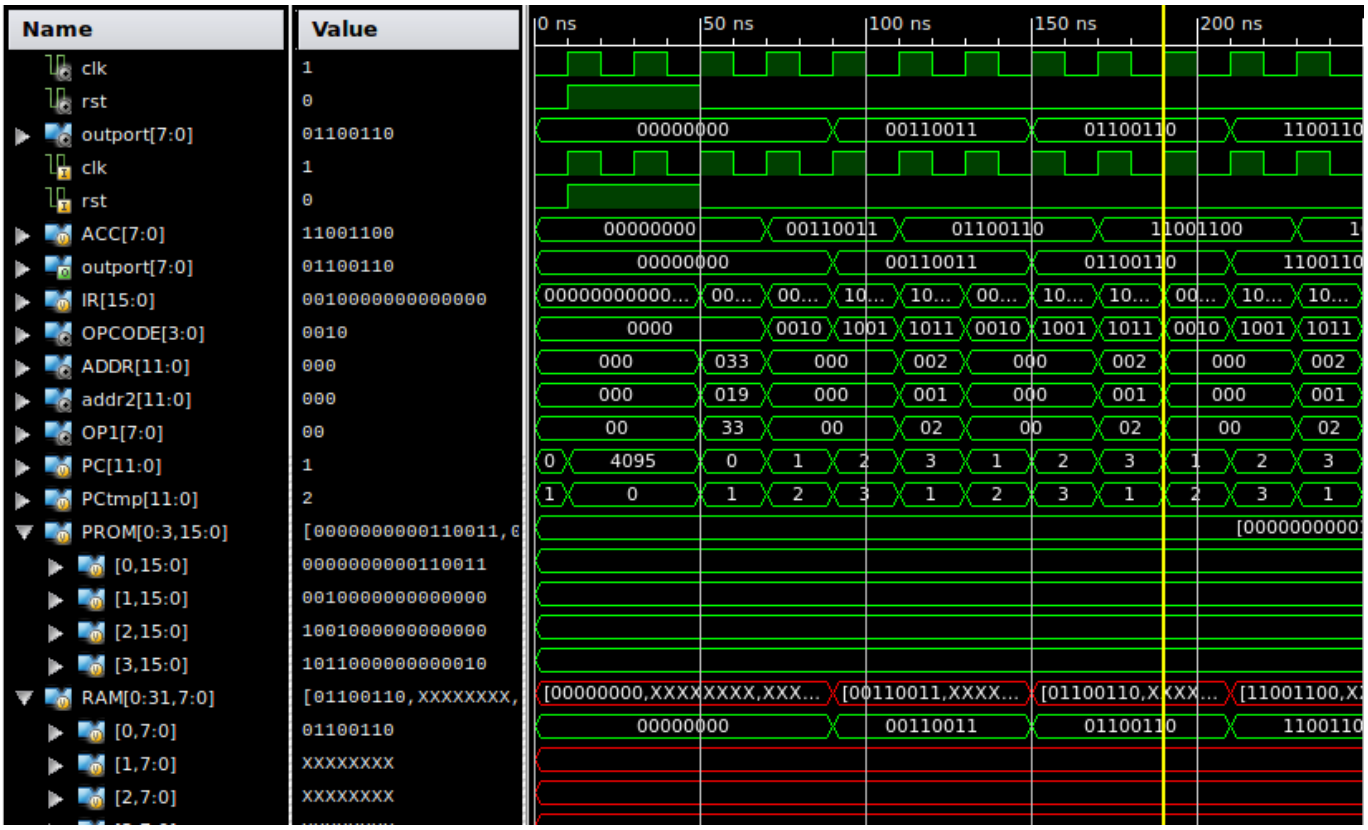
```
wire rst;
initial begin
    rst = 0;
    #10;
    rst = 1;
    #40;
    rst = 0;
end
```

Sygnał zegarowy definiuje się w opóźnianym czasowo bloku *always*.

Listing 14: Clk

```
wire clk;
initial clk = 0;
always #10 clk <= ~clk;
```

Tak przygotowane sygnały podłączone zostają do instancji CPU i uruchomione w symulacji.



Przebieg symulacji działania CPU

Symulacja przedstawia wykres stanów sygnałów w czasie. Zgodnie z oczekiwaniami zegar zmienia swój poziom co 10 ns, natomiast sygnał resetu został podniesiony na 40ns. Po tym procesor zaczął pracować. Po porównaniu opkodów kolejno wykonywanych instrukcji z zapisem programu nasuwa się wniosek o poprawności działania zamodelowanego układu.

3 Moduły ogólnego przeznaczenia

3.1 Projekty

Każdy z symulowanych układów jest osobnym symulowalnym jak również syntezowalnym projektem środowiska Xilinx ISE. Przygotowane są także skrypty do budowania dla programu *make*.

3.1.1 Układ katalogów

Praca wersjonowana jest rozproszonym systemem kontroli zawartości *git*. Drzewo katalogów rozpoczyna się rozgałęzieniem na *doc* oraz *projects*. *doc* zawiera źródła oraz materiały do zbudowania tego dokumentu, *projects* zawiera projekty, a drzewa katalogów każdego są podobne. Zamieszczona jest struktura projektu klawiatury.

```
keyboard/
    Controller.v
    Keyboard.v
    project
        default_wcfg
        isim.cmd
        keyboard.xise
        Makefile
    sim
        Keyboard_behav.v
        TopTestBench.v
        TopTest.v
    Top.ucf
    Top.v
    wcfg
        all.wcfg
        keyboard.wcfg
```

Wszelkie pliki syntezowalne mieszczą się na pierwszym poziomie w katalogach projektów. Źródła symulacyjne usytuowane są w podkatalogach *sim*. Przygotowane zbiory wykresów położone są w $wcfg$. Katalogi *project* gromadzą pliki projektów dla środowiska ISE Xilinx oraz alternatywnie skrypty $Makefile$. W nim również składowane są wszelkie pośrednie wyniki procesu syntezy.

Katalogiem szczególnym jest *generic*. W nim składowane są pliki powszechnego zastosowania współdzielone pomiędzy projektami. Sam nie jest projektem. W tym rozdziale zostaną wyszczególnione wszystkie źródła z tego właśnie katalogu.

3.2 Synteza

3.2.1 Licznik

Zastosowanie licznika powtarza się najczęściej. Zlicza on ilość wysokich stanów podawanego sygnału *sig* w taktach zegara $CLKB$, ale tylko gdy flagę pracy *en* ma włączoną. Podnosi flagę $full$ w chwilach przepełnień. Można go resetować w trakcie pracy, wtedy zaczyna zliczać od zera. Aktualny stan licznika wyprowadza połączeniem *cnt* do modułu nadrzędnego dla szczegółowszych porównań.

Listing 15: Licznik.v

```
module Counter #(
        parameter MAX=4,
        parameter K=1,
        parameter DELAY=0,
        parameter WIDTH=32
) (
        input           CLKB,
        // counter
        input           en,  // if high, then counter is enabled and is counting
        input           rst, // set counter register to zero
        input           sig, // signal which is counted
        output reg [WIDTH-1:0] cnt = {WIDTH{1'b0}},
        output          full // one pulse if counter is full
);

   always @(posedge CLKB)
      if(rst)
         cnt <= 0;
      else if(en & sig)
         if(cnt < MAX)
            cnt <= cnt + K;
         else
            cnt <= DELAY;

   assign full = (cnt == MAX);

endmodule
```

3.2.2 Drgania styków

Przyciski na płytce są mechaniczne. Pojedyncze wciśnięcie jest obarczone początkowymi drganiami styków, co dla FPGA jest wielokrotnym wciśnięciem tego samego przycisku w bardzo krótkich odstępach czasu. *Debouncer* zatrzymuje propagację krótkich sygnałów poprzez odmierzenie wymaganego czasu stabilizacji w liczniku. Dla symulacji czas ten jest celowo skrócony.

Listing 16: Debouncer.v

```
module Debouncer (
   input    clk,
   input    rst, // set counter register to zero
   input    sig, // signal which is debouncing
   output   full // one pulse if counter is full
);

   Counter #(
`ifdef SIM
      .MAX(10)
`else
      .MAX(10_000_000)
`endif
   ) counter_ (
      .CLKB(clk),
      .en(1'b1),
      .rst(rst),
      .sig(sig),
      .full(full)
   );
```

```
21
22 endmodule
```

3.2.3 Spowalniacz zegara

Jeśli zegar $50mhz$ jest zbyt szybki, można go spowolnić modułem *ModClk*. Bazuje na liczniku, wystawia momenty zbocza opadającego, narastającego, oraz nowy sygnał powolnego zegara o połowie wypełnienia.

Listing 17: ModClk.v

```
1  module ModClk #(
2      parameter DIV=2
3  ) (
4      input CLK50MHZ,
5      input rst,
6      output clk_hf, //half filled 50%
7      output neg_trig,
8      output pos_trig
9  );
10
11 `include "log2.v"
12
13     localparam WIDTH = log2(DIV);
14
15     wire [WIDTH-1:0] cnt;
16     Counter #(
17         .MAX(DIV-1),
18         .WIDTH(WIDTH)
19     ) counter_ (
20         .CLKB(CLK50MHZ),
21         .en(1'b1),
22         .sig(1'b1),
23         .rst(rst),
24         .cnt(cnt)
25     );
26
27     assign clk_hf = (cnt > (DIV-1)/2);
28     assign neg_trig = (cnt == (DIV-1)%DIV);
29     assign pos_trig= (cnt == (DIV-1)/2);
30
31 endmodule
```

3.2.4 Rejestr przesuwny

Rejestr przesuwny zapisuje daną wejściową z interfejsu równoległego do swojego wewnętrznego rejestru w momencie, gdy zauważy podniesioną flagę *set*. Jeśli flagi włączenia *en* oraz działania *tick* są ustawione, wtedy moduł rejestr przesuwa w lewo, wstawiając dostarczony bit rx w puste miejsce. Wyjściowy bit tx zawsze wskazuje na bit z końca rejestru.

Listing 18: Shiftreg.v

```
1  module Shiftreg #(
2      parameter WIDTH=8
3  ) (
4      input  CLKB,
5      // shiftreg
6      input  en,
```

```
 7     input  set, // setting shiftreg value to data_in if spi_trig occurs
 8     input  tick, // register shifting is syncronized with tick signal
 9     input  rx,
10     output tx,
11     input  [WIDTH-1:0] data_in,
12     output [WIDTH-1:0] data_out
13 );
14
15     reg [WIDTH-1:0] shiftreg = {WIDTH{1'b0}};
16     always @(posedge CLKB) begin
17         if(set)
18             shiftreg <= data_in;
19         else if(en & tick)
20     shiftreg <= { shiftreg[WIDTH-2:0], rx };
21     end
22
23     assign tx = shiftreg[WIDTH-1];
24     assign data_out = shiftreg;
25
26 endmodule
```

3.2.5 Wykrywacz zbocza

Wykrywanie zbocza zrealizowane jest przy użyciu rejestru przesuwnego stale zapamiętującego dwie ostatnie wartości śledzonego sygnału w takt podanego zegara. Flagi *pos* i *neg* sygnalizują zbocze przy wykryciu właściwego wzorca.

Listing 19: Edge_Detector.v

```
1  module Edge_Detector(
2      input  clk,
3      input  signal,
4      output pos,
5      output neg
6      );
7
8     // Record last 2 states of signal
9
10    wire [1:0] last2;
11    Shiftreg #(
12        .WIDTH(2)
13    ) shiftreg_ (
14              .CLKB(clk),
15              .en(1'b1),
16              .set(1'b0),
17              .tick(1'b1),
18              .rx(signal),
19              .data_in(2'b11),
20              .data_out(last2)
21              );
22
23    // Detect negative or positive edge
24
25    assign pos = ( last2 == 2'b01 );
26    assign neg = ( last2 == 2'b10 );
27
28 endmodule
```

3.2.6 Serializacja

Moduł dane serializuje, to jest dane dostarczone szyną równoległą przesyła kolejno bit po bicie pojedynczą linią w takt oferowanego zegara. Równocześnie wykonuje operację odwrotną. *Serial* skomponowany jest z poznanych modułów licznika i rejestru przesuwnego, ale uzupełnia je o synchroniczne zakończenie pracy.

Listing 20: Serial.v

```
module Serial  #(
    parameter WIDTH=32
) (
    input RST,
    input CLKB,
    // serial module interface
    input rx,
    output tx,
    input [WIDTH-1:0] data_in,
    output [WIDTH-1:0] data_out,
    input trig,
    output reg ready = 1'b1,
    input tick
);

    wire sent_all_bits;
    Counter #(
        .MAX(WIDTH-1)
    ) Counter_bits (
        .CLKB(CLKB),
        // counter
        .en(1'b1),   // if high counter is enabled and is counting
        .rst(ready), // set counter register to zero
        .sig(tick),  // signal which is counted; counts ticks
        .full(sent_all_bits) // one pulse if counter is full
    );

    Shiftreg #(
        .WIDTH(WIDTH)
    ) Shiftreg_ (
        .CLKB(CLKB),
        // shiftreg
        .en(~ready),
        .set(trig), // setting shiftreg value to data_in if trig occurs
        .tick(tick),
        .rx(rx),
        .tx(tx),
        .data_in(data_in),
        .data_out(data_out)
    );

    always @(posedge CLKB)
    if(RST)
        ready <= 1'b1;
    else if(sent_all_bits && tick)
        ready <= 1'b1;
    else if(trig)
        ready <= 1'b0;

endmodule
```

3.2.7 SPI

Serial Peripheral Interface jest popularnym sprzętowym interfejsem komunikacji. Jest on wyjaśniony w części poświęconej DAC-owi. Wykorzystuje moduł serializacji dodając jedynie funkcjonalność linii Chip Select.

Listing 21: Spi.v

```
module Spi #(
    parameter WIDTH=32
) (
    input RST,
    input CLKB,
    // spi lines
    output spi_sck,
    output spi_cs,
    input spi_miso,
    output spi_mosi,
    // spi module interface
    input [WIDTH-1:0] data_in,
    output [WIDTH-1:0] data_out,
    input trig,
    output ready,
    input clk,
    input tick
);

    wire ready_;
    assign ready = ready_;
    Serial #(
        .WIDTH(WIDTH)
    ) Serial_ (
        .CLKB(CLKB),
        .RST(RST),
        // serial module interface
        .rx(spi_miso),
        .tx(spi_mosi),
        .data_in(data_in),
        .data_out(data_out),
        .trig(trig),
        .ready(ready_),
        .tick(tick)
    );

    assign spi_cs = ready_;
    assign spi_sck = (~ready_) ? clk : 1'b0;

endmodule
```

3.2.8 Generator impulsów

Zadaniem generatora impulsów jest wytwarzanie impulsów jak najbliższych pożądanemu okresowi.

Listing 22: BaudRateGenerator.v

```
module BaudRateGenerator
#(
    parameter INC = 100,
    parameter N = 10
```

```
5 ) (
6     input  CLK50MHZ,
7     input  RST,
8     input  en,
9     output tick
10 );
11
12     reg [N:0] acc = {N{1'b0}};
13     always @(posedge CLK50MHZ)
14         if(RST) acc <= {N{1'b0}};
15         else if(en) acc <= acc[N-1:0] + INC;
16
17     assign tick = acc[N];
18
19 endmodule
```

3.2.9 Odwracacz bitów

Odwrócenie kolejności bitów w rejestrze wymaga automatycznego wygenerowania kodu. Niedogodność języka została schowana w module *Bits_Reverse*.

Listing 23: Bits_Reverse.v

```
1  module Bits_Reverse #(
2      parameter WIDTH = 8
3  ) (
4      input  [WIDTH-1:0] orginal,
5      output [WIDTH-1:0] reversed
6  );
7
8      // Reverse bits order
9      genvar        i;
10     generate
11         for(i=0; i<WIDTH; i=i+1)
12         begin
13     assign reversed[i] = orginal[WIDTH-1-i];
14         end
15     endgenerate
16
17 endmodule
```

3.2.10 Funkcja logarytmiczna

Funkcja ta jest bardzo pomocna w ustaleniu szerokości deklarowanego rejestru na podstawie górnego zakresu liczb przez niego zapamiętywanych. Wartość funkcji obliczana jest na etapie elaboracji.

Listing 24: log2.v

```
1 //constant function calculetes value at collaboration time
2 //source http://www.beyond-circuits.com/wordpress/2008/11/constant-functions/
3 function integer log2;
4     input integer value;
5     begin
6         value = value-1;
7         for (log2=0; value>0; log2=log2+1)
8           value = value>>1;
9     end
```

```
10 endfunction
```

3.3 Symulacja

3.3.1 Zegar

Jest to moduł niezbędny do symulacji, formuje prostokątną linie zegarową.

Listing 25: Clock.v

```
1  module Clock #(
2      //DELAY 10ns - clock 50MHZ
3      parameter DELAY = 10
4  ) (
5      output reg clk
6  );
7
8      initial clk = 0;
9      always #DELAY clk <= ~clk;
10
11 endmodule
```

3.3.2 Reset

Moduł w początkowej chwili na krótko podnosi linię, w zamierzeniu sygnału resetu.

Listing 26: Reset.v

```
1  module Reset #(
2      parameter DELAY = 40
3  ) (
4       output reg RST
5  );
6
7      initial begin
8          RST = 0;
9          #10;
10         RST = 1;
11         #DELAY;
12         RST = 0;
13     end
14
15 endmodule
```

3.3.3 Ustawiacz

Moduł przyjmuje grupę sygnałów przy inicjalizacji, po czym udostępnia zadania operujące na nich i informujące o swoich działaniach z zadaną dokładnością.

Listing 27: Set.v

```
1  module Set
2  #(
3      // LOGLEVEL = 0
4      //       bez zadnych komunikatow
5      // LOGLEVEL = 1
6      //       bledy
7      // LOGLEVEL = 2
```

```
8     //       ostrzezenia
9     //
10    // LOGLEVEL = 3
11    //       informuje o zamiarze zmiany sygnalu
12    // LOGLEVEL = 4
13    //       informuje o zmianie sygnalu
14    // LOGLEVEL = 5
15    //       informuje o zamiarze przeczekiwania
16    // LOGLEVEL = 6
17    //       informuje o przeczekaniu
18    parameter LOGLEVEL = 1,
19
20    parameter N = 1
21 ) (
22    output reg [N-1:0] signals
23 );
```

Zadanie *state* ustawia podany stan na sygnałach i loguje swoje zamiary.

