Języki Opisu Sprzętu

Projekt: Elektroniczny Sejf Hotelowy
Dokumentacja

Arkadiusz Kasprzak
Jarosław Cierpich
Wydział Fizyki i Informatyki Stosowanej
Informatyka Stosowana

1 grudnia 2019

Spis treści

1 Wstęp

Niniejszy dokument stanowi dokumentację projektu **Elektroniczny Sejf Hotelowy** wykonanego w ramach przedmiotu **Języki Opisu Sprzętu** (WFiIS AGH) przez Jarosława Cierpicha i Arkadiusza Kasprzaka. Dokument ten zawiera m.in. założenia projektowe oraz opis wymaganej funkcjonalności, dokumentację przeznaczoną dla użytkownika projektu, dokumentację techniczną, analizę procesu syntezy oraz opis procedury testowania.

2 Projekt

Ten rozdział poświęcony został opisowi założeń projektowych oraz wymaganej funkcjonalności.

2.1 Założenia projektowe

Projekt dostarczać ma funkcji elektronicznego sejfu hotelowego, to znaczy pozwalać ma na otwieranie go za pomocą z góry ustalonego szyfru oraz późniejsze zamknięcie go. Projekt ma być zrealizowany na płytce rozwojowej **Zedboard** (do Xilinx Zynq-7000). Ze względu na okoliczności wykonywania projektu (projekt jako zaliczenie przedmiotu) przyjęte zostały pewne uproszczenia:

- czujnik zamknięcia sejfu zastąpiony zostaje przełącznikiem obsługiwanym przez użytkownika
- szyfr jest parametrem projektu - nie jest możliwa jego modyfikacja na płytce bez powtórzenia procesu syntezy i implementacji
- ruch rygla reprezentowany jest za pomocą dwóch diod LED dostępnych na płytce

Do realizacji projektu użyty ma być język **SystemVerilog** oraz środowisko **Xilinx Vivado**. Szyfr składać ma się z trzech liczb z przedziału [0; 32). Liczby mają być wprowadzane przez użytkownika za pomocą pokrętła, przy czym kierunek obrotu pokrętła wskazuje, która liczba jest aktualnie wprowadzana: ruch zgody ze wskazówkami zegara oznacza pierwszą lub trzecią liczbę, ruch przeciwny do wskazówek zegara - drugą liczbę. Wprowadzenie nieprawidłowej liczby ma przerywać proces podawania szyfru - tzn. układ nie czeka, aż wprowadzony zostanie cały szyfr. Aktualnie wprowadzona liczba ma być prezentowana za pomocą wyświetlacza OLED. Układ ma być w miarę możliwości pozbawiony błędów związanych z drganiami styków czy niestandardowym działaniem użytkownika.

2.2 Wymagana funkcjonalność

Od projektu wymaga się dostarczenia następującej funkcjonalności:

- możliwość wprowadzenia przez użytkownika poprawnego szyfru składającego się z trzech liczb z zakresu [0; 32) za pomocą pokrętła
- możliwość wprowadzenia przez użytkownika niepoprawnego szyfru, co automatycznie przerwać ma proces otwierania sejfu
- możliwość monitorowania przez użytkownika aktualnego stanu otwarcia sejfu - za pomocą diod LED
- możliwość monitorowania przez użytkownika aktualnie wprowadzanej wartości - za pomocą wyświetlacza OLED
- możliwość rozpoczęcia procesu otwierania sejfu - za pomocą przycisku *Open*

- możliwość zamknięcia sejfu - za pomocą przycisku *Close*
- możliwość ustalenia aktualnej pozycji rygla sejfu - za pomocą przełącznika - zastępuje to czujnik zamknięcia sejfu
- możliwość łatwego zaprogramowania nowego szyfru - zmiana trzech wartości w kodzie