Listing 28: Set.v

```
25    // Zadanie ustawia okreslony stan
26    task state
27    (
28       input [N-1:0] new_signals
29    );
30       begin
31
32          // Poinformuj o stanie poprzednim
33          if( LOGLEVEL >= 3 )
34             $display("%t\t INFO3\t [ %m ] \t Stan sygnalow zostanie zmieniony.
                  Obecny stan '%b' (0x %h), spodziewany stan '%b' (0x %h)", $time,
                  signals, signals, new_signals, new_signals);
35
36          // Zmien stan
37          signals = new_signals;
38
39          // Poinformuj o zmianie
40          if( LOGLEVEL >= 4 )
41             $display("%t\t INFO4\t [ %m ] \t Stan sygnalow zostal zmieniony.
                  Obecny stan '%b' (0x %h)", $time, signals, signals);
42
43       end
44    endtask
```

Zadanie *low* uziemi sygnały. Prawie identycznie wyglądają *high* oraz *z* dla wysokiej impedancji.

Listing 29: Set.v

```
47    // Zadanie ustawia stan niski na wszystkich liniach
48    task low ();
49       begin
50          state( {N{1'b0}} );
51       end
52    endtask
```

Do ustawiania zadanego stanu na określony czas służy *state_during*. Istnieją też poręczniejsze, wyspecjalizowane zadania *low_during*, *high_during*, *z_during*.

Listing 30: Set.v

```
71      // Zadanie ustawia zadany stan i przeczekuje zadany okres czasu
72      task state_during
73      (
74          input [31:0] period,
75          input [N-1:0] new_signals
76      );
77          begin
78
79              // Ustaw stan sygnalow na zadany
80              state( new_signals );
81
82              // Opcjonalnie poinformuj o przeczekiwaniu
83              if( LOGLEVEL >= 5 )
84                  $display("%t\t INFO5\t [ %m ] \t Sygnal w stanie %b (hex %h) zostaje
                        zamrozony przez zadany okres czasu %d", $time, new_signals,
                        new_signals, period);
85
86              // Przeczekaj zadany czas
87              #period;
88
89              // Opcjonalnie poinformuj o dalszym biegu
90              if( LOGLEVEL >= 6)
91                  $display("%t\t INFO6\t [ %m ] \t Przeczekiwanie stanu %b (hex %h)
                        zostalo zakonczone po %d", $time, new_signals, new_signals,
                        period);
92
93          end
94      endtask
```

Ostatnia grupa zadań z modułu *Set* ustawia żądany stan, przeczekuje zadany czas, po czym przywraca pierwotny stan sygnałów. Oczywiście także tutaj występują specjalizacje w postaci *low_during_and_restore*, *high_during_and_restore*.

Listing 31: Set.v

```
71      // Zadanie ustawia okreslony stan i przeczekuje zadany okres czasu, po czym
            wraca do stanu poprzedniego
72      task state_during_and_restore
73      (
74          input [31:0]  period,
75          input [N-1:0] new_signals
76      );
77          reg [N-1:0]   saved_signals;
78          begin
79              saved_signals = signals;
80              state( new_signals );
81              #period;
82              state( saved_signals );
83          end
84      endtask
```

3.3.4 Monitor

Monitor dostaje grupę sygnałów i oferuje zadania do prześledzenia niezmienności ich stanów.

Listing 32: Monitor.v

```
0  module Monitor
1  #(
```

```
2     // LOGLEVEL = 0
3     //       bez zadnych komunikatow
4     // LOGLEVEL = 1
5     //       bledy
6     // LOGLEVEL = 2
7     //       ostrzezenia
8     //
9     // LOGLEVEL = 3
10    //       informuje o stalosci przeczekanego sygnalu
11    // LOGLEVEL = 4
12    //       informuj o oczekiwaniu na przyjecie przez sygnal zadanej wartosci
13    // LOGLEVEL = 5
14    //       informuj o zastaniu spodziewanego stanu sygnalow
15    // LOGLEVEL = 6
16    //       zrzuca stan monitorowanej linii z kazdej chwili czasu
17    parameter LOGLEVEL = 1,
18
19    // Szerokosc badanej szyny sygnalowej
20    //
21    parameter N = 1
22 ) (
23    input [N-1:0] signals
24 );
```

Można upewnić się czy sygnały znajdują się w spodziewanych stanach. Występują specjalizacje *ensure_low*, *ensure_high*, *ensure_z*.

Listing 33: Monitor.v

```
27    // Zadanie sprawdza, czy sygnaly sa w zadanym stanie
28    task ensure_state
29    (
30       input [N-1:0] expected_signals,
31       output ensurance
32    );
33       begin
34
35          // Jesli sygnaly sa zgodne z oczekiwaniami, wystawi jedynke
36          ensurance = 1'b1;
37
38          if( signals !== expected_signals ) begin
39
40             // Sygnaly sie roznia, wystaw zero
41             ensurance = 1'b0;
42
43             // Zglos blad
44             if( LOGLEVEL >= 1 )
45                $display("%t\t ERROR\t [ %m ] \t Sygnaly nie sa zgodne z
                      oczekiwaniami. Stan obecny '%b' (0x %h), spodziewany '%b' (0x
                      %h)", $time, signals, signals, expected_signals,
                      expected_signals);
46
47          end
48
49          // Zakomunikuj o zastaniu spodziewanych stanow
50          if( ensurance )
51             if( LOGLEVEL >= 5 )
52                $display("%t\t INFO5\t [ %m ] \t Sygnaly sa zgodne z
                      oczekiwaniami. Stan oczekiwany '%b' (0x %h)", $time,
                      expected_signals, expected_signals);
```

```
53
54          end
55      endtask
```

Do weryfikacji czy sygnały nie zmienią się przez podany czas służy zadanie *ensure_same_during*.

Listing 34: Monitor.v

```
90      // Zadanie bada stalosc zadanych sygnalow w ustalonym przedziale czasu
91      task ensure_same_during
92      (
93          input [31:0] period,
94          output       ensurance
95      );
96          integer          i;
97          reg [N-1:0] saved_signals;
98          begin
99
100             // Jesli stan linii pozostanie bez zmian, wystawi jedynke
101             ensurance = 1'b1;
102
103             // Zapisz stan linii z momentu rozpoczecia tego zadania
104             saved_signals = signals;
105
106             // Monitoruj linie przez zadany czas lub do momentu pierwszej zmiany
                    stanu
107             for(i=0; i<period && ensurance; i=i+1) begin
108                 if(signals !== saved_signals) begin
109
110                     // Sygnal sie zmienil podczas badania, zakoncz zerem
111                     ensurance = 1'b0;
112
113                     // Zglos blad
114                     if( LOGLEVEL >= 1 )
115                         $display("%t\t ERROR\t [ %m ] \t Nastapila nieoczekiwana
                                zmiana stanu monitorowanej linii po czasie %d 000 / %d 000.
                                 Stan obecny '%b' (0x %h), spodziewany '%b' (0x %h)", $time
                                , i, period, signals, signals, saved_signals, saved_signals
                                );
116
117                 end
118
119                 // Wypisz wszystkie iteracje petli na zyczenie ostatniego poziomu
                        logowania
120                 if( LOGLEVEL >= 9 )
121                     $display("%t\t INFO9\t [ %m ] \t Stan linii '%b' (0x %h) zapisana
                            '%b' (0x %h) czas %d 000", $time, signals, signals,
                            saved_signals, saved_signals, i);
122
123                 // Przejdz do nastepnego kroku czasowego
124                 #1;
125
126             end
127
128             // Zakomunikuj o oczekiwanej stalosci sygnalu w zadanym czasie
129             if( ~ensurance )
130
131                 if( LOGLEVEL >= 3 )
132                     $display("%t\t INFO3\t [ %m ] \t Stan '%b' (0x %h) linii zgodnie
                            z oczekiwaniami nie zmienil sie po czasie %d 000", $time,
                            signals, signals, i);
```

```
133
134         end
135     endtask
```

Do weryfikacji czy sygnały są w spodziewanym stanie oraz czy będą w nim tkwić przez podany czas, dostępne jest zadanie *ensure_state_during*. Są także doprecyzowane wersje *ensure_low_during*, *ensure_high_during*, *ensure_z_during*.

Listing 35: Monitor.v

```
138     // Zadanie oczekuje okreslonego stanu badanych linii w nadzorowanym okresie
139     task ensure_state_during
140     (
141         input [31:0] period,
142         input [N-1:0] expected_signals,
143         output ensurance
144      );
145         reg     same;
146         begin
147
148             if( LOGLEVEL >= 7 )
149                 $display("%t\t INFO7\t [ %m ] \t Sprawdzanie czy obecny stan
                        sygnalow '%b' (0x %h) bedzie niezmienny i zgodny z oczekiwanym
                        wzorcem '%b' (0x %h) przez zadany czas '%d'", $time, signals,
                        signals, expected_signals, expected_signals, period);
150
151             // Jesli linie pozostaly w zadanym stanie przez okres proby, potwierdzi
                    jedynka
152             ensurance = 1'b1;
153
154             // Sprawdz czy od poczatku wystapil stan wzorcowy
155             if( signals != expected_signals ) begin
156
157                 // Sygnaly roznia sie od wzorca od poczatku, zwroc zero
158                 ensurance = 1'b0;
159
160                 // Zlos blad
161                 if( LOGLEVEL >= 1 )
162                     $display("%t\t ERROR\t [ %m ] \t Wszystkie sygnaly juz od
                            poczatku roznia sie od wzorca '%b' (0x %h), natomiast
                            wystapily '%b' (0x %h)", $time, expected_signals,
                            expected_signals, signals, signals);
163
164             end
165
166             // Jesli sygnaly zaczely jako zgodne ze wzorcem, dopilnuj ich
                    niezmiennosci w badanym czasie
167             ensure_same_during(period, same);
168             if( ensurance && ~same ) begin
169
170                 // Co najmniej jedna linia sie poroznila, zwroc zero
171                 ensurance = 1'b0;
172
173                 // Zglos blad
174                 if( LOGLEVEL >= 1 )
175                     $display("%t\t ERROR\t [ %m ] \t Sygnaly '%b' (0x %h)
                            nieoczekiwanie rozminely sie ze wzorcem '%b' (0x %h)", $time,
                            signals, signals, expected_signals, expected_signals);
176             end
177
```

```
178         end
179     endtask
```

W przypadku chęci zaczekania na ustalenie się pożądanego stanu, można wywołać zadanie *wait_for_state*, lub jego konkretniejsze wersje *wait_for_low*, *wait_for_high*, *wait_for_z*.

Listing 36: Monitor.v

```
231     // Zadanie czeka na zadany stan
232     task wait_for_state
233     (
234         input [N-1:0] expected_signals
235     );
236         integer         i;
237         // reg      [1023:0]       a;
238         begin
239             i = 0;
240 
241             // Poinformuj o oczekiwaniu na zadany stan
242             if( LOGLEVEL >= 4 )
243                 $display("%t\t INFO4 \t [ %m ] \t Oczekiwanie na przyjecie stanu '%b
                        ' (0x %h). Stan obecny '%b' (0x %h)", $time, expected_signals,
                        expected_signals, signals, signals);
244 
245             // Oczekuj zadanego stanu
246             while( signals !== expected_signals ) begin
247                 i = i+1;
248                 #1;
249             end
250 
251             // Poinformuj o ustaleniu sie zadanego stanu
252             if( LOGLEVEL >= 4 )
253                 $display("%t\t INFO4\t [ %m ] \t Oczekiwany stan  '%b' (0x %h)
                        ustalil sie po czasie %d 000", $time, expected_signals,
                        expected_signals, i);
254 
255         end
256     endtask
```

4 DAC

Zadaniem konwertera cyfrowo-analogowego jest przetwarzanie przekazanych mu kolejnych liczb binarnych na ich analogowe odpowiedniki realizowane jako wartość napięcia na jego pinie wyjściowym w zakresie napięcia maksymalnego V_{ref}.

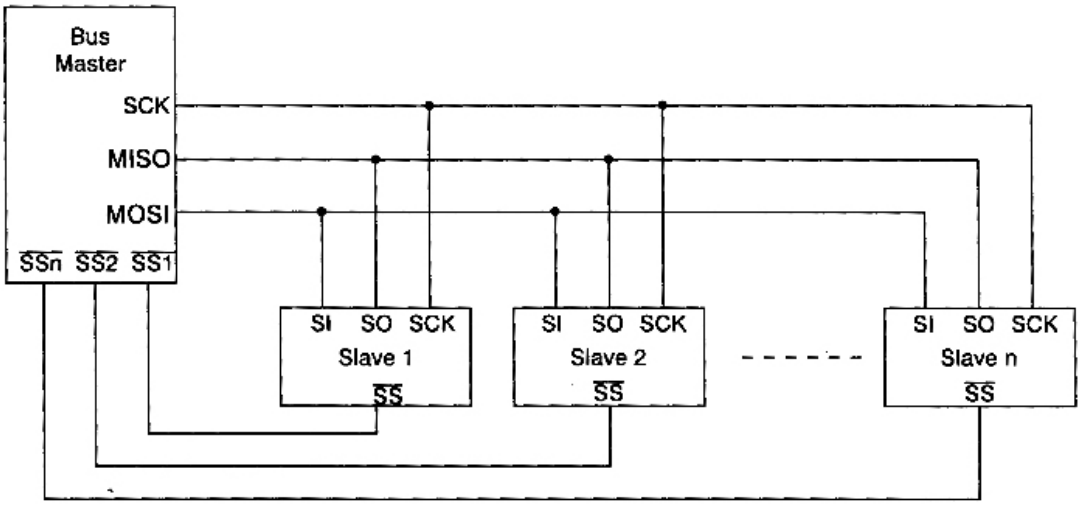
Obsługiwany przez przetwornik zakres liczb binarnych jest dokładnością przetwornika. Występujący na płytce układ scalony LTC2624 ma zatopione 4 przetworniki DAC o dokładności 12-bitowej. Wszystkie przetworniki domyślnie mają wartość napięcia maksymalnego $V_{ref} = 3.3V$, jednak dla dwóch z nich wartość tą można indywidualnie ustawić komunikując się z układem wzmacniaczy zawartych w kostce LP3906. Wartości napięć wyjściowych podaje wzór

$$V_{out} = \frac{D[11:0]}{4096} V_{ref}$$

4.1 Komunikacja

4.1.1 SPI

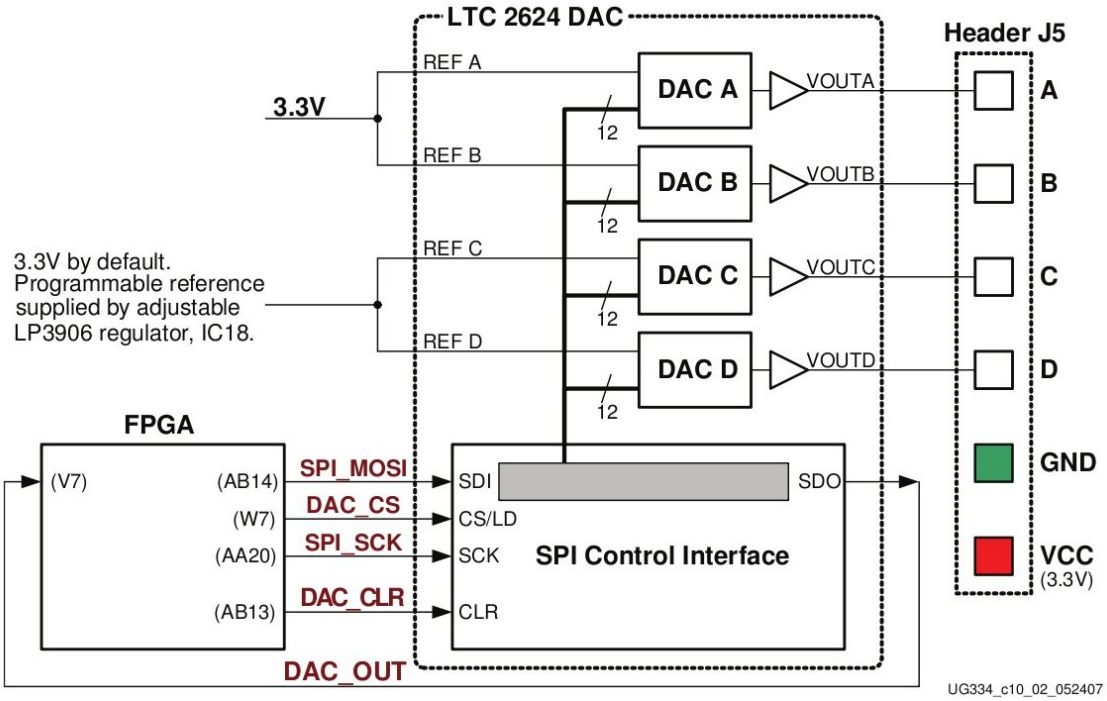
Układ LTC2624 zaimplementowaną ma logikę komunikacji w standardzie magistrali Serial Peripheral Interface.



Rysunek 2: SPI

Na magistrali występuje jeden układ nadrzędny - Master oraz co najmniej jeden Slave. Master generuje zegar na linii SCK. Master wysyła dane do Slavów szeregowo linią MOSI, Slavy odsyłają dane linią MISO. Transmisja jest fullduplexowana - przesył w obu kierunkach poszczególnych bitów następuje równocześnie w takt zegara. Między układami współdzielone są linie zegara oraz danych. Odseparowane natomiast są linie CS poszczególnych slavów - wywołanie niskiego potencjału przez mastera aktywuje danego slava do uczestnictwa w wymianie danych.