3 Dokumentacja użytkownika

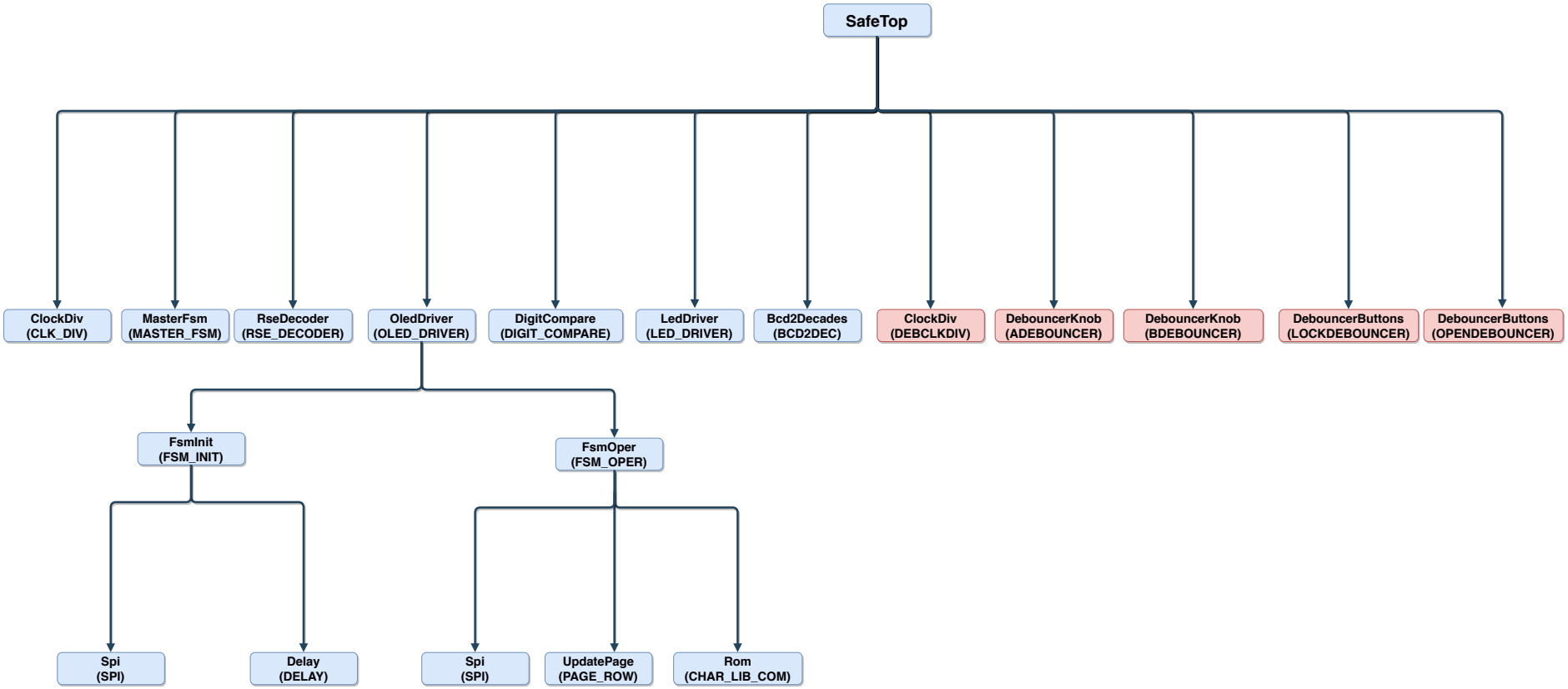
4 Dokumentacja techniczna

Rozdział opisuje szczegóły implementacji projektu - zarówno jeśli chodzi o warstwę sprzętową (hardware), jak i o oprogramowanie (software).

4.1 Warstwa Hardware

4.2 Warstwa Software - architektura

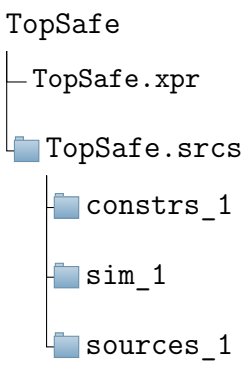
W tej sekcji omówiona zostanie architektura projektu - zarówno ze względu na przepływ danych między komponentami, jak i na ich hierarchię w projekcie.
Rysunek 1 przestawia hierarchię instancji modułów w projekcie, z wyróżnieniem instancji modułów odpowiedzialnych za tłumienie drgań na przyciskach i pokrętle - instancje te nie są generowane w czasie symulacji behawioralnej.
Diagram **DEJ DIAGRAM** przedstawia strukturę zastosowanych w projekcie modułów wraz z przepływem danych między nimi.



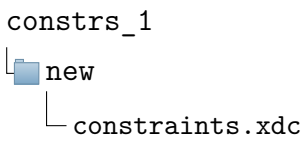
Rysunek 1: Hierarchia instancji modułów zastosowana w projekcie. Nazwy w nawiasach to nazwy instancji, przed nawiasami - nazwy modułów. Na czerwono zaznaczono instancje, które nie są generowane podczas symulacji behawioralnej (wygaszanie drgań)

4.3 Warstwa Software - struktura projektu

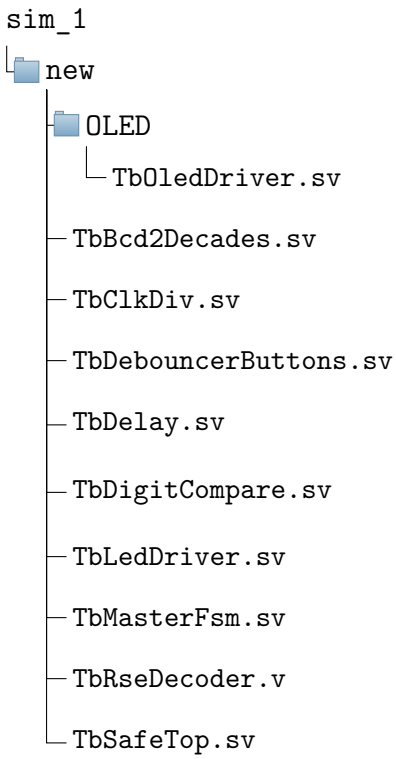
Wysokopoziomowa struktura katalogów i plików przedstawiona została poniżej:



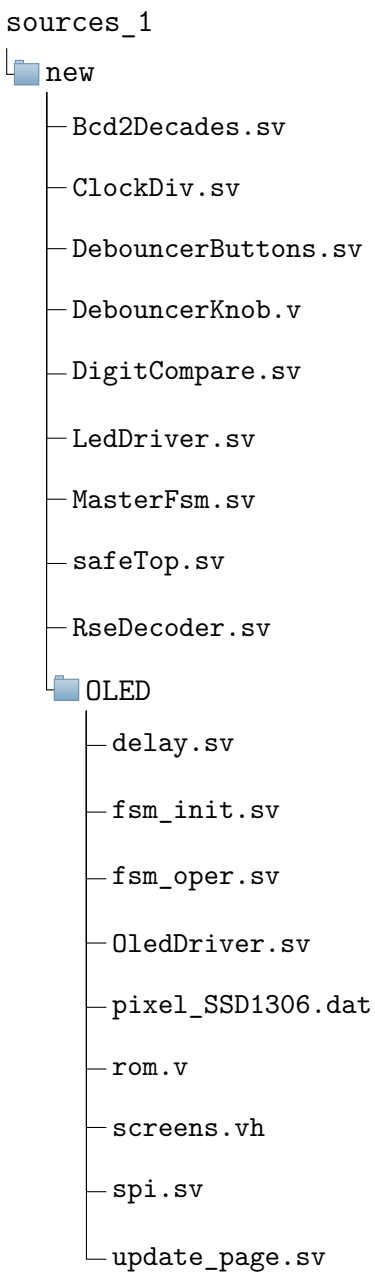
Jest to standardowy podział dla narzędzia **Xilinx Vivado**. Plik **TopSafe.xpr** jest głównym plikiem projektu. Struktura wewnętrzna katalogu **constrs_1** jest następująca:



Zawiera on plik *.xdc* z tzw. Design Constraints. Katalog **sim_1** zawiera moduły testujące (tzw. Testbench) i jego strukturę przedstawiono poniżej:



Struktura katalogu **sources_1** zawierającego kod źródłowy projektu:



4.4 Warstwa Software - parametry i moduły projektu

Ten podrozdział opisuje część implementacji projektu - zastosowane parametry oraz moduły napisane w języku SystemVerilog.

4.4.1 Parametry projektu

Główny moduł projektu udostępnia następujące **parametry** pozwalające na modyfikację działania sejfu:

- **slowClockPeriodLength** - współczynnik oznaczający wartość dzielnika częstotliwości zegara, którym taktowany jest sejf (z wyjątkiem debouncerów i wyświetlacza OLED). Wartość domyślna: 100000.

- **debouncerClockPeriodLength** - współczynnik oznaczający wartość dzielnika częstotliwości zegara, którym taktowane są debouncery w projekcie. Wartość domyślna: 300007.
- **areDebouncersUsed** - flaga włączająca lub wyłączająca generację debouncerów w projekcie. Wartość domyślna: 1 - debouncery mają zostać wygenerowane.
- **firstCodeNumber** - pierwsza liczba szyfru. Wartość domyślna: 15.
- **secondCodeNumber** - druga liczba szyfru. Wartość domyślna: 30.
- **thirdCodeNumber** - trzecia liczba szyfru. Wartość domyślna: 9.

4.4.2 Lista modułów projektu

Projekt składa się z następujących **modułów**:

- **SafeTop** - główny moduł projektu
- **Bcd2Decades** - licznik BCD (Binary Coded Decimal) o dwóch dekadach
- **ClockDiv** - dzielnik zegara
- **DebouncerButtons** - układ wygaszający drgania na przyciskach
- **DebouncerKnob** - układ wygaszający drgania na pokrętle
- **DigitCompare** - układ porównujący wprowadzone przez użytkownika liczby z szyfrem
- **LedDriver** - sterownik diod LED
- **MasterFsm** - główna logika sejfu
- **RseDecoder** - układ odpowiedzialny za komunikację z pokrętłem
- Moduły odpowiedzialne za **obsługę wyświetlacza OLED**:
  - **OledDriver** - główny moduł odpowiedzialny za obsługę wyświetlacza
  - **Delay** - moduł odpowiedzialny za generowanie opóźnień
  - **FsmInit** - moduł odpowiedzialny za przeprowadzenie procedury inicjalizacji wyświetlacza OLED
  - **FsmOper** - moduł odpowiedzialny za wysyłanie danych do wyświetlacza OLED
  - **Rom** - transkoder działający na zasadzie pamięci stałej - odpowiada za dostarczenie *czcionki*
  - **Spi** - implementacja uproszczonego (jednokierunkowego) protokołu SPI
  - **UpdatePage** - moduł powiązany z FsmOper

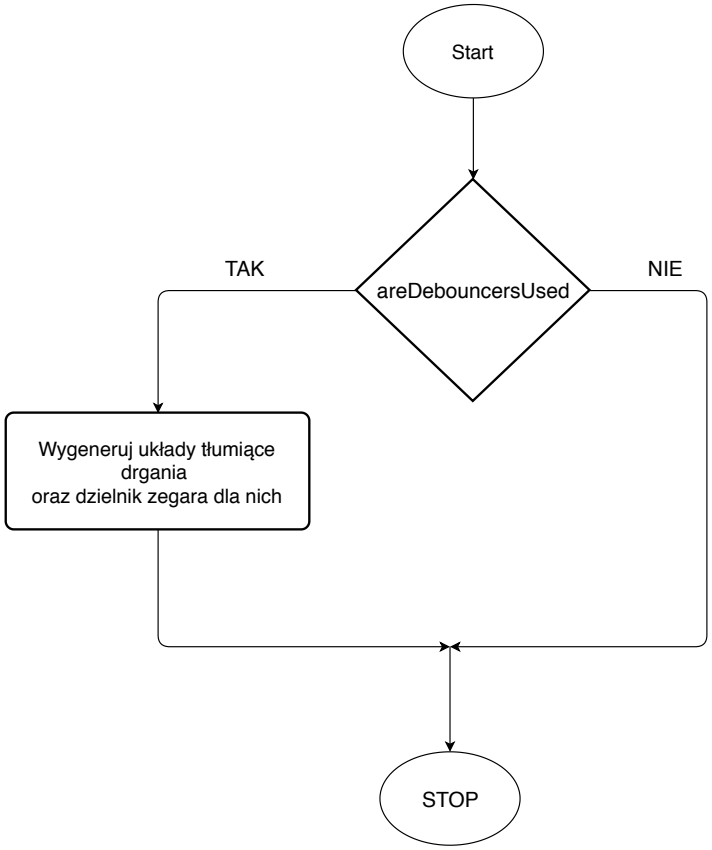
Do implementacji modułów obsługujących wyświetlacz OLED wykorzystany został w dużej części kod przygotowany w ramach zajęć laboratoryjnych z przedmiotu Języki Opisu Sprzętu. Dalsza część dokumentacji zawiera opis działania najważniejszych modułów projektu.

4.4.3 Moduł SafeTop

Jest to moduł usytuowany najwyżej w hierarchii projektu - odpowiedzialny jest on za połączenie ze sobą pozostałych modułów w projekcie za pomocą ich sygnałów wejściowych i wyjściowych. Ponadto zawiera on logikę odpowiedzialną za generowanie instancji debouncerów tylko, gdy do modułu przekazany zostanie parametr **areDebouncersUsed** o wartości 1 (logiczna prawda). Zostało to osiągnięte za pomocą **bloku generate** połączonego z instrukcją sterującą if-else. Listing 1 przestawia implementację opisywanego rozwiązania. Schemat blokowy przedstawiony na rysunku 2 ilustruje ideę zastosowanego algorytmu.

Listing 1: Przykład zastosowania bloku generate do warunkowego generowania instancji debouncerów w module SafeTop

```
generate
    if (areDebouncersUsed) begin : debGen

        // clock divider for debouncers
        wire debSlowClk;
        ClockDiv #(.clockPeriodLength(debouncerClockPeriodLength))
        DEBCLKDIV
        (
            .clk(clk), .rst(rst),
            .slowClk(debSlowClk)
        );

        // a signal debouncer
        DebouncerKnob #(.N(3)) ADEBOUNCER (
            .clk(debSlowClk), .rst(rst), .in(a), .out(aKnob)
        );

        // b signal debouncer
        DebouncerKnob #(.N(3)) BDEBOUNCER (
            .clk(debSlowClk), .rst(rst), .in(b), .out(bKnob)
        );

        // open signal debouncer
        DebouncerButtons #(.registerSize(3)) OPENDEBOUNCER (
            .clk(debSlowClk), .rst(rst), .in(open), .out(openDeb)
        );

        // lock signal debouncer
        DebouncerButtons #(.registerSize(3)) LOCKDEBOUNCER (
            .clk(debSlowClk), .rst(rst), .in(lock), .out(lockDeb)
        );

    end
    else begin : debNoGen
        assign aKnob = a;
        assign bKnob = b;
        assign openDeb = open;
        assign lockDeb = lock;
    end
endgenerate
```



Rysunek 2: Schemat blokowy - idea działania algorytmu generującego debouncery w module SafeTop

4.4.4 Moduł Bcd2Decades

Jest to moduł stanowiący licznik modulo 32 zwracający wartość w formacie BCD (Binary Coded Decimal) o dwóch dekadach. Założenie formatu BCD polega na kodowaniu binarnie każdej z cyfr reprezentacji dziesiętnej wejścia osobno, tzn. np. dla wejścia (w postaci dziesiętnej) 15 osobno zakodowana zostanie cyfra 1 (co daje wynik 0001) oraz osobno cyfra 5 (co daje wynik 0101). Ostatecznie otrzymujemy więc wynik 0001 0101. Moduł Bcd2Decades składa się ze zwykłego licznika binarnego modulo 32, przerzutnika wyjściowego oraz z procesu kombinacyjnego implementującego algorytm konwersji wartości binarnej na postać BCD. Zastosowano następujący algorytm konwersji:

1. Inicjalizujemy wartości kolumn (dziesiątki i jednostki) wartością zero.
2. Jeśli wartość w którejkolwiek z kolumn jest większa lub równa 5, dodajemy do niej wartość 3.
3. Wykonujemy przesunięcie bitowe o jeden w lewo, przy czym najstarszy bit jednostek staje się najmłodszym bitem dziesiątek i analogicznie najstarszy bit wartości binarnej staje się najmłodszym bitem jednostek - traktujemy je tak, jakby były ustawione w następujący sposób: {dziesiątki, jednostki, wartość binarna}
4. Jeśli przeprowadzono 5 iteracji (5 razy dokonano przesunięcia bitowego - liczba 5 to liczba bitów, za pomocą której reprezentujemy wartość binarną), to kończymy działanie. W przeciwnym razie powrót do kroku numer 2.

Implementacja algorytmu przestawiona została na listingu 2. Przykładowy jego przebieg zawiera natomiast tabela 1.

Listing 2: Implementacja algorytmu zamiany wartości binarnej na postać BCD.

```
reg [3:0] bcd0tmp;
reg [3:0] bcd1tmp;
always@* begin
    automatic integer i = 0;
    bcd0tmp = 4'b0000;
    bcd1tmp = 4'b0000;
    for (i = 0; i < 5; i = i + 1) begin
        bcd0tmp = (bcd0tmp >= 5 ? bcd0tmp + 4'd3 : bcd0tmp);
        bcd1tmp = (bcd1tmp >= 5 ? bcd1tmp + 4'd3 : bcd1tmp);

        bcd1tmp = bcd1tmp << 1;
        bcd1tmp[0] = bcd0tmp[3];
        bcd0tmp = bcd0tmp << 1;
        bcd0tmp[0] = binary_counter[bits-i-1];
    end
end
```

Tablica 1: Prezentacja działania algorytmu konwersji wartości binarnej do postaci BCD na przykładzie liczby 17.