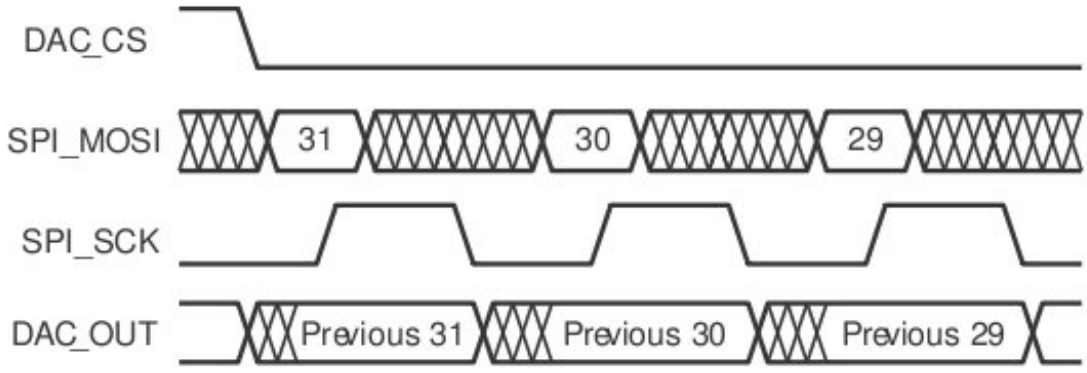
4.1.2 Połączenia



Rysunek 3: Schemat polaczen ukladow LTC2624 i FPGA

Przed pierwszym użyciem należy układ zresetować chwilowo obniżając stan linii DAC_CLR.

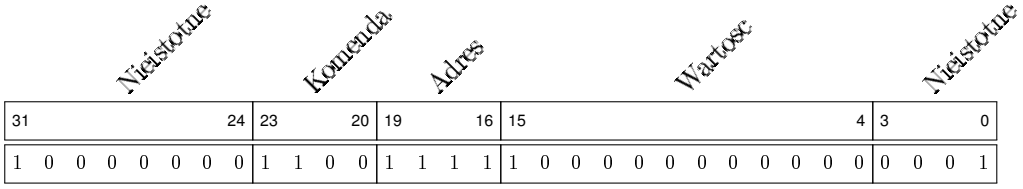
Na linie SPI_SCK należy podać zegar o częstotliwości nie przekraczającej 50Mhz. Obniżając stan linii DAC_CS rozpoczynamy komunikację z układem. Wtedy w takt zegara przesyłamy szeregowo do niego kolejne bity danych linią SPI_MOSI. LTC2624 ładuje kolejne przesyłane bity do swojego rejestru przesuwnego na narastającym zboczu zegara oraz zwraca swoją poprzednią zawartość linią DAC_OUT na opadającym zboczu. Natychmiast po wysłaniu kompletu danych należy koniecznie podnieść stan linii DAC_CS zakańczając transmisję.



Rysunek 4: Wykres stanow linii

4.1.3 Protokół komunikacji

Register 4.1: Przesylana ramka (a)



Dane przesyłane do układu LTC2624 są 32-bitową ramką uwidocznioną powyższym polem bitowym. Bity wysyła się kolejno zaczynając od najstarszego. Transmisja zaczyna się ośmioma nic nie znaczącymi bitami. Po nich wysyłane jest 4-bitowe pole komendy - typowo o wartości 0011, co oznacza natychmiastowe wystawienie zadanej wartości napięcia. Następnie podawany jest adres konwertera według poniższej tabeli

19			16	Adres
0	0	0	0	DAC A
0	0	0	1	DAC B
0	0	1	0	DAC C
0	0	1	1	DAC D
1	1	1	1	Wszystkie

Trzecie pole jest 12-bitową wartością binarną odpowiadającą wystawianemu napięciu. Ramka kończy się 4 nieistotnymi bitami. W przykładzie na wszystkich konwerterach pojawiłaby się połowa z ustawionych zakresów napięć.

4.2 Aplikacja

Zrealizowany projekt wykorzystuje opisany układ obecny na płytce przetwornika cyfrowo-analogowego. Po wystartowaniu lub resecie ustawiane jest zerowe napięcie. Używając dwóch przycisków można napięcie podnieść lub obniżyć w zakresie $0V - 3.3V$ z krokiem $32/4096$. Zawsze pojedynczy pakiet ustawia wszystkie cztery konwertery dostępne w układzie na raz. Podgląd na bieżący stan podawanego napięcia pokazują diody LED.

4.3 Synteza

4.3.1 Warstwa wierzchnia

Moduł najwyższy łączy interfejs SPI przetworników z odfiltrowanymi z drgań sygnałami przycisków i diodami led. Instancjuje także moduł kontrolera i warstwę spajającą interfejs DAC-a z wewnętrznym SPI.

Listing 37: Top.v

```
1 module Top (
2     input CLK50MHZ,
3     input RST,
4     // dac
5     output SPI_SCK,
```

```
 6     output DAC_CLR,
 7     output DAC_CS,
 8     output SPI_MOSI,
 9     input DAC_OUT,
10     // control
11     input BTN_WEST,
12     input BTN_EAST,
13     input BTN_NORTH,
14     output [7:0] LED
15 );
```

4.3.2 Kontroler

Kontroler posiada połączenia do przycisków *RST* i odfiltrowanych z drgań *less*, *more* oraz *maxx*. W zależności od ich wciśnięć, kontroler wystawi nową wartość *data* dostępną dla DAC-ów oraz zawiadamia o tym flagą *dactrig*. Zawsze adresowane są wszystkie DAC-i z komendą natychmiastowego ustawienia nowej wartości. Podgląd aktualnie wystawianej wartości pokazują diody LED.

Listing 38: Controller.v

```
1  module Controller(
2     input RST,
3     input CLK50MHZ,
4     // verilog module interface
5     output      [11:0] data,
6     output reg  [3:0 ] address = 4'b1111,
7     output reg  [3:0 ] command = 4'b0011,
8     output             dactrig,
9     // control
10    input              less,
11    input              more,
12    input              maxx,
13    // leds
14    output      [7:0 ] LED
15 );
```

Moduł ma wewnętrzny rejestr przechowujący bieżący stan wystawianego napięcia. Wciskane przyciski go modyfikują.

Listing 39: Controller.v

```
17    reg [7:0] d = 8'd0;
18    always @(posedge CLK50MHZ)
19       if(RST)        d <= 8'd0;
20       else if(maxx)  d <= 8'hff;
21       else if(less)  begin if( 0 < d ) d <= d - 1; end
22       else if(more)  begin if( ~&  d ) d <= d + 1; end
23
24    assign data = { d, 4'h0 };
25    assign LED = d;
```

Pojawienie się nowej wartości należy obwieścić modułowi nadrzędnemu.

Listing 40: Controller.v

```
27    assign dactrig = (less || more || maxx);
```

4.3.3 DacSpi

Jest to warstwa zajmująca się opóźnieniem zegara bazowego oraz doprecyzowaniem ogólnego modułu SPI. $50mHz$ to górna granica działania kostki DAC-ów. Dla pewniejszego działania, zalecane jest niższe taktowanie, co zrealizowane jest modułem *ClkMod* z podanym parametrem dzielnika $DIV = 6$. Do SPI przekazane zostaje 32-bitowe połączenie spojone z danych uzyskanych od kontrolera i otoczone bitami nieznaczącymi.

Listing 41: Controller.v

```
32    wire [WIDTH-1:0] dacdatatosend = {8'h80, command, address, data, 4'h1};
```

Zegar magistrali SPI taktowany jest tylko w razie konieczności. Resetowanie DAC-ów następuje odwrotnie do przycisku reset.

Listing 42: Controller.v

```
51    assign SPI_SCK = (~dacdone) ? spi_sck : 1'b0;
52    assign DAC_CLR = ~RST;
```

4.4 Symulacja

W teście inicjalizowane są zegar, reset, moduł syntezowalny, zachowawczy i przypadek testowy.

4.4.1 Przypadek testowy

W przypadku testowym powoływane są ustawiacze przycisków zwiększania oraz zmniejszania napięcia, a także monitor linii resetu. Symulacja czeka na zresetowanie układu, następnie poprzez przyciski dwukrotnie zwiększa napięcie i raz je zmniejsza.

4.4.2 DAC zachowawczo

Interfejs modułu ukazuje poziomy logowania wraz z liniami upozorowanej kostki.

Listing 43: DacLTC2624Behav.v

```
0  module DacLTC2624Behav
1  #(
2      // LOGLEVEL = 0
3      //    bez zadnych komunikatow
4      // LOGLEVEL = 1
5      //    pokazuje bledy
6      // LOGLEVEL = 2
7      //    pokazuje ostrzezenia
8      //
9      // LOGLEVEL = 3
10     //    informuje o odbieraniu danych, odebraniu i ich interpretacji
11     // LOGLEVEL = 4
12     //    informuje o wlasciwej liczbie bitow w pakiecie
13     // LOGLEVEL = 5
14     //    informuje o adresie daca
15     // LOGLEVEL = 6
16     //    informuje o zakonczeniu odbioru danych (moment podniesienia flagi DAC_CS
           )
17     // LOGLEVEL = 7
18     //    informuje o odbieraniu poszczegolnych pol pakietu
19     // LOGLEVEL = 8
20     //    informuje o odbiorze poszczegolnych bitow pakietu
```

```
21      // LOGLEVEL = 9
22      //    informuje o przebiegu resetowania daca
23      parameter LOGLEVEL       = 5,
24      parameter LOGLEVEL_SCK   = 3,
25      parameter LOGLEVEL_CLR   = 3,
26      parameter LOGLEVEL_MOSI  = 3,
27      parameter LOGLEVEL_SCK_MOSI = 3
28  ) (
29      input  SPI_SCK,
30      input  DAC_CS,
31      input  DAC_CLR,
32      input  SPI_MOSI,
33      output DAC_OUT
34  );
```

Moduł zachowawczy sprawdza, czy użytkownik zresetował kostke przed użyciem. Informacja ta zawarta zostaje w rejestrze *inited*. Wątek powróci przy odebraniu ramki.

Listing 44: DacLTC2624Behav.v

```
63      // Przed uzyciem daca nalezy go najpierw zresetowac poprzez chwilowe
            obnizenie linii DAC_CLR
64      reg     inited = 1'b0;
65      initial begin
66
67          if(LOGLEVEL >= 9)
68              $display("%t\t INFO9\t [ %m ] \t Oczekiwanie na zresetowanie daca",
                    $time);
69          monitor_clr.wait_for_low();
70          monitor_clr.ensure_low_during( 40 );
71          monitor_clr.wait_for_high();
72
73          if(LOGLEVEL >= 9)
74              $display("%t\t INFO9\t [ %m ] \t Zresetowano daca", $time);
75          inited = 1'b1;
76      end
```

Kalkulowany jest indeks przesyłanego bitu.

Listing 45: DacLTC2624Behav.v

```
78      // Rejest zlicza kolejno odbierane bity
79      reg [5:0] idx = 6'd0;
80      // Licznik idx zerowany jest na poczatku kazdej transmisji
81      always @(negedge DAC_CS)
82          idx <= 6'd0;
83      // Resetuj licznik lub go podbij na kazdym narastajacym zboczu zegara
84      always @(posedge SPI_SCK) begin
85          idx <= idx + 1;
86      end
```

Zadanie odbioru pojedynczego bitu upewnia się, że wartość bitu jest stabilna w zczytywanym oknie czasowym.

Listing 46: DacLTC2624Behav.v

```
88      // Odbiera jeden bit
89      task receive_bit
90      (
91          output received_bit
92      );
```

```
93          begin
94              if(LOGLEVEL >= 8)
95                  $display("%t\t INFO8\t [ %m ] \t Odbieranie kolejnego bitu", $time);
96
97      // Po wlaczeniu daca do szyny spi zegar powinien zaczac nisko
98              monitor_sck.ensure_low_during( 40 );
99              monitor_sck.wait_for_high();
100
101     // 4ns stabilnosci zegara i zczytywanej linii miso
102     monitor_sck_mosi.ensure_state_during( 4 );
103     received_bit = SPI_MOSI;
104
105     // konczenie okresu zegara
106     monitor_sck.ensure_high_during( 40 - 4 );
107     monitor_sck.wait_for_low();
108
109             if(LOGLEVEL >= 8)
110                 $display("%t\t INFO8\t [ %m ] \t Odebrano bit %b", $time,
                        received_bit);
111         end
112     endtask
```

Odbieranie całego pakietu podzielone jest na mniejsze fragmenty w celu dokładniejszego logowania komunikacji.

Listing 47: DacLTC2624Behav.v

```
114     // Pola bitowe otrzymanych danych
115     reg [ 3:0] dontcare4 = 4'd0;
116     reg [11:0] value     = 12'd0;
117     reg [ 3:0] address   = 4'd0;
118     reg [ 3:0] command   = 4'd0;
119     reg [ 8:0] dontcare8 = 8'd0;
120     wire [31:0] packet = { dontcare4, value, address, command, dontcare8 };
121     // Odbiera 32 bity danych pakietu
122     task receive_packet
123     ();
124         integer i;
125         begin
126             if(LOGLEVEL >= 3)
127                 $display("%t\t INFO3\t [ %m ] \t Odbieranie pakietu", $time);
128
129     // Receive 8 dont care bits
130             if(LOGLEVEL >= 7)
131                 $display("%t\t INFO7\t [ %m ] \t Odbieranie 8 pierwszych nie
                        znaczacych bitow", $time);
132     for(i = 0; i < 8; i=i+1) begin
133             receive_bit( dontcare8[i] );
134     end
135
136     // Receive command
137             if(LOGLEVEL >= 7)
138                 $display("%t\t INFO7\t [ %m ] \t Odbieranie komendy", $time);
139     for(i = 0; i < 4; i=i+1) begin
140             receive_bit( command[i] );
141     end
142
143     // Receive address of dac
144             if(LOGLEVEL >= 7)
145                 $display("%t\t INFO7\t [ %m ] \t Odbieranie adresu", $time);
```

```
146     for(i = 0; i < 4; i=i+1) begin
147             receive_bit( address[i] );
148     end
149
150     // Receive 12 bit of value
151         if(LOGLEVEL >= 7)
152             $display("%t\t INFO7\t [ %m ] \t Odbieranie wartosci", $time);
153     for(i = 0; i < 12; i=i+1) begin
154             receive_bit( value[i] );
155     end
156
157     // Receive 4 dont care bits
158         if(LOGLEVEL >= 7)
159             $display("%t\t INFO7\t [ %m ] \t Odbieranie 4 ostatnich nie
                    znaczacych bitow", $time);
160     for(i = 0; i < 4; i=i+1) begin
161             receive_bit( dontcare4[i] );
162     end
163
164         if(LOGLEVEL >= 3)
165             $display("%t\t INFO3\t [ %m ] \t Odebrano pakiet", $time);
166        end
167     endtask
```

Obniżenie linii *DAC_CS* rozpoczyna przesył ramki. Tutaj potwierdzane jest uprzednie zresetowanie układu.

Listing 48: DacLTC2624Behav.v

```
169     reg received_packet = 1'b0;
170     // Odbiera pakiet i wyzwala o tym flage
171     always @(negedge DAC_CS)
172         if(!inited) begin
173             if(LOGLEVEL >= 1)
174                 $display("%t\t BLAD\t [ %m ] \t Nie zresetowano ukladu przed
                        nadaniem nadanych", $time);
175         end else begin
176             if(LOGLEVEL >= 3)
177                 $display("%t\t INFO3\t [ %m ] \t Odbieranie danych", $time);
178
179     received_packet = 1'b0;
180     receive_packet();
181     received_packet = 1'b1;
182
183         end
```

Śledzić należy przesłanie zbyt dużej ilości bitów w ramce.

Listing 49: DacLTC2624Behav.v

```
185     // Zatrzaskuje moment przeslania zbyt wielu bitow
186     reg too_many_bits = 1'b0;
187     always @(negedge DAC_CS)
188         too_many_bits = 1'b0;
189     always @(posedge received_packet) begin
190         monitor_sck.wait_for_low();
191         monitor_sck.wait_for_high();
192
193         too_many_bits = 1'b1;
194     end
195     wire received_too_many_bits = received_packet && too_many_bits;
```

Podniesienie linii *DAC_CS* oznacza zakończenie nadawania ramki. Wypisywane są wtedy szczegółowe komunikaty o zaadresowanym DAC-u, podanej komendzie oraz wystawionej wartości, lub błędy transmisji jeśli jakiekolwiek zaistniały.

Listing 50: DacLTC2624Behav.v

```
197     // Podniesienie DAC_CS konczy transmisje, weryfikowane i wypisywane sa
            przeslane dane
198     always @(posedge DAC_CS) begin
199         if(inited) begin
200 
201             // Zakomunikuj koniec odbioru
202             if(LOGLEVEL >= 6)
203                 $display("%t\t INFO6\t [ %m ] \t Podniesiono flage DAC_CS, co konczy
                        odbior danych", $time);
204 
205             //Bledy nieodpowiedniej ilosci odebranych bitow
206             if(received_too_many_bits) begin
207                 if(LOGLEVEL >= 1)
208                     $display("%t\t BLAD\t [ %m ] \t Do daca wyslanych zostalo wiecej
                            bitow niz 32", $time);
209             end else if(idx < 32) begin
210                 if(LOGLEVEL >= 1)
211                     $display("%t\t BLAD\t [ %m ] \t Do daca wyslanych zostalo %d
                            bitow. Nalezy wyslac 32", $time, idx);
212 
213             // Odebrano wlasciwa ilosc bitow, parsuj dane
214             end else begin
215 
216                 if(LOGLEVEL >= 4)
217                     $display("%t\t INFO4\t [ %m ] \t Odebrane dane zawieraja wlasciwa
                            ilosc bitow", $time);
218 
219                     // Wypisz odebrane pola
220                     if(LOGLEVEL >= 3)
221                         $display("%t\t INFO3\t [ %m ] \t Ustawiono\twartosc %d (0x%h)\
                                tna adresie %d (0x%h)\tz komenda %d (0x%h)", $time, value,
                                value, address, address, command, command);
222 
223                     // Wypisz ustawian-ego/-ne dac-a/-i bazujac na przeslanym adresie
                            lub zakomunikuj blad
224                     case(address)
225                         4'b0000:
226                             if(LOGLEVEL >= 5)
227                                 $display("%t\t INFO5\t [ %m ] \t Dac o adresie %b (0x%h)
                                        jest dac-iem A", $time, address, address);
228                         4'b0001:
229                             if(LOGLEVEL >= 5)
230                                 $display("%t\t INFO5\t [ %m ] \t Dac o adresie %b (0x%h)
                                        jest dac-iem B", $time, address, address);
231                         4'b0010:
232                             if(LOGLEVEL >= 5)
233                                 $display("%t\t INFO5\t [ %m ] \t Dac o adresie %b (0x%h)
                                        jest dac-iem C (mozliwe ustawienie wzmocnienia)",
                                       $time, address, address);
234                         4'b0011:
235                             if(LOGLEVEL >= 5)
236                                 $display("%t\t INFO5\t [ %m ] \t Dac o adresie %b (0x%h)
                                        jest dac-iem D (mozliwe ustawienie wzmocnienia)",
                                       $time, address, address);
```

```
237                     4'b1111:
238                         if(LOGLEVEL >= 5)
239                             $display("%t\t INFO5\t [ %m ] \t Dac o adresie %b (0x%h)
                                    odpowiada wszystkim dac-om", $time, address, address
                                    );
240                     default:
241                         if(LOGLEVEL >= 1)
242                             $display("%t\t BLAD\t [ %m ] \t Nieprawidlowy adres daca
                                    ", $time);
243                 endcase
244
245                 // Sprawdz czy wyslano wlasciwa komende, jedyna poprawna to 0011
246                 // Odbierana komenda jest w odwroconym porzadku bitow
247                 if(command != 4'b1100)
248                     if(LOGLEVEL >= 1)
249                         $display("%t\t BLAD\t [ %m ] \t Nieprawidlowa komenda %b (0
                                x%h) - aby natychmiastowo ustawic dac nalezy wyslac 0011
                                 (0x3)", $time, command, command);
250
251             end
252         end
253     end
```

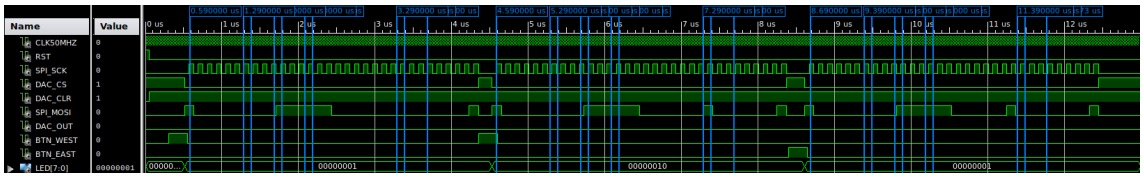
Istotnym aspektem modułu behawioralnego jest zwracanie poprzednio zapamiętanej wartości linią *DAC_OUT*.

Listing 51: DacLTC2624Behav.v

```
255     // Na linii wyjsciowej DAC_OUT beda sie pojawiac kolejne bity wypychane z
            rejestru przesuwnego daca
256     // Nastepuje to na narastajacym zboczu zegara SPI_SCK przy obnizonej linii
            aktywacji transmisji DAC_CS
257     assign DAC_OUT = DAC_CS ? 1'b0 : packet[31];
```

4.4.3 Przebieg

Przedstawiona symulacja pokazuje wykres przebiegów czasowych linii prowadzących do układu DAC. Pola z bitami nieistotnymi przesyłanej ramki są tak dobrane aby pojedynczymi, skrajnymi pikami pokazywały początek i koniec ramki.



Zestaw komunikatów z przebiegu symulacji.

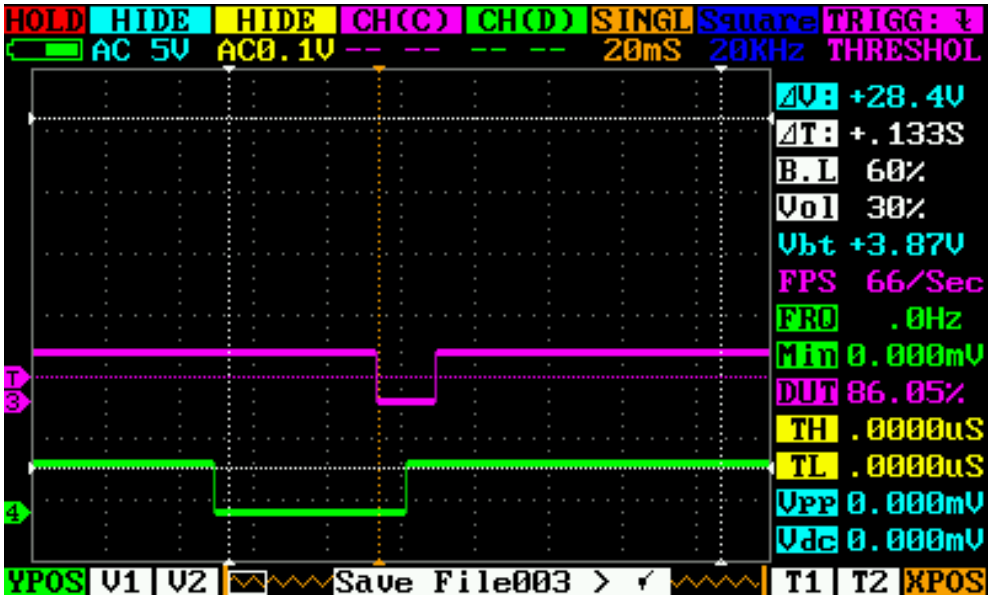
Listing 52: DAC logi

```
1   301000 INFO3 [ TopTest.TopTestBench_ ]  Zwiekszanie napiecia
2   510000 INFO3 [ TopTest.DacLTC2624Behav_ ]  Odbieranie danych
3   510000 INFO3 [ TopTest.DacLTC2624Behav_.receive_packet ]  Odbieranie pakietu
4   4350000 INFO4 [ TopTest.DacLTC2624Behav_ ]  Odebrane dane zawieraja wlasciwa
        ilosc bitow
5   4350000 INFO3 [ TopTest.DacLTC2624Behav_ ]  Ustawionowartosc     0 (0x000)na
        adresie 15 (0xf)z komenda 12 (0xc)
6   4350000 INFO5 [ TopTest.DacLTC2624Behav_ ]  Dac o adresie 1111 (0xf) odpowiada
         wszystkim dac-om
```

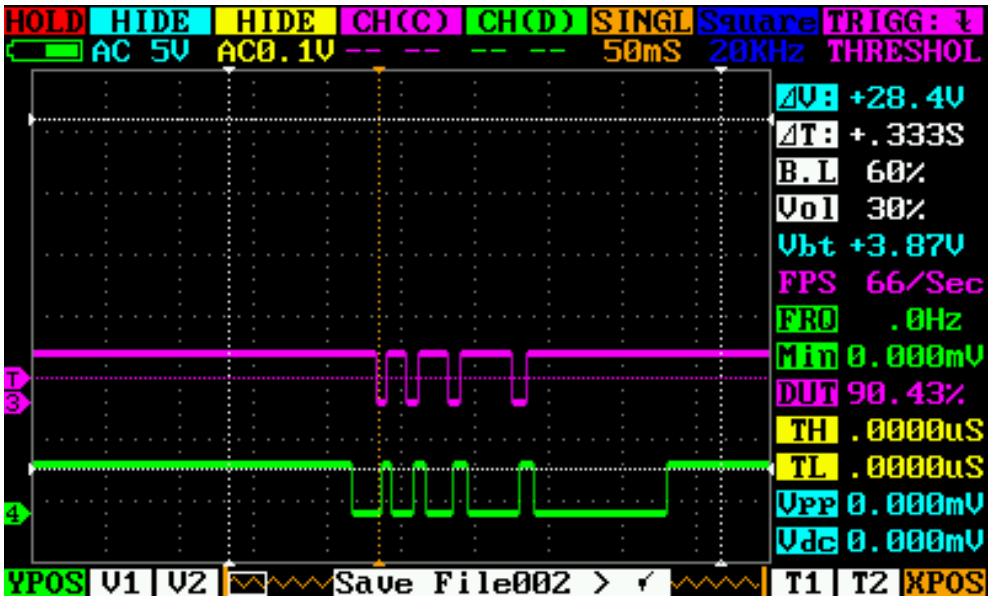
```
 7   4351000 INFO3 [ TopTest.TopTestBench_ ]  Zwiekszanie napiecia
 8   4351000 INFO3 [ TopTest.DacLTC2624Behav_.receive_packet ]  Odebrano pakiet
 9   4530000 INFO3 [ TopTest.DacLTC2624Behav_ ]  Odbieranie danych
10   4530000 INFO3 [ TopTest.DacLTC2624Behav_.receive_packet ]  Odbieranie pakietu
11   8370000 INFO4 [ TopTest.DacLTC2624Behav_ ]  Odebrane dane zawieraja wlasciwa
        ilosc bitow
12   8370000 INFO3 [ TopTest.DacLTC2624Behav_ ]  Ustawionowartosc  128 (0x080)na
        adresie 15 (0xf)z komenda 12 (0xc)
13   8370000 INFO5 [ TopTest.DacLTC2624Behav_ ]  Dac o adresie 1111 (0xf) odpowiada
         wszystkim dac-om
14   8371000 INFO3 [ TopTest.DacLTC2624Behav_.receive_packet ]  Odebrano pakiet
15   8401000 INFO3 [ TopTest.TopTestBench_ ]  Zmniejszanie napiecia
16   8610000 INFO3 [ TopTest.DacLTC2624Behav_ ]  Odbieranie danych
17   8610000 INFO3 [ TopTest.DacLTC2624Behav_.receive_packet ]  Odbieranie pakietu
18   12450000 INFO4 [ TopTest.DacLTC2624Behav_ ]  Odebrane dane zawieraja wlasciwa
        ilosc bitow
19   12450000 INFO3 [ TopTest.DacLTC2624Behav_ ]  Ustawionowartosc   64 (0x040)na
        adresie 15 (0xf)z komenda 12 (0xc)
20   12450000 INFO5 [ TopTest.DacLTC2624Behav_ ]  Dac o adresie 1111 (0xf)
        odpowiada wszystkim dac-om
21   12451000 INFO3 [ TopTest.DacLTC2624Behav_.receive_packet ]  Odebrano pakiet
```

5 Rotor

Płytka Spartan 3AN jest wyposażona w obrotowy przełącznik o dwóch wyprowadzeniach do FPGA. W stanie jałowym są one przyłączone do wysokiego potencjału. Mechaniczne obracanie pokrętła powoduje zwieranie styków na liniach i uziemienie napięcia. Kierunek obrotu wyznacza linia, która wcześniej zostanie zwarta do masy lub, co jest ekwiwalentne, wcześniej wróci do stanu jałowego.



Rysunek 5: Obrót w kierunku zgodnym z ruchem wskazówek zegara. Fioletowa linia pokazuje sygnał *rota*, natomiast zielona *rotb*.



Rysunek 6: Kilka obrotów następujących po sobie

5.1 Aplikacja

FPGA po skonfigurowaniu lub zresetowaniu zapala jedną diode, piątą lub szóstą w szeregu. Obrót rotora powoduje przesunięcie stanu wszystkich diód o jedną pozycję w lewo lub w prawo

w zależności od kierunku obrotu. Pozycje skrajne się cyklicznie uzupełniają. Rotor można także nacisnąć, co spowoduje zmianę stanu pierwszej diody na przeciwny. Nacisk ma własną linię sygnałową nie powiązaną z obrotowymi.

5.2 Synteza

5.2.1 Warstwa wierzchnia

FPGA korzysta z wyprowadzeń zegara, dolnego przycisku resetu, *ROT_CENTER* jest do obsługi wciśnięć. *ROT_A* oraz *ROT_B* są połączeniami do pokrętła. Są też wyprowadzenia dla diód i dodatkowe połączenia pokrętła na oscyloskop.

Listing 53: Rs232Tx.v

```
14 module Top (
15     input           CLK50MHZ,
16     input           RST,
17     // rotor control
18     input           ROT_CENTER,
19     input           ROT_A,
20     input           ROT_B,
21     // leds
22     output [7:0] LED,
23     // debug
24     output          DEBUG_A,
25     output          DEBUG_B
26     );
```

Przycisk *ROT_CENTER* jest pozbawiany drgań styków dzięki przeprowadzeniu przez *Debouncer*-a. Zainstancjonowane są tu także moduły *Rotor*-a oraz *Controller*-a. Wyprowadzenia debugera to przypisania ciągłe na odpowiadające połączenia obrotu.

5.2.2 Rotor

Rotor otrzymuje bezpośrednio sygnały o mechanicznych obrotach pokrętła i je interpretuje, w wyniku czego sygnalizuje wykryte obroty wraz z ich rozpoznanym kierunkiem.

Listing 54: Rotor.v

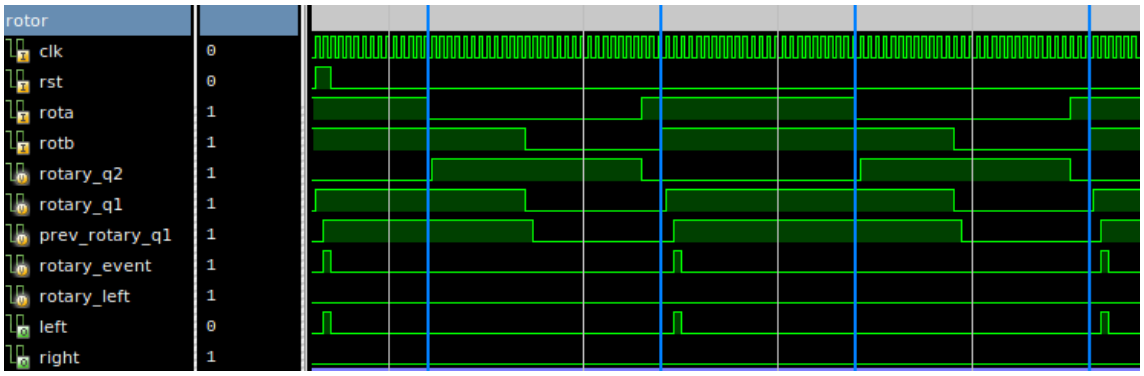
```
1  module Rotor(
2      input clk,
3      input rst,
4      // inputs
5      input rota,
6      input rotb,
7      // one pulse output direction signals
8      output left,
9      output right
10 );
```

Dla zobrazowanie dalszych zależności między sygnałami, zamieszczony jest zrzut z symulacji.

Rejestry pomocnicze zatrzaskują stan linii *rotb*. *rotary_q1* gdy wyprowadzenia pokrętła są zgodne, natomiast *rotary_q2* gdy są przeciwne.

Listing 55: Rotor.v

```
12     reg rotary_q1 = 1'b0;
13     always @(posedge clk)
```



Rysunek 7: Symulacja sygnałów rotora

```
14          if(rota ~^ rotb)
15              rotary_q1 <= rotb;
16
17      reg rotary_q2 = 1'b0;
18      always @(posedge clk)
19          if(rota ^ rotb)
20               rotary_q2 <= rotb;
```

Dodatkowy rejestr *prev_rotary_q1* przechowuje wartość *rotary_q1* z poprzedniej chwili. To umożliwia wykrycie zbocza narastającego *rotary_q1*, czyli momentu gdy obrót się skończył i oba wyprowadzenia pokrętła są ponownie wysokie w stanie jałowym.

Pogląd na kierunek obrotu daje drugi rejestr pomocniczy *rotary_q2*. Linie pokrętła nie kończą wysterowywania obrotu jednocześnie. Kolejność powrotów mówi o kierunku obrotu. *rotary_q2* zachowa stan *rotb* w chwili pierwszego powrotu. Wystarczy teraz sprawdzić zachowany stan *rotary_q2*, by odpowiedzieć czy ona wróciła pierwsza, co implikowałoby obrót w prawo - zgodny z ruchem wskazówek zegara.

Listing 56: Rotor.v

```
22      reg prev_rotary_q1 = 1'b0;
23      reg rotary_event = 1'b0;
24      reg rotary_left = 1'b0;
25      always @(posedge clk) begin
26          prev_rotary_q1 <= rotary_q1;
27          if(~prev_rotary_q1 && rotary_q1) begin
28              rotary_event <= 1'b1;
29              rotary_left <= rotary_q2;
30          end else
31              rotary_event <= 1'b0;
32      end
33
34      assign left = rotary_event & ~rotary_left;
35      assign right = rotary_event & rotary_left;
36
37  endmodule
```

5.2.3 Kontroler

Kontroler operuje na rejestrze, który powiązany jest z diodami led na płytce. Po świeżej konfiguracji płytki świeci się dioda piąta. Przycisk resetu ustawia jedynie szóstą. Przycisk centralny pokrętła zmieni stan diody pierwszej na przeciwny. Kontroler dostaje też impulsy *left* oraz

right informujący o zajściu obrotu i jego kierunku. Wtedy przesuwa bity rejestru w wyznaczonym kursie.

Listing 57: Controller.v

```
1  module Controller(
2            input clk,
3            input rst,
4            // tick inputs
5            input center,
6            input left,
7            input right,
8            //  leds
9            output reg [7:0] leds = 8'b0001_0000
10     );
11
12     always @(posedge clk)
13         if(rst)             leds <= 8'b0010_0000;
14         else if(center)     leds[0] <= ~leds[0];
15         else if(left)       leds <= { leds[6:0], leds[7] };
16         else if(right)      leds <= { leds[0], leds[7:1] };
17
18 endmodule
```

5.3 Symulacja

Moduły testowe jedynie symulują obracanie pokrętła przez użytkownika poprzez wysterowanie jego sygnałów. Moduł najwyższy symulacji jedynie zapewnia zainstancjonowanie zegara, resetu, modułu syntezowalnego i testowego odpowiednika pokrętła.

5.3.1 Przypadek testowy

Listing 58: TopTestBench.v

```
1  module TopTestBench #(
2      LOGLEVEL = 3,
3      LOGLEVEL_BEHAV = 3,
4      LOGLEVEL_BEHAV_CENTER = 3,
5      LOGLEVEL_BEHAV_ROTA = 3,
6      LOGLEVEL_BEHAV_ROTB = 3
7  ) (
8      // rotor control
9      output ROT_CENTER,
10     output ROT_A,
11     output ROT_B
12 );
```

On z kolei instancjuje *Rotor_behav* i operuje pokrętłem poprzez jego zestaw zadań.

Listing 59: TopTestBench.v

```
37         // kilka ruchow
38         rotor_behav.turn_left();
39         rotor_behav.turn_left();
40         rotor_behav.press_center();
41         rotor_behav.turn_right();
```

5.3.2 Rotor zachowawczo

Rotor dysponuje garstką prostych zadań.

Listing 60: Rotor_behav.v

```
1    task turn_left();
2       begin
3           ..
4           // rozpoczecie skretu w lewo
5           set_rota.low_during( 250 );
6           set_rotb.low_during( 300 );
7
8           // konczenie skretu w lewo
9           set_rota.high_during( 50 );
10          set_rotb.high_during( 500 );
11          ..
12      end
13   endtask
```

Obrót w prawo jest analogiczny do lewego.