Dziesiątki	Jednostki	Postać binarna	Operacja
0000	0000	10001	
0000	0001	00010	<<#1
0000	0010	00100	<<#2
0000	0100	01000	<<#3
0000	1000	10000	<<#4
0000	1011	10000	dodaj 3
0001	0111	00000	<<#5

4.4.5 Moduł RseDecoder

4.4.6 Moduł MasterFsm

5 Analiza raportu syntezy

Rozdział zawiera analizę raportu syntezy wygenerowanego przez narzędzie **Xilinx Vivado 2019.1**. Analizie poddane zostały:

- Sposób kodowania stanów
- Użyte zasoby, np.: ROM, komórki logiczne

5.1 Kodowanie stanów

- W większości przypadków zastosowane zostało kodowanie typu **one-hot**, co jest zwykle zachowaniem domyślnym dla narzędzi Xilinx
- Dotyczy to następujących modułów: MasterFsm, RseDecoder, Spi, FsmOper, FsmInit
- Tabela 2 przedstawia przykładowe kodowanie **one-hot**

Tablica 2: Przykład kodowania typu one-hot zaistniały w wyniku syntezy projektu (moduł MasterFsm)

Stan (State)	Nowe kodowanie	Poprzednie kodowanie
safeLocked	000000001	00000000000000000000000000000000
startOpening	000000010	00000000000000000000000000000001
waitKnobCounterEnable	000000100	00000000000000000000000000000010
firstNumberGood	000001000	00000000000000000000000000000011
secondNumberGood	000010000	00000000000000000000000000000100
thirdNumberGood	000100000	00000000000000000000000000000101
safeUnlocked	001000000	00000000000000000000000000000111
safeLockGood	010000000	00000000000000000000000000001000
wrongNumber	100000000	00000000000000000000000000000110

- W przypadku pozostałych maszyn stanów (Delay, UpdatePage, OledDriver) zostało zastosowane kodowanie sekwencyjne.
- Tabela 3 przedstawia przykładowe kodowanie **sekwencyjne**:

Tablica 3: Przykład kodowania typu sequential zaistniały w wyniku syntezy projektu (moduł UpdatePage)

Stan (State)	Nowe kodowanie	Poprzednie kodowanie
idle	000	00000000000000000000000000000000
ClearDC	001	00000000000000000000000000000001
SendCmd	010	00000000000000000000000000000010
Transition1	011	00000000000000000000000000000100
Transition2	100	00000000000000000000000000000101
Transition5	101	00000000000000000000000000000110
SetDC	110	00000000000000000000000000000011

5.2 Użycie zasobów

- Użyty został jeden blok ROM o rozmiarach 1024x8. Przeznaczony on jest na przechowywanie informacji o czcionce - informacje wczytane z pliku **pixel_SSD1306.dat**
- Cały projekt składa się z **635** komórek logicznych (cells), w tym największą ilość stanowią komórki typu **FDCE** oraz **LUT** z różną ilością argumentów.
- Tabela 4 przedstawia użycie komórek w poszczególnych modułach projektu.

Tablica 4: Zużycie komórek logicznych przez poszczególne moduły projektu

	Instancja	**Moduł**	**Komórki (Cells)**
1	top		635
2	- BCD2DEC	Bcd2Decades	30
3	- CLK_DIV	ClockDiv__parameterized0	57
4	- LED_DRIVER	LedDriver	2
5	- MASTER_FSM	MasterFsm	37
6	- OLED_DRIVER	OledDriver	375
7	- - FSM_INIT	FsmInit	180
8	- - - DELAY	Delay	71
9	- - - SPI	Spi_2	52
10	- - FSM_OPER	FsmOper	184
11	- - - CHAR_LIB_COM	Rom	1
12	- - - PAGE_ROW	UpdatePage	27
13	- - - SPI	Spi	47
14	- RSE_DECODER	RseDecoder	23
15	- \debGen.ADEBOUNCER	DebouncerKnob	6
16	- \debGen.BDEBOUNCER	DebouncerKnob_0	6
17	- \debGen.DEBCLKDIV	ClockDiv	64
18	- \debGen.LOCKDEBOUNCER	DebouncerButtons	6
19	- \debGen.OPENDEBOUNCER	DebouncerButtons_1	6

6 Testy

Ostatni rozdział poświęcony został procesowi testowania projektu - w tym przygotowanym modułom testowym oraz procesowi testowania manualnego.

6.1 Moduły testujące

Projekt zawiera **10** modułów testowych (tzw. moduły *Testbench*). Umożliwiają one przeprowadzenie symulacji działania modułów projektu. Testy przeprowadzone zostały za pomocą dwóch typów symulacji:

- symulacja behawioralna (*behavioural simulation*)
- symulacja po syntezie z uwzględnieniem parametrów czasowych (*post-synthesis timing simulation*)

Podstawowa struktura większości modułów *Testbench* jest podobna - składają się one z:

- deklaracji parametrów wejściowych modułu (jeśli takie są)
- deklaracji zmiennych stanowiących wejścia i wyjścia testowanego modułu oraz zmiennej odpowiadającej za *Global System Reset - GSR*
- instancji testowanego modułu (*UUT - Unit Under Test*)
- generacji sygnałów wejściowych testowanego modułu (w tym zwykle sygnału zegara i resetu)

Listing 3 ilustruje opisaną powyżej strukturę.

Listing 3: Uproszczona struktura wykonanych modułów testujących

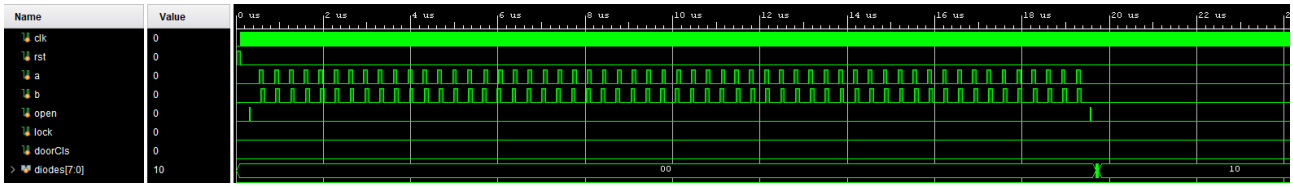
```
module TbExample();

    // parametry
    localparam mod = 3;

    // wejscia
    reg clk, rst;
    reg in1;

    // wyjscia
    reg [3:0] out1;

    // ...

    // GSR - Global System Reset
    wire gsr = glbl.GSR;

    // UUT - Unit Under Test
    ExampleModule #(.mod(mod)) EXAMPLE (
        .clk(clk), .rst(rst), .in1(in1), .out1(out1));

    // generacja sygnalow wejsciowych

    // zegar
```

```
    initial begin
        clk = 1'b0;
        @(negedge gsr);
        forever #5 clk = ~clk;
    end

    // reset
    initial begin
        rst = 1'b1;
        @(negedge gsr);
        #5 rst = 1'b0;
    end

    // in1
    initial begin
        // kod generujacy wartosci sygnalu in1
    end

    // ...

endmodule
```