Listing 61: Rotor_behav.v

```
1    task turn_right();
2       begin
3           ..
4           // rozpoczecie skretu w prawo
5           set_rotb.low_during( 250 );
6           set_rota.low_during( 300 );
7
8           // konczenie skretu w prawo
9           set_rotb.high_during( 50 );
10          set_rota.high_during( 500 );
11          ..
12      end
13   endtask
```

Po zwolnieniu wciśniętego przycisku, dołożona jest chwila przerwy.

Listing 62: Rotor_behav.v

```
1    task press_center();
2       begin
3           ..
4           set_center.high_during( 400 );
5           set_center.low_during( 200 );
6           ..
7       end
8    endtask
```

5.3.3 Komunikaty

Przedstawione jest pełne wyjście z pierwszego zadania obrotu w lewo przy maksymalnym stopniu logowania.

```
1 Time resolution is 1 ps
2 Simulator is doing circuit initialization process.
```

```
3       0 INFO3 [ TopTest.TopTestBench_.rotor_behav.set_center.state ] Stan
          sygnalow zostanie zmieniony. Obecny stan 'x' (0x x), spodziewany stan
          '0' (0x 0)
4       0 INFO4 [ TopTest.TopTestBench_.rotor_behav.set_center.state ] Stan
          sygnalow zostal zmieniony. Obecny stan '0' (0x 0)
5       0 INFO3 [ TopTest.TopTestBench_.rotor_behav.set_rota.state ] Stan sygnalow
            zostanie zmieniony. Obecny stan 'x' (0x x), spodziewany stan '1' (0x 1)
6       0 INFO4 [ TopTest.TopTestBench_.rotor_behav.set_rota.state ] Stan sygnalow
            zostal zmieniony. Obecny stan '1' (0x 1)
7       0 INFO3 [ TopTest.TopTestBench_.rotor_behav.set_rotb.state ] Stan sygnalow
            zostanie zmieniony. Obecny stan 'x' (0x x), spodziewany stan '1' (0x 1)
8 Finished circuit initialization process.
9 300000 INFO3 [ TopTest.TopTestBench_ ] Poczatek symulacji
10 300000 INFO3 [ TopTest.TopTestBench_.rotor_behav.turn_left ] Obracanie pokretla
     w lewo
11 300000 INFO3 [ TopTest.TopTestBench_.rotor_behav.set_rota.state ] Stan sygnalow
     zostanie zmieniony. Obecny stan '1' (0x 1), spodziewany stan '0' (0x 0)
12 300000 INFO4 [ TopTest.TopTestBench_.rotor_behav.set_rota.state ] Stan sygnalow
     zostal zmieniony. Obecny stan '0' (0x 0)
13 300000 INFO5 [ TopTest.TopTestBench_.rotor_behav.set_rota.state_during ] Sygnal
     w stanie 0 (hex 0) zostaje zamrozony przez zadany okres czasu        250
14 550000 INFO6 [ TopTest.TopTestBench_.rotor_behav.set_rota.state_during ]
    Przeczekiwanie stanu 0 (hex 0) zostalo zakonczone po        250
15 550000 INFO3 [ TopTest.TopTestBench_.rotor_behav.set_rotb.state ] Stan sygnalow
     zostanie zmieniony. Obecny stan '1' (0x 1), spodziewany stan '0' (0x 0)
16 850000 INFO3 [ TopTest.TopTestBench_.rotor_behav.set_rota.state ] Stan sygnalow
     zostanie zmieniony. Obecny stan '0' (0x 0), spodziewany stan '1' (0x 1)
17 850000 INFO4 [ TopTest.TopTestBench_.rotor_behav.set_rota.state ] Stan sygnalow
     zostal zmieniony. Obecny stan '1' (0x 1)
18 850000 INFO5 [ TopTest.TopTestBench_.rotor_behav.set_rota.state_during ] Sygnal
     w stanie 1 (hex 1) zostaje zamrozony przez zadany okres czasu         50
19 900000 INFO6 [ TopTest.TopTestBench_.rotor_behav.set_rota.state_during ]
    Przeczekiwanie stanu 1 (hex 1) zostalo zakonczone po         50
20 900000 INFO3 [ TopTest.TopTestBench_.rotor_behav.set_rotb.state ] Stan sygnalow
     zostanie zmieniony. Obecny stan '0' (0x 0), spodziewany stan '1' (0x 1)
21 400000 INFO4 [ TopTest.TopTestBench_.rotor_behav.turn_left ] Obrocono pokretlo
    w lewo
```

6 Rs232

Rs232 jest asynchronicznym, szeregowym standardem komunikacyjnym. Zdefiniowane są osobne linie do wysyłanych i odbieranych danych. Szybkość transmisji ustala się ręcznie w urządzeniach końcowych.

Stanem wysokim określony jest stan bezczynności. Nadawanie danych rozpoczyna się od opuszczenia linii *tx* na okres jednego bitu tzw. startowego. Następnie przesyłane są kolejne bity danych i kończone są wysokim bitem stopu. Standard przewiduje możliwy bit parzystości poprzedzający stop.

6.1 Aplikacja

Zaimplementowana aplikacja oczekuje wysłania bajtu danych przez urządzenie po drugiej stronie kabla, po czym mu go odsyła w niezmienionej postaci. Działa jak zwykłe echo. Pracuje z prędkością 115200 bitów na sekundę. Nie implementuje bitu parzystości.

6.2 Synteza

6.2.1 Warstwa wierzchnia

Moduł najwyższy w zasadzie przepuszcza wszystkie połączenia do właściwego modułu echa. Jedynie wyprowadza dodatkowo stan linii komunikacyjnych na złącze oscyloskopu do łatwej analizy. Natomiast moduł echa instancjuje część odbierającą i transmitującą dane oraz je łączy.

Listing 63: Rs232Echo.v

```
1  module Rs232Echo(
2          input           clk,
3          input           RST,
4          input           RxD,
5          output          TxD
6  );
7
8          wire RxD_data_ready;
9          wire [7:0] RxD_data;
10         Rs232Rx rx(.CLK50MHZ(clk), .RST(RST), .RxD(RxD), .RxD_data_ready(
               RxD_data_ready), .RxD_data(RxD_data));
11         Rs232Tx tx(.CLK50MHZ(clk), .RST(RST), .TxD(TxD), .TxD_start(
               RxD_data_ready), .TxD_data(RxD_data));
12
13 endmodule
```

6.2.2 Transmisja

Transmiter jest prostszy, zatem od niego warto zacząć. Należy przekazać mu w parametrach taktowanie zegara podstawowego płytki oraz zakładaną prędkość pracy. Aby z modułu skorzystać wystarczy podsunąć wysyłany bajt połączeniem TxD_data, po czym podnieść na jeden cykl TxD_start. Wtedy zacznie formować stan linii wyjściowej TxD poprzez serializację dostarczonego bajtu wraz z dodatkowymi skrajnymi bitami startu i stopu. Swoją zajętość sygnalizuje flagą TxD_busy.

Listing 64: Rs232Tx.v

```
1  module Rs232Tx
2  #(
```

```
3     parameter FREQ = 50000000,    // 50MHz
4     parameter BAUD = 115200       // 115 200 bounds/sec
5 ) (
6     input         CLK50MHZ,
7     input         RST,
8     input         TxD_start,
9     input [7:0]   TxD_data,
10    output        TxD,
11    output        TxD_busy
12 );
```

Zalecany standard 232 przewiduje tolerancję niedokładności zegara nie przekraczającą 5% w dowolnym kierunku. Przy przesyłanych łącznie 10 bitach w pakiecie oznacza to rozjechanie się zegarów odbiornika i nadajnika maksymalnie o pół bitu. Prosty licznik oparty na zegarze podstawowym $50Mhz$ nie spełnia wymagań. Należy zachowywać resztę z dodawań po każdym przepełnieniu. Dlatego do wyznaczenia momentu nadania kolejnego bitu służy dedykowany moduł *BaudRateGenerator* z wykalkulowaną inkrementacją.

Listing 65: Rs232Tx.v

```
14    `include "../../generic/log2.v"
15    localparam N = log2(BAUD);
16
17    wire          BaudTick;
18    wire          TxD_ready;
19    BaudRateGenerator #(
20          .INC( ((BAUD<<(N-4))+(FREQ>>5))/(FREQ>>4) ), // = 302
21          .N(N)
22      ) baud115200 (
23          .CLK50MHZ(CLK50MHZ),
24          .RST(RST),
25          .en(TxD_busy),
26          .tick(BaudTick)
27     );
```

Znany moduł do serializacji zajmuje się wysyłaniem danych w takt sygnału generowanego przez *BaudRateGenerator*. Dane te należy otoczyć bitem startu i stopu oraz w całości zanegować dla spójności z jałową jedynką.

Listing 66: Rs232Tx.v

```
29     // START BIT, data, STOP BIT
30     wire [9:0]   rs_data = { 1'b1, ~TxD_data, 1'b0 };
31    // output TxD line should be high at idle
32     wire         TxD_neg;
33     Serial #(
34          .WIDTH(10)
35     ) Serial_ (
36          .CLKB(CLK50MHZ),
37          .RST(RST),
38          // serial module interface
39          .rx(1'b0),
40          .tx(TxD_neg),
41          .data_in(rs_data),
42          .trig(TxD_start),
43          .ready(TxD_ready),
44          .tick(BaudTick)
45     );
46
47    assign TxD       = ~ TxD_neg;
```

```
48      assign TxD_busy = ~ TxD_ready;
49
50  endmodule
```

6.2.3 Odbiór

Moduł odbiorczy podobnie ma parametryzowane zegar podstawowy płytki oraz szybkość pracy. Gdy odbierze nowe dane, wystawia je na wyjścia *RxD_data* oraz informuje o zdarzeniu flagą *RxD_data_ready* przez jeden cykl.

Listing 67: Rs232Rx.v

```
1  module Rs232Rx
2  #(
3          parameter FREQ = 50000000,          // 50MHz
4          parameter BAUD = 115200             // 115 200 bounds/sec
5  ) (
6          input           CLK50MHZ,
7          input           RST,
8          input           RxD,
9          output [7:0]    RxD_data,
10         output          RxD_data_ready
11 );
```

Protokół jest asynchroniczny, więc sygnał zegarowy nie jest przekazywany, co utrudnia nieco odbiór. Linia *RxD* jest próbkowana z trzykrotnie większą częstotliwością, a przyjęty odebrany bit jest wynikiem głosowania próbek. Podejście wymusza powołanie również szybszego modułu *BaudRateGenerator*-a.

Listing 68: Rs232Rx.v

```
13      `include "../../generic/log2.v"
14      localparam N = log2(BAUD);
15
16      wire                receving;
17      wire                BaudTick3;
18      BaudRateGenerator #(
19          // 3 times oversampling
20          .INC( ((BAUD*3<<(N-7))+(FREQ>>8))/(FREQ>>7) ), // = 2416
21          .N(N)
22      ) baud_115200x3 (
23          .CLK50MHZ(CLK50MHZ),
24          .RST(RST),
25          .en(receving),
26          .tick(BaudTick3)
27      );
28
29      wire            BaudTick;
30      BaudRateGenerator #(
31          .INC( ((BAUD<<(N-4))+(FREQ>>5))/(FREQ>>4) ), // = 302
32          .N(N)
33      ) baud_115200 (
34          .CLK50MHZ(CLK50MHZ),
35          .RST(RST),
36          .en(receving),
37          .tick(BaudTick)
38      );
```

Tutaj instancjalizowany jest rejestr przesuwny dla próbek oraz kreowany układ głosujący przypisaniem ciągłym.

Listing 69: Rs232Rx.v

```
40      wire [2:0]    rx3;
41      Shiftreg #(
42          .WIDTH(3)
43      ) rx3_shitftreg (
44              .CLKB(CLK50MHZ),
45              .en(1'b1),
46              .set(1'b0),
47              .tick(BaudTick3),
48              .rx(RxD),
49              .data_in(3'b000),
50              .data_out(rx3)
51              );
52
53      // Vote for rx value
54      wire                        rx =
55                                  rx3 != 3'b000 &&
56                                  rx3 != 3'b001 &&
57                                  rx3 != 3'b010 &&
58                                  rx3 != 3'b100;
```

Serializer zajmuje się deserializacją napływających bitów pakietu. Bity początkowy startu i końcowy stopu zostają obcięte.

Listing 70: Rs232Rx.v

```
59      wire                        trig;
60      wire                        ready;
61      wire [9:0]                  data_out;
62      Serial #(
63            .WIDTH(10)
64       ) Serial_ (
65            .CLKB(CLK50MHZ),
66            .RST(RST),
67            // serial module interface
68            .rx(rx),
69            .data_in(10'b0),
70            .data_out(data_out),
71            .trig(trig),
72            .ready(ready),
73            .tick(BaudTick)
74       );
75
76      // get rid of START and STOP bits
77      assign RxD_data = data_out[8:1];
```

Konieczna jest maszyna stanów. Najpierw wyczekuje niskiego bitu startu, po czym odbiera pakiet i zgłasza jego przyjęcie. Cykl się powtarza.

Listing 71: Rs232Rx.v

```
79      localparam [1:0]
80        WAIT_STARTBIT = 3'd0,
81        START_RECEVING = 3'd1,
82        RECEVING = 3'd2,
83        RECEIVED = 3'd3;
84
```

```
85      reg [2:0]                   state = WAIT_STARTBIT;
86      always @(posedge CLK50MHZ)
87        if(RST)
88          state <= WAIT_STARTBIT;
89        else
90          case(state)
91            WAIT_STARTBIT:
92              if(~RxD)
93                state <= START_RECEVING;
94            START_RECEVING:
95                state <= RECEVING;
96            RECEVING:
97              if(ready)
98                state <= RECEIVED;
99            RECEIVED:
100               state <= WAIT_STARTBIT;
101         endcase
102
103     assign          trig = (state == START_RECEVING);
104     assign          RxD_data_ready = (state == RECEIVED);
105     assign          receving = (state == RECEVING);
106
107 endmodule
```

6.3 Symulacja

Moduł symulacyjny wysyła serię bajtów wzorcowych, po czym oczekuje odesłania ich z powrotem.

Moduł najwyższy *TopTest* kreuje instancje zegara, resetu, *Rs*232 i wierzchni moduł syntezowalny oraz zapewnia między nimi połączenia.

6.3.1 Rs232

*Rs*232 zawiera pamięć z napisem, który zamierza nadać, po czym odebrać.

Listing 72: sim/Rs232.v

```
28      localparam CHARS = 8;
29      reg [7:0] mem [CHARS:0];
30      initial begin
31          mem[0] = "A";
32          mem[1] = "G";
33          mem[2] = "H";
34          mem[3] = " ";
35          mem[4] = "W";
36          mem[5] = "F";
37          mem[6] = "i";
38          mem[7] = "I";
39          mem[8] = "S";
40      end
```

Nadawane są kolejno litery korzystając z zadania modułu zachowawczego.

Listing 73: sim/Rs232.v

```
42      integer j = 0;
43      initial begin
44          ..
```

```
45          // wyslanie kolejnych liter napisu powitalnego
46          for(j=0; j<CHARS; j=j+1) begin
47              rs232_behav.transmit( mem[j] );
48              ..
49          end
50      end
```

Śledzona jest linia rx, która to powinny wysłane dane otrzymać z powrotem w tej samej formie i porządku, co zostaje poddane weryfikacji.

Listing 74: sim/Rs232.v

```
64      integer k = 0;
65      reg [7:0] byte_received = 8'd0;
66      always @(negedge rx) begin
67          if(inited) begin
68
69              // odbior bajtu
70              rs232_behav.receive( byte_received );
71              ..
72
73              // weryfikacja
74              if( byte_received != mem[k] )
75                  if(LOGLEVEL >= 1)
76                      $display("%t\t ERROR [ %m ] \t Odebrany bajt %b 0x%h %d %s rozni
                            sie od wyslanego wzorca %b 0x%h %d %s", $time, byte_received,
                            byte_received, byte_received, byte_received, mem[k], mem[k],
                            mem[k], mem[k]);
77
78              // inkrementuj numer biezacego bajtu
79              k = k + 1;
80          end
81      end
```

6.3.2 Rs232 zachowawczo

Moduł udostępnia zadania do wysyłania oraz odbierania danych. Zajmuje się formowanie ich w pakiety z doklejonymi bitami startu i stopu lub wyłuskiwaniem jego zawartości. Informuje o przebiegu swojej pracy z wybraną sześciostopniową dokładnością poziomu logowania.

Listing 75: sim/Rs232 behav.v

```
1  module Rs232_behav
2  #(
3      // LOGLEVEL = 0
4      //       bez zadnych komunikatow
5      // LOGLEVEL = 1
6      //       pokazuje bledy
7      // LOGLEVEL = 2
8      //       pokazuje ostrzezenia
9      //
10     // LOGLEVEL = 3
11     //       informuje o wyslaniu/otrzymaniu pelnego bajtu
12     // LOGLEVEL = 4
13     //       informuje o wysylaniu/otrzymywaniu poszczegolnych bitow
14     // LOGLEVEL = 5
15     //       informuje o wyslaniu/otrzymywaniue start i stop bitow
16     // LOGLEVEL = 6
17     //       informuje o przeczekiwaniu tolerowanego przesuniecia zegara
```

```
18      parameter LOGLEVEL=3,
19      parameter LOGLEVEL_RX=3,
20      parameter LOGLEVEL_TX=3,
21
22      parameter BAUDRATE = 115_200
23  ) (
24      input rx,
25      output tx
26  );
```

Wyliczone są czasy trwania pojedynczego bitu na podstawie żądanej prędkości transmisji oraz tolerancji na błąd przesunięcia zegara.

Listing 76: sim/Rs232 behav.v

```
64      // czas jednego okresu przy zakladanej szybkosci
65      localparam PERIOD = 1_000_000_000 / BAUDRATE;
66      // tolerancja zegara
67      localparam BITTOL = 0.05 * PERIOD;
```

Zadanie wysyłające przyjmuje bajt danych. Wysyła go w pętli poprzedzając bitem startu i zakańczając stopem.

Listing 77: sim/Rs232 behav.v

```
64      // Zadanie wysyla przekazany bajt wraz z poprzedajacym bitem startu i
            nastepujacym stopu
65      task transmit
66      (
67          input [7:0] byte_tosend
68      );
69          integer      i;
70          begin
71              ..
72              set_tx.low_during( PERIOD );
73
74              // Przekazany bajt
75              for(i=0; i < 8; i=i+1) begin
76                  set_tx.state_during( PERIOD, byte_tosend[i] );
77                  ..
78              end
79
80              // Stop bit
81              ..
82              set_tx.high_during( PERIOD );
83              ..
84          end
85      endtask
```

Zadanie odbierające dane symetrycznie do nadającego spodziewa się bitu startu, bajtu danych i stop bitu.