W dalszej części tego podrozdziału omówione zostaną poszczególne moduły testujące oraz wyniki przeprowadzonych symulacji behawioralnych.

6.1.1 Testy modułu SafeTop

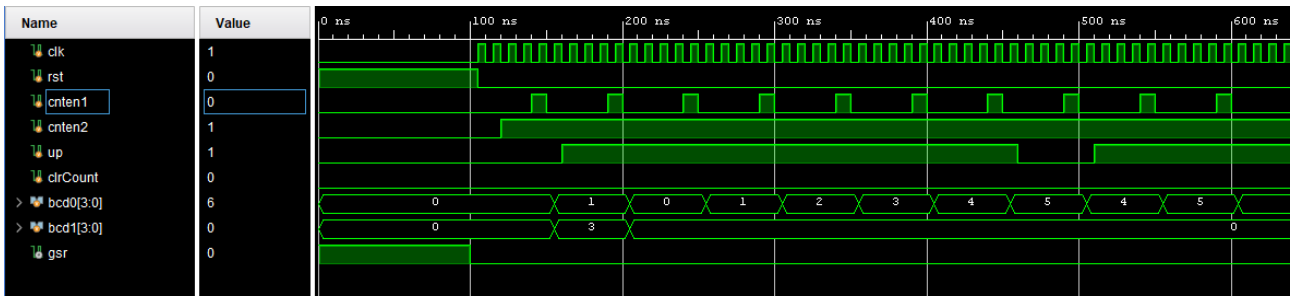
Test modułu SafeTop zakłada wyłączenie generacji debouncerów w module. Rysunek 3 przestawia wartości sygnałów uzyskane podczas testu. Podczas testu wygenerowana została odpowiednia kombinacja sygnałów wejściowych *a* i *b* (sygnały pochodzące z pokrętła), by wprowadzić poprawny szyfr sejfu. Po wprowadzeniu ostatniej wartości oraz wygenerowaniu sygnału open wartość sygnału *diodes* zmienia się na taką, która wskazuje, że sejf został otwarty. Zachowanie modułu jest więc zgodne z oczekiwanym.



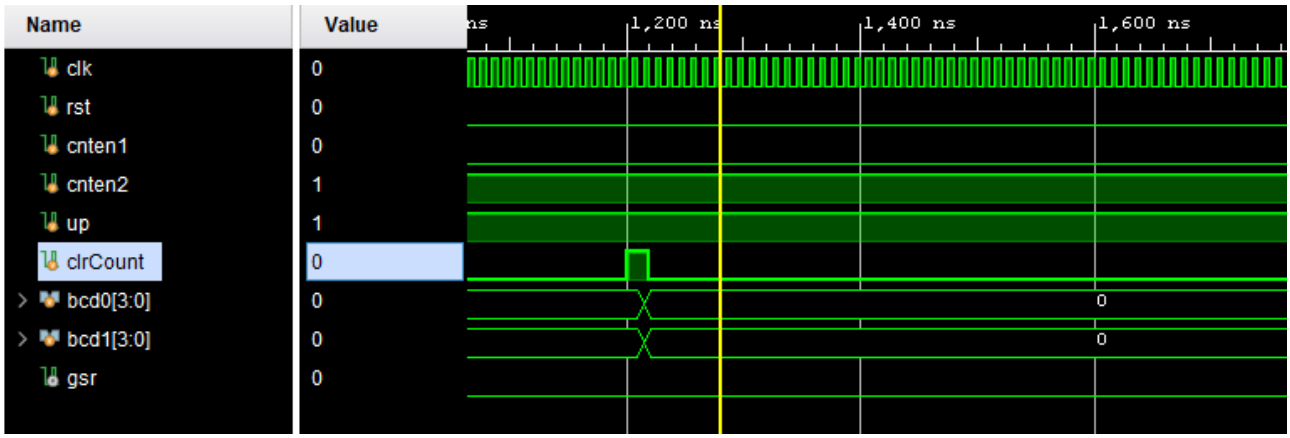
Rysunek 3: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły SafeTop

6.1.2 Testy modułu Bcd2Decades

Moduł Bcd2Decades odpowiedzialny jest za zliczanie sygnałów wejściowych oraz reprezentację licznika w postaci Binary Coded Decimal. Na każdym rosnącym zboczu zegara próbkowane są linie *cnten1* oraz *cnten2*. Jeśli na obu z nich występuje stan wysoki, to licznik jest modyfikowany zgodnie z kierunkiem wskazanym przez sygnał *up*. Rysunek 4 przedstawia pomyślnie przeprowadzony test opisanej funkcjonalności. Możliwe jest również wyzerowanie licznika za pomocą sygnału *clrCount* - dokładnie takie działanie można zaobserwować na rysunku 5.