Listing 78: sim/Rs232 behav.v

```
118     task receive
119     (
120         output reg [7:0] byte_received
121     );
122         integer i;
123         reg startbit;
124         reg stopbit;
```

```
125         begin
126             // Zaczekaj na start bit
127             ..
128             monitor_rx.wait_for_low();
129             ..
130             // Odbierz bit startu
131             receive_bit( startbit );
132             if(startbit != 1'b0)
133                 if(LOGLEVEL >= 1)
134                     $display("%t\t ERROR [ %m ] \t Bit startu powinien byc niski",
                            $time);
135             ..
136             // Odbierz bajt danych
137             for(i=0; i < 8; i=i+1) begin
138                 receive_bit( byte_received[i] );
139                 ..
140             end
141
142             // Odbierz oczekiwany stop bit
143             ..
144             receive_bit( stopbit );
145             if(stopbit != 1'b1)
146                 if(LOGLEVEL >= 1)
147                     $display("%t\t ERROR [ %m ] \t Oczekiwany bit stopu powinien byc
                            wysoki", $time);
148             ..
149         end
150     endtask
```

Odbiór bitu realizowany jest w kolejnym dedykowanym zadaniu. Sprawdzane jest, czy linia *rx* jest stabilna przez długość trwania okresu z wyłączeniem początkowego i końcowego możliwego błędu synchronizacji zegarów.

Listing 79: sim/Rs232 behav.v

```
89      // Zadanie odbiera bit probkujac go 3 krotnie
90      task receive_bit
91      (
92          output reg received
93      );
94          begin
95              ..
96              #(BITTOL);
97              ..
98              monitor_rx.ensure_same_during( (PERIOD-2*BITTOL) );
99              ..
100             received = rx;
101             ..
102             #(BITTOL);
103             ..
104         end
105     endtask
```

6.3.3 Wyjście

Listing 80: Rs log

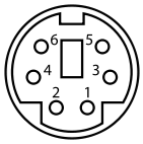
```
255 96800000 INFO3 [ TopTest.rs232 ]  Wyslano bajt '01001000' (0x 48) (dec  72) (
    ascii H)
```

```
256 183600000 INFO3 [ TopTest.rs232 ]  Wyslano bajt '01000101' (0x 45) (dec  69) (
    ascii E)
257 183710000 INFO3 [ TopTest.rs232 ]  Odebrano bajt '01001000' (0x 48) (dec  72) (
    ascii H)
258 270400000 INFO3 [ TopTest.rs232 ]  Wyslano bajt '01001100' (0x 4c) (dec  76) (
    ascii L)
259 270570000 INFO3 [ TopTest.rs232 ]  Odebrano bajt '01000101' (0x 45) (dec  69) (
    ascii E)
```

7 Klawiatura PS2

7.1 Interfejs PS2

Interfejs PS2 występuje najczęściej jako 6 stykowe gniazdo mini-DIN, takie też dostępne jest na płytce Spartan 3AN Starter Kit. Złącze mini-DIN ustandaryzowane zostało przez niemiecki instytut Deutsches Institut fuer Norm http://www.din.de, natomiast sposób komunikacji opracowała firma IBM w 1987 roku.



Rysunek 8: Port mini-DIN

Styk trzeci jest masą, styk czwarty zasila urządzenia peryferyjne napięciem 5 V podając natężenie nie większe niż 275mA. Linią pierwszą wymieniane są dane w takt zegara na linii piątej. Tyle wystarcza do komunikacji z urządzeniem peryferyjnym myszki lub klawiatury. Połączenia numer dwa i sześć zazwyczaj nie są wykorzystywane. Jednak w rzadkich przypadkach producent może je wykorzystać dla drugiego urządzenia. Wtedy z pomocą kabla rozdzielającego można podłączyć mysz i klawiaturę do jednego, wspólnego gniazda mini-DIN. To rozwiązanie stosowane jest w starych laptopach oraz na płytce Spartan 3AN dla oszczędności miejsca.

Linie zegara i danych wysterowane są wysoko w stanie jałowym. Obie są dwukierunkowe, co jest zrealizowane przez konfigurację otwartego kolektora podciągniętego do stanu wysokiego. Zegar taktowany jest zawsze przez urządzenie peryferyjne z częstotliwością w zakresie $10 - 16.7kHz$. Urządzenie przesyłając porcje danych do hosta najpierw przesyła niski bit startu, po czym następuje bajt danych w kolejności zaczynającej od bitu najmniej znaczącego, dalej jest bit parzystości i wysoki bit stopu. Stan bitów może być zmieniany w połowie czasu wysokiego zegara, bity odczytywane są przez hosta na zboczu opadającym.

Host może wysłać komendę do urządzenia. Przetrzymuje on wtedy linie zegarową oraz danych w stanach niskich przez czas co najmniej 100ms. Po tym zwalnia linie zegarową. Jest do żądanie do urządzenia o umożliwienie przesłania danych i wytaktowanie linii zegarowej. Po podaniu zegara, host ustawia bity w chwilach niskiego zegara. Ramka wygląda podobnie zawierając bit startu, bajt danych, parzystość i stop. Jednak po tym następuje dodatkowy bit potwierdzenia. Jest to obniżenie linii danych przez urządzenie i wygenerowania dla niego dodatkowego pulsu zegara. Potwierdza to właściwy odbiór danej.

7.2 Klawiatura

Klawiatura PS2 nadaje momenty wciśnięcia lub zwolnienia poszczególnych klawiszy. Jeśli klawisz jest przetrzymany dłużej, wtedy nadawane są przypomnienia jego wciśnięcia w odstępach 100ms.

Kody wciskanych klawiszy, nazywane skan kodami, powiązane są z ich położeniem i wcale się nie zakłada układu QWERTY rozmieszczenia klawiszy. Skan kody wymagają przekonwertowania do odpowiadającego użytkownikowi układu znaków.

Skan kod zajmuje jeden bajt. Krótkie wciśnięcie klawisza wysyła jego skan kod w pojedynczej ramce komunikacyjnej. Zwolnienie klawisza jest zdefiniowane przez wysłanie ramki poprzedzającej z bajtem F0 i kolejnej ramki ze skan kodem klawisza zwalnianego.

Rozróżnia się klawisze lewego oraz prawego alt-a i cntr-la. Lewe klawisze przesyłane są jak wszystkie pozostałe. Natomiast klawisze prawe poprzedzone są ramką o treści E0.

Do klawiatury można wysyłać polecenia od hosta. Polecenie echo określone bajtem EE powoduje odesłanie przez klawiaturę tego samego bajtu EE. Można ustawić częstotliwość powtarzania wciśniętego klawisze komendą $F3$. Po niej klawiatura odpowiada potwierdzeniem FA i następny bajt przesyła host z żądaną częstotliwością. Komenda ED ustawia stan diod led $CapsLock$, $ScrollLock$, $NumLock$. Klawiatura ją także potwierdza odsyłając FA i oczekuje kolejnego bajtu z maską diod.

7.3 Aplikacja

Aplikacja odbiera przesłany przez klawiaturę bajt i wyświetla go na diodach led. Można zauważyć skan kod wciskanego klawisza oraz mignięcia bajtu F0 przy zwalnianiu. Aplikacja nie przesyła do klawiatury żadnych komend.

7.4 Synteza

7.4.1 Warstwa wierzchnia

Moduł najwyższy jedynie instancjuje kontrolera i moduł klawiatury.

Listing 81: Top.v

```
1  module Top (
2      input  CLK50MHZ,
3      input  RST,
4      // keyboard
5      input  PS2_CLK1,
6      input  PS2_DATA1,
7      output [7:0] LED,
8  );
9
10     wire [7:0] scancode;
11     wire scan_ready;
12     Keyboard keyboard_ (
13         .CLK50MHZ(CLK50MHZ),
14         .RST(RST),
15         .ps2_clk(PS2_CLK1),
16         .ps2_data(PS2_DATA1),
17         .scancode(scancode),
18         .scan_ready(scan_ready)
19     );
20
21     Controller controller_ (
22         .CLK50MHZ(CLK50MHZ),
23         .RST(RST),
24         .scancode(scancode),
25         .scan_ready(scan_ready),
26         .led(LED)
27     );
28
29 endmodule
```

7.4.2 Kontroler

Kontroler jedynie wyświetla na diodach przesłany skan kod.

Listing 82: Controller.v

```
1  module Controller(
2    input       CLK50MHZ,
3    input       RST,
4    input [7:0]       scancode,
5    input       scan_ready,
6    output reg [7:0] led = 8'b0011_1100
7  );
8
9      always @(posedge CLK50MHZ)
10         if(RST)
11           led <= 8'b0111_1110;
12         else if( scan_ready )
13           led <= scancode;
14
15 endmodule
```

7.4.3 Klawiatura

Najważniejszym modułem projektu jest sama klawiatura. Przyjmuje linie zegarową *ps2_clk* i danych *ps2_data*, po odebraniu skan kodu wystawia go przewodem równoległym *scancode* i informuje o tym ustawiając flagę *scan_ready*.

Listing 83: Keyboard.v

```
1  module Keyboard(
2      input  CLK50MHZ,
3      input  RST,
4      input  ps2_clk,
5      input  ps2_data,
6      output [7:0] scancode,
7      output scan_ready
8      );
```

Wykrywanie zbocza opadającego zegara zrealizowane jest dwubitowym rejestrem przesuwnym i ciągłym porównywaniem go z wzorcem zbocza w pomocniczym module *Bits_Reverse*.

Listing 84: Keyboard.v

```
10     // Detect negative edge on input ps2 clock line
11     wire     ps2_clk_negedge;
12     Edge_Detector ps2_clk_negedge_detector (
13         .clk(CLK50MHZ),
14         .signal(ps2_clk),
15         .neg(ps2_clk_negedge)
16     );
```

Przesyłana ramka zczytywana jest szeregowo na wykrytym zboczu opadającym przy wykorzystaniu modułu serializacji. Moduł odbiera jedynie 10 pierwszych bitów z ramki, ostatni stop bit nie jest zapisywany. Ponieważ pierwszy takt zegara ps2 wprawia maszynę stanów w ruch i dopiero włącza serializację do pracy, takt ten nie zostanie policzony wewnątrz serializacji. Obejściem byłoby dopisanie rejestru opóźniającego takty zegara ps2 o cykl i przekazanie go do serializacji lub, co jest prostsze, obcięcie ramki.

Listing 85: Keyboard.v

```
18     wire         trig;
19     wire         ready;
20     wire [9:0] frame;
21     Serial #(
```

```
22          // ignore 11th stop bit as first tick is not counted
23          .WIDTH(10)
24      ) Serial_ (
25          .CLKB(CLK50MHZ),
26          .RST(RST),
27          // serial module interface
28          .rx(ps2_data),
29          .data_in(10'b0),
30          .data_out(frame),
31          .trig(trig),
32          .ready(ready),
33          .tick(ps2_clk_negedge)
34      );
```

Ramka pozbawiana jest otaczających bitów startu, parzystości i stopu, co ekstrahuje przesyłany bajt danych. Ogólny moduł do obsługi rejestru przesuwnego zaprojektowany został do odbioru bitów w kolejności od najbardziej znaczącego. Protokół PS2 przesyła je w odwrotnej kolejności, przez co wymagane jest użycie dodatkowego modułu przywracającego pierwotny porządek. Odebrane dane nie są weryfikowane pod kątem zgodności z bitem parzystości.

Listing 86: Keyboard.v

```
36      // Get rid of start and odd bits, then reverse bit order of the data
37      Bits_Reverse reversing (
38          .orginal( frame[9:2] ),
39          .reversed( scancode)
40      );
```

Klawiatura zawiera maszynę stanów. Najpierw czeka na rozpoczęcie transmisji opadającym zboczem zegara. Kiedy to nastąpi, uruchamia odbiór szeregowy i czeka na jego zakończenie. Zakończenie odbioru wprawia ją w ostatni stan sygnalizujący modułowi wyższemu gotowość odebranego bajtu.

Listing 87: Keyboard.v

```
42      localparam [2:0]
43       WAIT_STARTBIT = 3'd0,
44       START_RECEVING = 3'd1,
45       RECEIVING = 3'd2,
46       RECEIVED = 3'd3;
47
48      reg [1:0]   state = WAIT_STARTBIT;
49      always @(posedge CLK50MHZ)
50        if(RST)
51          state <= WAIT_STARTBIT;
52        else
53          case(state)
54            WAIT_STARTBIT:
55              if(ps2_clk_negedge)
56                state <= START_RECEVING;
57            START_RECEVING:
58                state <= RECEIVING;
59            RECEIVING:
60              if(ready)
61                state <= RECEIVED;
62            RECEIVED:
63                state <= WAIT_STARTBIT;
64          endcase
65
66       assign         trig = (state == START_RECEVING);
```

```
67      assign          scan_ready = (state == RECEIVED);
68
69  endmodule
```

7.5 Symulacja

Moduł najwyższy *TopTest* powołuje zegar, reset, moduł najwyższy syntezowalny oraz przypadek testowy do przeprowadzenia.

7.5.1 Przypadek testowy

Przypadek testowy powołuje instancję zachowawczą klawiatury, po czym operuje na jej zestawie zadań wciskając oraz zwalniając parę z nich.

Listing 88: TopTestBench.v

```
1  module TopTestBench (
2      input RST,
3      // keyboard
4      output PS2_CLK1,
5      output PS2_DATA1
6  );
7
8      Keyboard_behav #(
9          .LOGLEVEL(5)
10     ) keyboard_behav (
11         .clk(PS2_CLK1),
12         .data(PS2_DATA1)
13     );
14
15     initial begin
16         @(negedge RST);
17         #10_000;
18
19         keyboard_behav.press_right_alt();
20         keyboard_behav.type_char("a");
21         keyboard_behav.release_right_alt();
22
23         keyboard_behav.type_(keyboard_behav.ENTER);
24     end
25
26 endmodule
```

7.5.2 Klawiatura zachowawczo

Moduł zachowawczy przyjmuje parametr żądanej szczegółowości logowania oraz wysterowywuje linie zegara i danych.

Listing 89: Keyboard_behav.v

```
1  module Keyboard_behav
2  #(
3      // LOGLEVEL = 0
4      //       bez zadnych komunikatow
5      // LOGLEVEL = 1
6      //       pokazuje bledy
7      // LOGLEVEL = 2
```

```
8     //        pokazuje ostrzezenia
9     //
10    // LOGLEVEL = 3
11    //        informuje o wyslaniu/otrzymaniu pelnego bajtu
12    // LOGLEVEL = 4
13    //        informuje o wysylaniu/otrzymywaniu poszczegolnych bitow
14    // LOGLEVEL = 5
15    //        informuje o wyslaniu/otrzymywaniue start i stop bitow
16    // LOGLEVEL = 6
17    //        informuje o przeczekiwaniu tolerowanego przesuniecia zegara
18    //
19    parameter LOGLEVEL=3,
20    parameter LOGLEVEL_CLK=3,
21    parameter LOGLEVEL_DATA=3
22 ) (
23    output   clk,
24    output   data
25 );
```

Ustawiane zostają stałe połowy okresu $HALF\_PERIOD = 7000000$ oraz ćwiartka. Instancjalizowane zostają moduły ustawiania linii zegara i danych oraz inicjalizowane zostają stanami wysokimi.

Konstrukcja zadania wysyłającego daną *send_* jest bliźniaczo podobna do zadania *transmit* z modułu *Rs232_behav*. Wysyłane bity są jedynie zmieniane w połowie wysokiego zegara, a ramka poszerzona jest o bit parzystości wyrażany operatorem redukcji

Listing 90: Keyboard_behav.v

```
75    odd = ~^ scancode;
```

Zapisane są mapowania klawiszy specjalnych do odpowiadających skan kodów.

Listing 91: Keyboard_behav.v

```
110   // Pare klawiszy specjalnych
111
112   reg [7:0] ESC          = 8'h76;
113   reg [7:0] TAB          = 8'h0d;
114   reg [7:0] BKSPC        = 8'h0d;
115   reg [7:0] ENTER        = 8'h5a;
116   reg [7:0] SPACE        = 8'h29;
117   reg [7:0] LEFT_SHIFT   = 8'h12;
118   reg [7:0] RIGHT_SHIFT  = 8'h59;
119   reg [7:0] LEFT_CTRL    = 8'h14;
120   reg [7:0] LEFT_ALT     = 8'h11;
121
122   reg [7:0] EXT          = 8'he0;
123   reg [7:0] RELEASE      = 8'hf0;
```

Zdefiniowanych jest parę podstawowych operacji na klawiszach.

Listing 92: Keyboard_behav.v

```
125   // Zadania ogolne do wcisniecia, puszczenia oraz przetrzymania zadanego
          klawisza
126
127   task press_
128   (
129      input [7:0] scancode
130   );
131      begin
```

```
132             send_(scancode);
133         end
134     endtask
135 
136     task release_
137     (
138         input [7:0] scancode
139     );
140         begin
141             press_(EXT);
142     press_(scancode);
143         end
144     endtask
145 
146     task type_
147     (
148         input [7:0] scancode
149     );
150         begin
151     press_(scancode);
152     #1000;
153     release_(scancode);
154     #1000;
155         end
156     endtask
```

Dla rozróżnienia lewego lub prawego ctrl-a, dołączone są dedykowane zadania. To samo podejście zastosowane jest dla alt-a.

Listing 93: Keyboard_behav.v

```
158     // Klawisze lewego i prawego alt-a oraz ctrl-a sa rozroznialne
159 
160     task press_left_control
161     ();
162         begin
163     press_(LEFT_CTRL);
164         end
165     endtask
166 
167     task release_left_control
168     ();
169         begin
170     release_(LEFT_CTRL);
171         end
172     endtask
173 
174     task press_right_control
175     ();
176         begin
177     press_(EXT);
178     press_(LEFT_CTRL);
179         end
180     endtask
181 
182     task release_right_control
183     ();
184         begin
185     press_(EXT);
186     release_(LEFT_CTRL);
```

```
187         end
188     endtask
```

Translacją znaków tablicy ASCII do odpowiadających skan kodów zajmuje się tablica $char_to_scancode$.