Rysunek 4: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły Bcd2Decades



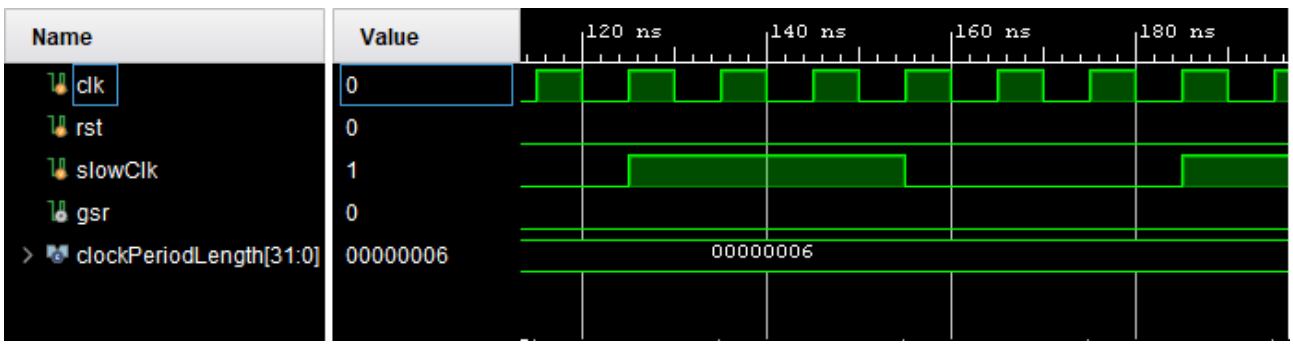
Rysunek 5: Reakcja modułu Bcd2Decades na stan wysoki sygnału clrCount.

6.1.3 Testy modułu RseDecoder

6.1.4 Testy modułu MasterFsm

6.1.5 Testy modułu ClockDiv

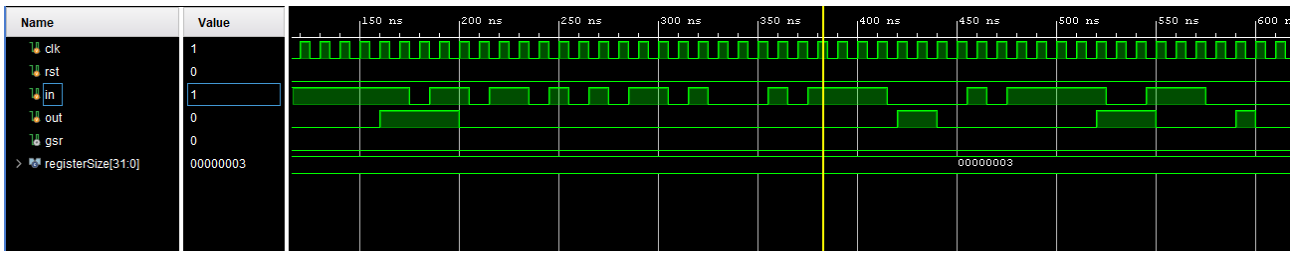
Test modułu ClockDiv jest bardzo prosty. Jego rezultaty przedstawione zostały na rysunku 6. Na wejście modułu podawany jest zegar. Długość okresu wynikowego zegara (*slowClk*) ustalona została za pomocą parametru na sześć okresów zegara wejściowego. Test wskazuje więc, że działanie modułu jest zgodne z oczekiwaniami - okres sygnału *slowClk* jest sześć razy dłuższy od okresu sygnału *clk*.



Rysunek 6: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły ClockDiv

6.1.6 Testy modułu DebouncerButtons

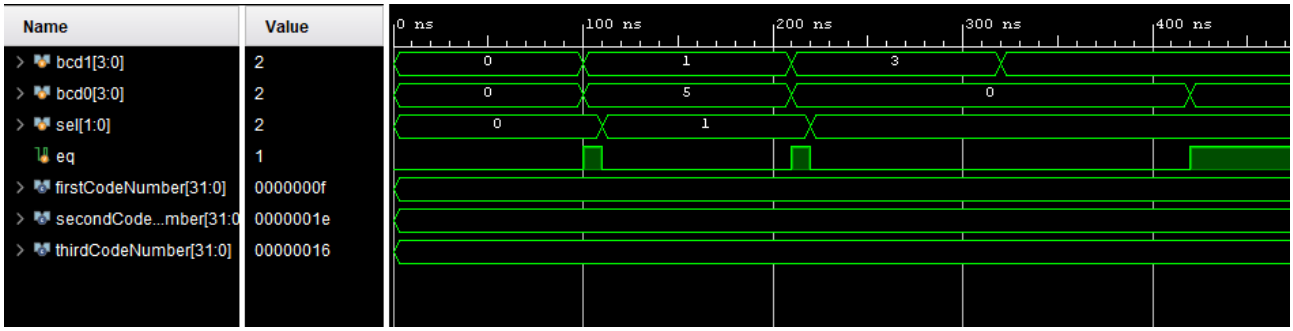
Rysunek 7 przedstawia przebieg symulacji modułu DebouncerButtons. Moduł ten próbkuje na każdym rosnącym zboczu zegara wartość na linii *in*. Jeśli w ciągu 3 kolejnych próbek linia utrzymuje stan wysoki, na wyjściu *out* generowany jest sygnał o wartości 1. Obserwacja przebiegów pozwala stwierdzić, że moduł zachowuje się zgodnie z oczekiwaniami.