Listing 94: Keyboard_behav.v

```
220     // Mapowanie znaku do skan kodu
221     reg [7:0] char_to_scancode [255:0];
222     initial begin
223         char_to_scancode["a"] = 8'h1c;
224         char_to_scancode["b"] = 8'h32;
225         ...
```

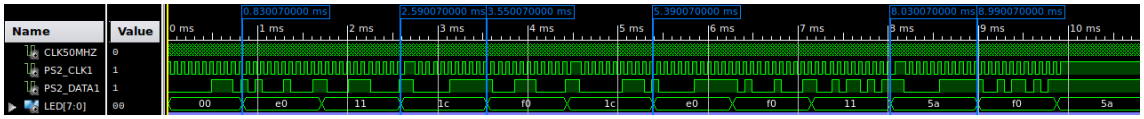
Ostatnie zadania operują na przekazanych znakach tablicy ASCII i przesyłają ich skan kod.

Listing 95: Keyboard_behav.v

```
262     // Zadania operuja na znakach
263
264     task press_char
265     (
266         input [7:0] char
267     );
268         begin
269             press_(char_to_scancode[char]);
270         end
271     endtask
272
273     task release_char
274     (
275         input [7:0] char
276     );
277         begin
278             release_(char_to_scancode[char]);
279         end
280     endtask
281
282     task type_char
283     (
284         input [7:0] char
285     );
286         begin
287             type_(char_to_scancode[char]);
288         end
289     endtask
```

7.5.3 Przebieg

Wycinek zrzutu ekranu pokazuje przebieg symulacji. Niebieskie markery oddzielają sekwencję dla poszczególnych klawiszy.



Rysunek 9: Przebieg symulacji

Zamieszczone jest wyjście logów dla domyślnych poziomów logowania.

Listing 96: Keyboard logs output

```
10000000 INFO3 [ TopTest.TopTestBench_.keyboard_behav.press_right_alt ] Wcisnieto prawy alt
10000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 11100000 (0x e0)
10000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '11100000' (0x e0)
890000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt 11100000 (0x e0)
890000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 00010001 (0x 11)
890000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '00010001' (0x 11)
1770000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt 00010001 (0x 11)
1770000000 INFO3 [ TopTest.TopTestBench_.keyboard_behav.type_char ] Wpisywanie klawisza o literze/cyfrze 01100001 (0x 61) (dec  97) (ascii a)
1770000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.type_ ] Wpisywanie przycisku o skan kodzie 00011100 (0x 1c)
1770000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 00011100 (0x 1c)
1770000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '00011100' (0x 1c)
2650000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt 00011100 (0x 1c)
2730000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.release_ ] Zwalnianie przycisku o skan kodzie 00011100 (0x 1c)
2730000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 11110000 (0x f0)
2730000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '11110000' (0x f0)
3610000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt 11110000 (0x f0)
3610000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 00011100 (0x 1c)
3610000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '00011100' (0x 1c)
4490000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt 00011100 (0x 1c)
4570000000 INFO3 [ TopTest.TopTestBench_.keyboard_behav.release_right_alt ] Zwalnianie prawego alt-a
4570000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 11100000 (0x e0)
4570000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '11100000' (0x e0)
5450000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt 11100000 (0x e0)
5450000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.release_ ] Zwalnianie przycisku o skan kodzie 00010001 (0x 11)
5450000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 11110000 (0x f0)
5450000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '11110000' (0x f0)
6330000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt 11110000 (0x f0)
6330000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto przycisk o skan kodzie 00010001 (0x 11)
6330000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie bajtu '00010001' (0x 11)
```

```
30  7210000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt
        00010001 (0x 11)
31  7210000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.type_ ] Wpisywanie
        przycisku o skan kodzie 01011010 (0x 5a)
32  7210000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto
        przycisk o skan kodzie 01011010 (0x 5a)
33  7210000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie
        bajtu '01011010' (0x 5a)
34  8090000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt
        01011010 (0x 5a)
35  8170000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.release_ ] Zwalnianie
         przycisku o skan kodzie 01011010 (0x 5a)
36  8170000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto
        przycisk o skan kodzie 11110000 (0x f0)
37  8170000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie
        bajtu '11110000' (0x f0)
38  9050000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt
        11110000 (0x f0)
39  9050000000 INFO4 [ TopTest.TopTestBench_.keyboard_behav.press_ ] Wcisnieto
        przycisk o skan kodzie 01011010 (0x 5a)
40  9050000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wysylanie
        bajtu '01011010' (0x 5a)
41  9930000000 INFO5 [ TopTest.TopTestBench_.keyboard_behav.send_ ] Wyslano bajt
        01011010 (0x 5a)
```

8 VGA

Standard VGA odpowiada za przesłanie obrazu do wyświetlacza. Wprowadziła go firma IBM w 1987 roku w jej linii komputerów osobistych. Standard definiuje maksymalne wymiary obrazu w trybie znakowym jako 720x480, natomiast w trybie graficznym jest to 640x480 przy 16 lub 256 dostępnych kolorach i częstotliwości odświeżania do 70 Hz. Punkty obrazu, zwane pikselami, przesyłane są w porządku od lewej do prawej kolumny, zaczynając od górnego do dolnego wiersza.

Po przesłaniu każdego wiersza, następuje chwilowe obniżenie linii synchronizacyjnej $HSYNC$. Natomiast po przesłaniu całej ramki, chwilowo opuszczana jest linia $VSYNC$. W punktach tych i ich pewnym otoczeniu należy uziemić linie kolorów.

Kolor pikseli określa złożenie nasycenia trzech podstawowych barw: czerwonej, zielonej i niebieskiej. Ich stan podawany jest osobnymi liniami analogowo wartościami napięcia z zakresu $0.0V - 0.7V$.

Płytka Spartan wykorzystuje po 4 cyfrowe wyjścia FPGA dla każdej z barw łączone w drabinki rezystorowe tworząc proste ADC.

8.1 Aplikacja

Zaprojektowana aplikacja pokazuje na ekranie przyłączonego wyświetlacza jeden z ośmiu kolorów. Użytkownik wybiera ulubiony kolor posługując się dwoma przyciskami. Jeden z nich przełącza kolor tła na następną dostępną barwę, drugi wraca do poprzedniej. Końce listy dostępnych barw są połączone, próba wykroczenia poza nią powoduje płynne przejście na jej przeciwny koniec.

8.2 Synteza

8.2.1 Połączenia

Moduł najwyższy przyjmuje sygnały zegarowy, resetu oraz dwóch przycisków służących zmianie koloru wyświetlanego tła. Moduł wyprowadza sygnały synchronizacji ramek i kolumn oraz pożądany kolor podany poprzez trzy czterobitowe rejestry nasycenia czerwieni, błękitu i zieleni.

Listing 97: Top.v

```
1  module Top (
2      input           CLK50MHZ,
3      input           RST,
4      // vga interface
5      output [3:0] VGA_R,
6      output [3:0] VGA_G,
7      output [3:0] VGA_B,
8      output          VGA_HSYNC,
9      output          VGA_VSYNC,
10     // color control
11     input           BTN_NEXT,
12     input           BTN_PREV,
13 );
```

Sygnały przycisków przewijania należy pozbawić drgań styków wykorzystując poznany moduł *Debouncer*. Następnie zostają zainstancjonowane moduły *Sync* służący synchronizacji oraz *Controller*, który obsługuje przyciski i podaje wybrany kolor.

8.2.2 Synchronizacja

Standard VGA powstał w czasach monitorów kineskopowych. Technologia ta ukierunkowywuje elektromagnesami naładowaną wiązkę o wybranej energii w punkt na luminoforze, czym wyzwala fotony światła. Momenty gdy wiązka powraca na początek kolejnej linii lub całej ramki są punktami synchronizacji. Podczas powrotów wiązki, a także w chwilach przed i po, musi być ona wygaszona (przez obniżenie wartości składowych kolorów), aby nie zakłócić kolorów nadanych luminoforowi w pierwszym przebiegu.

Zadaniem modułu synchronizacyjnego jest właściwe wytaktowanie linii zawiadamiających o początku nowej ramki lub nowej kolumny. Są to sygnały cyfrowe utrzymywane w stanie wysokim, w chwilach synchronizacji zostają obniżane. Domyślne parametry dostosowane są do przesyłania obrazu rozmiaru 640x480 pikseli przy częstotliwości odświeżania 60 Hz. Odmierzanie czasu przesyłu kolumn bazuje na podstawowym zegarze występującym na płytce o częstotliwości 50 mHz. Natomiast przesłanie ramki sygnalizowane jest po zliczeniu wysłania odpowiedniej liczby kolumn.

Sygnał *dispalying* określa czy kolory mogą być podawane, sygnały współrzędnych x i y podają bieżącą lokalizuję wiązki.

Listing 98: Sync.v

```
1  module Sync
2  #(
3      parameter H_S  = 2*800,
4      parameter H_FP = 2*16,
5      parameter H_PW = 2*96,
6      parameter H_BP = 2*48,
7      parameter V_S  = 521,
8      parameter V_PW = 2,
9      parameter V_FP = 10,
10     parameter V_BP = 29
11 ) (
12     input           CLK50MHZ,
13     input           RST,
14     // vga interface
15     output          VGA_HSYNC,
16     output          VGA_VSYNC,
17     // tick for next pixel
18     output [10:0] x,
19     output [10:0] y,
20     output          displaying
21 );
```

Synchronizacja opiera się na dwóch licznikach. Pierwszy zlicza zegar podstawowy 50 mHz, drugi zlicza przepełnienia pierwszego śledząc bieżącą linie w ramce. Połączenia i oraz j są aktualnymi stanami liczników.

Listing 99: Sync.v

```
23          wire [10:0] i;
24          wire        h;
25          Counter #(
26                  .MAX(H_S)
27          ) Counter_h (
28                  .CLKB(CLK50MHZ),
29                  // counter
30                  .en(1'b1),
31                  .rst(RST),
32                  .sig(1'b1), // count all CLK50MHZ ticks
```

```
33                  .cnt(i),
34                  .full(h)
35          );
36
37          wire [9:0] j;
38          Counter #(
39                  .MAX(V_S)
40          ) Counter_v (
41                  .CLKB(CLK50MHZ),
42                  // counter
43                  .en(1'b1),
44                  .rst(RST),
45                  .sig(h), // count h sync
46                  .cnt(j)
47          );
```

Dzięki przypisaniu ciągłemu do warunku logicznego, liczniki w stanach od 0 do długości trwania pulsu wywołują efekt synchronizacji. Natomiast połączenie *dispalying* obejmuje również chwile zakazu wyświetlania wokół tych punktów.

Listing 100: Sync.v

```
49          assign displaying = (
50              i >= H_PW + H_BP &&
51              i <  H_S  - H_FP &&
52              j >= V_PW + V_BP &&
53              j <= V_S  - V_FP
54          );
55
56          assign VGA_HSYNC = (i >  H_PW);
57          assign VGA_VSYNC = (j >= V_PW);
58
59          assign x = i - H_PW - H_BP;
60          assign y = j - V_PW - V_BP;
61
62  endmodule
```

8.2.3 Controller

Moduł kontrolera odpowiada za kolory poszczególnych pikseli. Dostaje on informacje czy może podawać kolor oraz obecne położenie wiązki. Jednakże jest on uproszczony i wypełnia cały obszar jednym jedynie kolorem, a więc położenie wiązki jest mu zupełnie zbędne. Dostępnych jest jedynie 8 kolorów. Kolory przełączane są dwoma przyciskami na następny lub poprzedni.

Listing 101: Sync.v

```
49  module Controller (
50      input       CLK50MHZ,
51      input       RST,
52      // vga interface
53      output [3:0] VGA_R,
54      output [3:0] VGA_G,
55      output [3:0] VGA_B,
56      // color control
57      input        next,
58      input        prev,
59      input [10:0] x,
60      input [10:0] y,
61      input displaying
```

```
62 );
```

Kolor bieżący zapisany jest w trzybitowym rejestrze. Zmieniany jest poprzez zewnętrzne przyciski. Przepełnianie licznika zachowuje się jak wybieranie od początku.

Listing 102: Sync.v

```
16     reg [2:0]       i = 1;
17     always @(posedge CLK50MHZ)
18         if(RST)
19           i <= 1;
20         else if(next)
21           i <= i + 1;
22         else if(prev)
23           i <= i - 1;
```

Każdy z bitów bieżącego koloru odpowiada jednej ze składowych podstawowych. Wartości pośrednie nie są wykorzystywane.

Listing 103: Sync.v

```
25     assign VGA_R = (displaying && i[0]) ? 4'hf : 4'h0;
26     assign VGA_G = (displaying && i[1]) ? 4'hf : 4'h0;
27     assign VGA_B = (displaying && i[2]) ? 4'hf : 4'h0;
```

8.3 Symulacja

Symulacja musi zweryfikować, czy sygnały synchronizacyjne pojawiają się w spodziewanych okresach oraz czy wtedy oraz w ich otoczeniu wiązka jest wygaszona. Do tego zliczane są linie w ramce.

8.3.1 Przypadek testowy

Zaimplementowana jest symulacja wciśnięć przycisków zmian kolorów. Wciskany jest dwukrotnie przycisk żądania następnego koloru, po czym raz wciśnięty zostaje przycisk koloru poprzedniego.

8.3.2 Moduł behawioralny

Za odbiór i sprawdzenie nadawanych sygnałów wtyczki VGA odpowiada moduł *Vga_Behav*. Przyjmuje trzy czterobitowe sygnały kolorów oraz dwie synchronizujące linii i ramek. Moduł jedynie odbiera sygnały, nie generuje żadnych zwrotnych.

Vga_Behav przyjmuje szereg parametrów. Domyślne wartości odpowiadają odbiorowi obrazu o rozmiarach 640x480 pikseli przy częstotliwości odświeżania 60 Hz. Czasy niewiele różnią się od tych podanych w dokumentacji do płytki, dostosowane są dla zegara 50 mHz.

Listing 104: Vga_behav.v

```
1 module Vga_Behav
2 #(
3     // LOGLEVEL = 0
4     //       bez zadnych komunikatow
5     // LOGLEVEL = 1
6     //       pokazuje bledy
7     // LOGLEVEL = 2
8     //       pokazuje ostrzezenia
9     //
```

```
10      // LOGLEVEL = 3
11      //       informuje o oczekiwaniu na poczatek nowej ramki
12      // LOGLEVEL = 4
13      //       informuje o zsynchronizowaniu ramki
14      //
15      parameter LOGLEVEL = 5,
16      parameter LOGLEVEL_SYNC = 5,
17      parameter LOGLEVEL_LINES = 5,
18
19      // Domyslnie 640x480
20      parameter V_S    = 16_700_000,
21      parameter V_FP   =    320_000,
22      parameter V_PW   =     64_040,
23      parameter V_BP   =    928_000,
24      parameter H_S    =     32_020,
25      parameter H_PW   =      3_860,
26      parameter H_FP   =        640,
27      parameter H_BP   =      1_900,
28
29      parameter LINES = 521
30  ) (
31      input [3:0] vga_r,
32      input [3:0] vga_g,
33      input [3:0] vga_b,
34      input vga_hsync,
35      input vga_vsync
36  );
```

Moduł wyczekuje początku nowej ramki. Dopiero po synchronizacji zaczyna sprawdzenia w modułach pomocniczych *Vga_Behav_Sync* oraz *Vga_Behav_Lines_Counter*.

Listing 105: Vga_Behav.v

```
47      reg    synchronized=1'b0;
48      initial begin
49          if( LOGLEVEL >= 3 )
50              $display("%t\t INFO3\t [ %m ] \t Oczekiwanie na poczatek nowej ramki.",
                    $time);
51
52          // Poczekaj na pierwszy puls synchronizacji ramki
53          // Nie sprawdza jednak dlugosci jego trwania, pomiar pulsu synchronizacji
                nastapi od drugiej ramki
54          monitor_vga_vsync.wait_for_low();
55          monitor_vga_vsync.wait_for_high();
56
57          // Zsynchronizowano, zacznij odbierac ramki
58          synchronized=1'b1;
59
60          if( LOGLEVEL >= 4 )
61              $display("%t\t INFO4\t [ %m ] \t Zsynchronizowano, rozpoczecie odbioru
                    ramek.", $time);
62      end
```

8.3.3 Synchronizacja

Moduł weryfikacji synchronizacji ramek składa się z dwóch podobnych bloków. Pierwszy weryfikuje uziemienie kolorów podczas i wokół synchronizacji ramek.

Listing 106: Vga_Behav_Synv.v

```
70      // Sprawdzanie synchronizacji ramek
71      always @(negedge vga_vsync) begin
72          if(synchronized) begin
73              if( LOGLEVEL >= 3 )
74                  $display("%t\t INFO3\t [ %m ] \t Rozpoczecie odbioru nowej ramki.",
                        $time);
75
76              fork begin
77                  // Dlugosc pulsu synchronizacji ramki
78                  // +1: wymaga symulacja
79                  monitor_vga_vsync.ensure_low_during( V_PW +1 );
80
81                  // Czas do nastepnej synchronizacji ramki
82                  // -1: kompensacja +1 z poprzedniego; nastepne -1 aby skonczyl
                        chwile przed nastepnym cyklem i zlapal liste wrazliwosci
83                  monitor_vga_vsync.ensure_high_during( V_S - V_PW -1 -1 );
84
85              end begin
86
87                  // Dlugosc pulsu synchronizacji ramki
88                  monitor_vga_colours_v.ensure_low_during( V_PW + V_BP );
89
90                  // Czas wyswietlania wszystkich kolejnych wierszy w ramce
91                  if( LOGLEVEL >= 4 )
92                      $display("%t\t INFO4\t [ %m ] \t Nadawanie wierszy", $time);
93                  #(V_S - V_FP - V_PW - V_BP + 13581);
94                  if( LOGLEVEL >= 5 )
95                      $display("%t\t INFO5\t [ %m ] \t Nadano wiersze", $time);
96
97                  // Czas do nastepnej synchronizacji ramki
98                  monitor_vga_colours_v.ensure_low_during( V_FP -13581 -1 );
99              end join;
100
101             $display();
102         end;
103     end
```

Drugi blok weryfikuje uziemienie kolorów podczas i wokół synchronizacji wierszy.

Listing 107: Vga_Behav_Synv.v

```
105     // Sprawdzanie synchronizacji wierszy
106     always @(negedge vga_hsync) begin
107         if(synchronized) begin
108             // logs.info1("Rozpoczecie odbioru nowego wiersza");
109             if( LOGLEVEL >= 3 )
110                 $display("%t\t INFO3\t [ %m ] \t Rozpoczecie odbioru nowego wiersza.
                        ", $time);
111
112             fork begin
113                 // Dlugosc pulsu synchronizacji wierszy
114                 // +1: wymaga symulacja
115                 monitor_vga_hsync.ensure_low_during( H_PW +1 );
116                 // Czas do nastepnej synchronizacji wierszy
117                 // -1: kompensacja +1 z poprzedniego; nastepne -1 aby skonczyl
                        chwile przed nastepnym cyklem i zlapal liste wrazliwosci
118                 monitor_vga_hsync.ensure_high_during( H_S - H_PW -1 -1 );
119
120             end begin
121
```

```
122             // Dlugosc pulsu synchronizacji wierszy
123             monitor_vga_colours_h.ensure_low_during( H_PW + H_BP );
124
125             // Czas wyswietlania wszystkich kolejnych pikseli w wierszu
126             if( LOGLEVEL >= 4 )
127                 $display("%t\t INFO4\t [ %m ] \t Nadawanie kolorow w wierszu",
                        $time);
128             #(H_S - H_FP - H_PW - H_BP);
129             if( LOGLEVEL >= 5 )
130                 $display("%t\t INFO5\t [ %m ] \t Nadano kolory w wierszu", $time)
                        ;
131
132             // Czas do nastepnej synchronizacji wierszy
133             monitor_vga_colours_h.ensure_low_during( H_FP -1 );
134
135         end join;
136
137         $display();
138     end
139 end
```

8.3.4 Zliczanie linii

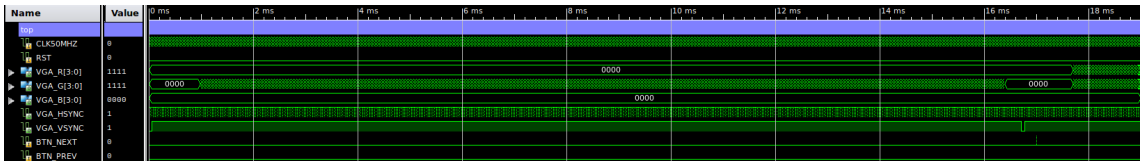
Osobny moduł zajmuje się zliczeniem linii i weryfikacją czy nastąpiło ich w ramce prawidłowa ilość.

Listing 108: Vga_Behav_Lines_Counter.v

```
40  // Zlicza ilosc odebranych wierszy w ramce i sprawdza czy jest wlasciwa
41  integer i=0;
42  always @(negedge vga_vsync)
43      if(synchronized) begin
44          i = 0;
45
46          // Przeczekaj cykl ramki
47          monitor_vga_vsync.wait_for_high();
48          monitor_vga_vsync.wait_for_low();
49
50          // Sprawdz ilosc odebranych linii, zakomunikuj warunkowo o rezultacie
51          if(i != LINES+1) begin
52              if(LOGLEVEL >= 1)
53                  $display("%t\t BLAD\t [ %m ] \t Pomiedzy synchronizacjami kolumn
                        wyslano %d linii. W cyklu powinno ich nastapic %d.", $time, i,
                         LINES);
54          end else
55              if(LOGLEVEL >= 3)
56                  $display("%t\t INFO3\t [ %m ] \t Odebrano wlasciwa ilosc linii %d
                        w cyklu.", $time, i);
57
58      end
59  always @(negedge vga_hsync)
60      if(synchronized) begin
61          i = i + 1;
62
63          // Przeczekaj cykl linii
64          monitor_vga_hsync.wait_for_low();
65          monitor_vga_hsync.wait_for_high();
66      end
```

8.3.5 Przebieg

Wycinek zrzutu ekranu pokazuje przebieg symulacji dla wybranego pierwszego koloru.



Rysunek 10: Przebieg symulacji

Zamieszczony jest fragment wyjścia logów dla domyślnych poziomów logowania.

Listing 109: Vga logs output

```
  0 INFO3 [ TopTest.vga_behav_ ]  Oczekiwanie na poczatek nowej ramki.
  0 INFO3 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostanie
      zmieniony. Obecny stan 'x' (0x x), spodziewany stan '0' (0x 0)
  0 INFO4 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostal
      zmieniony. Obecny stan '0' (0x 0)
  0 INFO3 [ TopTest.TopTestBench_.set_prev.state ]  Stan sygnalow zostanie
      zmieniony. Obecny stan 'x' (0x x), spodziewany stan '0' (0x 0)
  0 INFO4 [ TopTest.TopTestBench_.set_prev.state ]  Stan sygnalow zostal
      zmieniony. Obecny stan '0' (0x 0)
Finished circuit initialization process.
500000 INFO3 [ TopTest.TopTestBench_ ]  Zadanie nastepnego koloru
500000 INFO3 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostanie
    zmieniony. Obecny stan '0' (0x 0), spodziewany stan '1' (0x 1)
500000 INFO4 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostal
    zmieniony. Obecny stan '1' (0x 1)
750000 INFO3 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostanie
    zmieniony. Obecny stan '1' (0x 1), spodziewany stan '0' (0x 0)
750000 INFO4 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostal
    zmieniony. Obecny stan '0' (0x 0)
64071000 INFO4 [ TopTest.vga_behav_ ]  Zsynchronizowano, rozpoczecie odbioru
    ramek.
96090000 INFO3 [ TopTest.vga_behav_.vga_behav_sync_ ]  Rozpoczecie odbioru
    nowego wiersza.
101850000 INFO4 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadawanie kolorow w
    wierszu
127470000 INFO5 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadano kolory w wierszu

128110000 INFO3 [ TopTest.vga_behav_.vga_behav_sync_ ]  Rozpoczecie odbioru
    nowego wiersza.
133870000 INFO4 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadawanie kolorow w
    wierszu
159490000 INFO5 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadano kolory w wierszu

160130000 INFO3 [ TopTest.vga_behav_.vga_behav_sync_ ]  Rozpoczecie odbioru
    nowego wiersza.
165890000 INFO4 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadawanie kolorow w
    wierszu
191510000 INFO5 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadano kolory w wierszu

...

16714470000 INFO3 [ TopTest.vga_behav_.vga_behav_sync_ ]  Rozpoczecie odbioru
    nowego wiersza.
16714470000 INFO3 [ TopTest.vga_behav_.vga_behav_sync_ ]  Rozpoczecie odbioru
    nowej ramki.
```

```
16720230000 INFO4 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadawanie wiersza
16745850000 INFO5 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadano  wiersz

...

17000750000 INFO3 [ TopTest.TopTestBench_ ]  Zadanie nastepnego koloru
17000750000 INFO3 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow
    zostanie zmieniony. Obecny stan '0' (0x 0), spodziewany stan '1' (0x 1)
17000750000 INFO4 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostal
     zmieniony. Obecny stan '1' (0x 1)
17001000000 INFO3 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow
    zostanie zmieniony. Obecny stan '1' (0x 1), spodziewany stan '0' (0x 0)
17001000000 INFO4 [ TopTest.TopTestBench_.set_next.state ]  Stan sygnalow zostal
     zmieniony. Obecny stan '0' (0x 0)
17002010000 INFO5 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadano  wiersz

...

17675070000 INFO3 [ TopTest.vga_behav_.vga_behav_sync_ ]  Rozpoczecie odbioru
    nowego wiersza.
17680830000 INFO4 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadawanie wiersza
17706450000 INFO5 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadano  wiersz
17706510000 INFO4 [ TopTest.vga_behav_.vga_behav_sync_ ]  Nadawanie ramki
```

9 Bibliografia

Spartan 3an User Guide

Zbigniew Hajduk "Wprowadzenie do języka Verilog"

Rafał Baranowski "Mikrokontrolery AVR ATtiny w praktyce"

Pong P. Chu "FPGA Prototyping By Verilog Examples: Xilinx Spartan-3 Version"

http://edu.i-lo.tarnow.pl/inf/alg/002_struct/index.php

https://www.youtube.com/user/ElektroPrzewodnik/videos

http://www.elektroda.pl/rtvforum/topic1842513.html#8832060

http://www.asic-world.com/

http://testbench.in

http://johnroach.info/2011/01/15/getting-vga-output-using-vga-and-a-spartan-3an-board/

http://www.george-smart.co.uk/wiki/FPGA_PS2_Mouse

http://vhdldesign.blogspot.fr/2011/01/ps2-timing-diagram.html

http://www.eecg.toronto.edu/~jayar/ece241_08F/AudioVideoCores/ps2/ps2.html

http://www.computer-engineering.org/ps2protocol/

http://en.f-alpha.net/electronics/digital-electronics/digital-comparator/lets-go/
experiment-5-2-bit-magnitude-comparator.html

http://comp-phys.net/2014/03/22/including-vector-graphics-in-latex-using-includesvg/

http://forums.xilinx.com/t5/Spartan-Family-FPGAs/
ADC-on-Spartan-3AN-Starter-Kit-with-VHDL/td-p/179666

http://opencores.org/project,spi_master_slave

http://gft2009.forumactif.com/t40-dac-adc-fpga-vhdl-spartan-3e-xilinx

http://forums.xilinx.com

10 Dodatki

10.1 Makefile

Projekty można zsyntetyzować, przesymulować oraz przesłać na płytkę z wykorzystaniem programu GNU make. Skrypt wykorzystuje narzędzia Xilinxa dostępne z linii poleceń. W repozytorium podmodułu generic dostępny jest katalog 'makefile' z szablonami dla wszystkich projektów. Makefile'e projektów wyszczególniają swoje zależne pliki po czym załączają ogólny plik Makefile nazwany 'generic'.

Listing 110: Makefile

```
SHELL := /bin/bash -O extglob

top=Top
part=xc3s700an-fgg484-5

# FIXME set to your value
isedir  ?= /opt/Xilinx/14.3/ISE_DS
xil_env ?= . $(isedir)/settings32.sh

intstyle=-intstyle ise

ifneq ($(wildcard default_wcfg),)
  wcfg_view=-view ../wcfg/`cat default_wcfg`
endif

# all: bit configure

bit: $(synt)
  for src in $(synt); do echo "verilog work $$src"; done | sort -u > Top.prj
  [ -d xst/projnav.tmp ] || mkdir -p xst/projnav.tmp
  $(xil_env); xst $(intstyle) -ifn "../../generic/makefile/config.xst" -ofn "$(top).syr"
  $(xil_env); ngdbuild $(intstyle) -dd _ngo -nt timestamp -uc "../$(top).ucf" -p $(part) $(top).ngc $(top).ngd
  $(xil_env); map $(intstyle) -p $(part) -cm area -ir off -pr off -c 100 -o Top_map.ncd $(top).ngd $(top).pcf
  $(xil_env); par -w $(intstyle) -ol high -t 1 Top_map.ncd $(top).ncd $(top).pcf
  $(xil_env); trce $(intstyle) -v 3 -s 5 -n 3 -fastpaths -xml $(top).twx $(top).ncd -o $(top).twr $(top).pcf
  $(xil_env); bitgen $(intstyle) -f "../../generic/makefile/config.ut" $(top).ncd

configure: bit
  $(xil_env); impact -batch <<< "` cat ../../generic/makefile/impact_batch.tpl | sed "s:BITSTREAM_FILE:$$PWD/$(top).bit:" `"

sim: $(sim)
  for src in $(sim); do echo "verilog work $$src"; done | sort -u > TopTest_beh.prj
  $(xil_env); fuse $(intstyle) -d SIM -incremental -lib unisims_ver -lib unimacro_ver -lib xilinxcorelib_ver -o TopTest_isim_beh.exe -prj TopTest_beh.prj TopTest
  $(xil_env); "./TopTest_isim_beh.exe" $(intstyle) -gui -tclbatch isim.cmd -wdb "TopTest_isim_beh.wdb" $(wcfg_view)

simr: $(sim)
  for src in $(sim); do echo "verilog work $$src"; done | sort -u > TopTest_beh.
```

```
        prj
38    $(xil_env); fuse $(intstyle) -d SIM -incremental -lib unisims_ver -lib
        unimacro_ver -lib xilinxcorelib_ver -o TopTest_isim_beh.exe -prj
        TopTest_beh.prj TopTest
39    $(xil_env); "./TopTest_isim_beh.exe" $(intstyle) -tclbatch isim.cmd -wdb "
        TopTest_isim_beh.wdb" $(wcfg_view)
40
41  distclean:
42    rm -rf !(*.xise|Makefile|isim.cmd|default_wcfg)
```

Listing 111: config.ut

```
1  -w
2  -g DebugBitstream:No
3  -g Binary:no
4  -g CRC:Enable
5  -g Reset_on_err:No
6  -g ConfigRate:25
7  -g ProgPin:PullUp
8  -g DonePin:PullUp
9  -g TckPin:PullUp
10 -g TdiPin:PullUp
11 -g TdoPin:PullUp
12 -g TmsPin:PullUp
13 -g UnusedPin:PullDown
14 -g UserID:0xFFFFFFFF
15 -g StartUpClk:CClk
16 -g DONE_cycle:4
17 -g GTS_cycle:5
18 -g GWE_cycle:6
19 -g LCK_cycle:NoWait
20 -g Security:None
21 -g DonePipe:No
22 -g DriveDone:No
23 -g en_sw_gsr:No
24 -g en_porb:Yes
25 -g drive_awake:No
26 -g sw_clk:Startupclk
27 -g sw_gwe_cycle:5
28 -g sw_gts_cycle:4
```

Listing 112: config.xst

```
1  set -tmpdir "xst/projnav.tmp"
2  set -xsthdpdir "xst"
3  run
4  -ifn Top.prj
5  -ifmt mixed
6  -ofn Top
7  -ofmt NGC
8  -p xc3s700an-5-fgg484
9  -top Top
10 -opt_mode Speed
11 -opt_level 1
12 -iuc NO
13 -keep_hierarchy No
14 -netlist_hierarchy As_Optimized
15 -rtlview Yes
16 -glob_opt AllClockNets
17 -read_cores YES
```

```
-write_timing_constraints NO
-cross_clock_analysis NO
-hierarchy_separator /
-bus_delimiter <>
-case Maintain
-slice_utilization_ratio 100
-bram_utilization_ratio 100
-verilog2001 YES
-fsm_extract YES -fsm_encoding Auto
-safe_implementation No
-fsm_style LUT
-ram_extract Yes
-ram_style Auto
-rom_extract Yes
-mux_style Auto
-decoder_extract YES
-priority_extract Yes
-shreg_extract YES
-shift_extract YES
-xor_collapse YES
-rom_style Auto
-auto_bram_packing NO
-mux_extract Yes
-resource_sharing YES
-async_to_sync NO
-mult_style Auto
-iobuf YES
-max_fanout 500
-bufg 24
-register_duplication YES
-register_balancing No
-slice_packing YES
-optimize_primitives NO
-use_clock_enable Yes
-use_sync_set Yes
-use_sync_reset Yes
-iob Auto
-equivalent_register_removal YES
-slice_utilization_ratio_maxmargin 5
```

../projects/generic/makefile/impact_batch.tpl

```
setMode -bs
setMode -sm
setMode -hw140
setMode -acecf
setMode -acempm
setMode -pff
setMode -bs

setCable -port auto

Identify -inferir
identifyMPM

assignFile -p 1 -file "BITSTREAM_FILE"
Program -p 1 -onlyFpga

quit
```

10.1.1 DAC

../projects/dac/project/Makefile

```
# project specific synt modules

synt += ../Top.v
synt += ../Controller.v
synt += ../DacSpi.v
synt += ../../generic/Debouncer.v

# generic synt modules

synt += ../../generic/Counter.v
synt += ../../generic/ModClk.v
synt += ../../generic/Spi.v
synt += ../../generic/Serial.v
synt += ../../generic/Shiftreg.v

# project specific sim modules

sim += $(synt)
sim += ../sim/TopTest.v
sim += ../sim/TopTestBench.v
sim += ../sim/DacLTC2624-behav.v

# generic sim modules

sim += ../../generic/sim/Clock.v
sim += ../../generic/sim/Reset.v
sim += ../../generic/sim/Set.v
sim += ../../generic/sim/Monitor.v

# generic makefile

include ../../generic/makefile/generic
```

10.1.2 Rotor

../projects/rotor/project/Makefile

```
synt += ../Top.v
synt += ../Controller.v
synt += ../Rotor.v
synt += ../../generic/Counter.v
synt += ../../generic/Debouncer.v

sim += $(synt)
sim += ../sim/TopTest.v
sim += ../sim/TopTestBench.v
sim += ../sim/Rotor_behav.v
sim += ../../generic/sim/Clock.v
sim += ../../generic/sim/Reset.v
sim += ../../generic/sim/Set.v

include ../../generic/makefile/generic
```

10.1.3 Rs232

../projects/rs232/project/Makefile

```
1  synt += ../Top.v
2  synt += ../Rs232Echo.v
3  synt += ../Rs232Tx.v
4  synt += ../Rs232Rx.v
5  synt += ../../generic/BaudRateGenerator.v
6  synt += ../../generic/Serial.v
7  synt += ../../generic/Counter.v
8  synt += ../../generic/Shiftreg.v
9
10 sim += $(synt)
11 sim += ../sim/TopTest.v
12 sim += ../sim/Rs232.v
13 sim += ../sim/Rs232_behav.v
14 sim += ../../generic/sim/Clock.v
15 sim += ../../generic/sim/Reset.v
16 sim += ../../generic/sim/Monitor.v
17 sim += ../../generic/sim/Set.v
18
19 include ../../generic/makefile/generic
```

10.1.4 VGA

../projects/vga/project/Makefile

```
1  synt += ../Top.v
2  synt += ../Controller.v
3  synt += ../Sync.v
4  synt += ../../generic/Counter.v
5  synt += ../../generic/Debouncer.v
6
7  sim += $(synt)
8  sim += ../sim/TopTest.v
9  sim += ../sim/TopTestBench.v
10 sim += ../sim/Vga_Behav.v
11 sim += ../sim/Vga_Behav_Lines_Counter.v
12 sim += ../sim/Vga_Behav_Sync.v
13 sim += ../../generic/sim/Clock.v
14 sim += ../../generic/sim/Reset.v
15 sim += ../../generic/sim/Monitor.v
16 sim += ../../generic/sim/Set.v
17
18 include ../../generic/makefile/generic
```