Rysunek 7: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły DebouncerButtons

6.1.7 Testy modułu DigitCompare

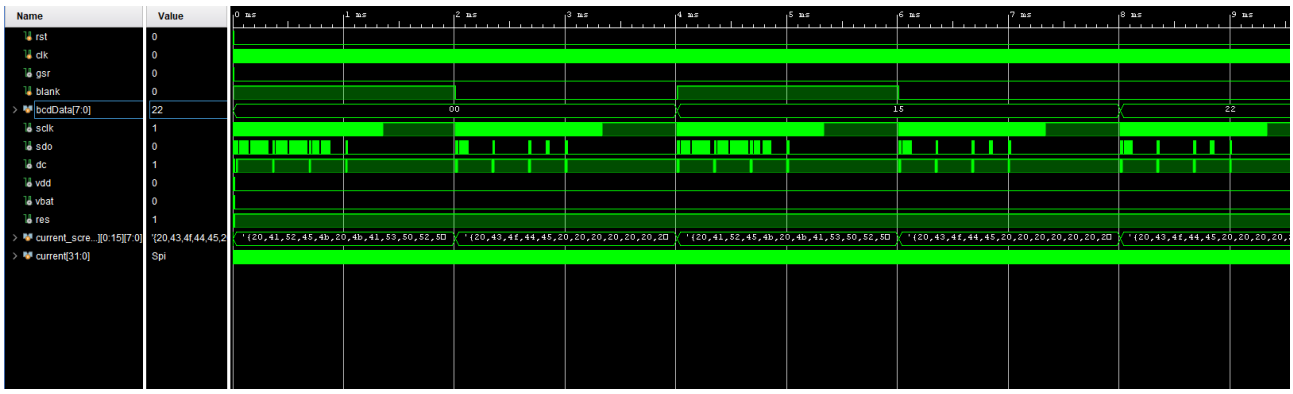
Test modułu DigitCompare polegał na podawaniu na wejście modułu różnych możliwych wartości szyfru i porównywaniu ich z prawdziwym szyfrem (tutaj 15, 30, 22). W momencie, gdy podana na wejście liczba (w postaci BCD - sygnał *bcd1* to cyfra dziesiątek, a *bcd0* to jedności) jest zgodna z aktualnie sprawdzaną wartością szyfru, sygnał *eq* powinien zmienić swój stan na wysoki. Rysunek 8 przedstawia przebieg testu. Wynik symulacji wskazuje na poprawne działanie modułu.



Rysunek 8: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły DigitCompare

6.1.8 Testy modułu OledDriver

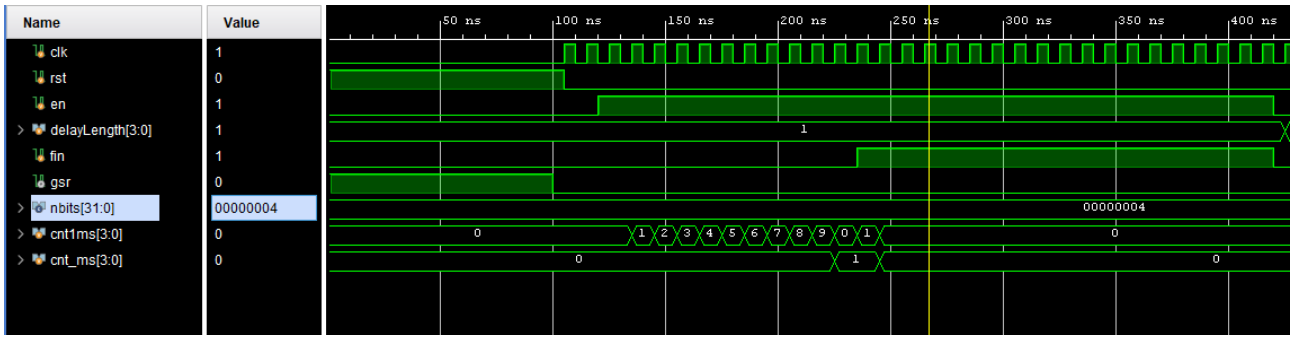
Moduł OledDriver reaguje na zmianę wartości sygnału *blank* oraz wektora *bcdData*. Test polegał więc na odczekaniu, aż zakończy się proces inicjalizacji wyświetlacza OLED, a następnie na zmianie jednego z tych dwóch sygnałów. W momencie zmiany przewidywanym zachowaniem jest nastąpienie transmisji SPI, co powinno być widoczne na linii *sdo*. Przebieg symulacji (rysunek 9) potwierdza przewidywania - moduł działa poprawnie.



Rysunek 9: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły OledDriver

6.1.9 Testy modułu Delay

Na potrzeby testu moduł Delay został skonfigurowany za pomocą parametrów w taki sposób, by odliczał jedną milisekundę jako dziesięć okresów zegara (licznik *cnt1ms)*. Po zakończeniu odliczania powinna wystąpić zmiana wartości sygnału *fin* na stan wysoki. Ilość milisekund, które powinny zostać policzone (licznik *cnt_ms*) określa wartość sygnału wektora wejściowego *delayLength*. Przebieg symulacji przedstawiony został na rysunku 10 - jest on zgodny z oczekiwaniami.



Rysunek 10: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły Delay

6.2 Testy manualne

7 Możliwe udoskonalenia

Projekt niepozbawiony jest niedoskonałości. Najważniejsze z nich to:

- sporadyczny zły odczyt kierunku obracania pokrętła - skutkuje to zwykle wprowadzeniem niepoprawnego kodu i koniecznością powtórzenia procedury
- przesuwanie się zawartości wyświetlacza OLED przy zmianie wyświetlanych danych

Literatura

[1] dr inż. Andrzej Skoczeń: materiały do przedmiotu Języki Opisu Sprzętu
http://www.fis.agh.edu.pl/ skoczen/hdl/

[2] Prezentacja Binary-to-BCD Converter dostępna na stronie:
http://www.tkt.cs.tut.fi/kurssit/1426/S12/Ex/ex4/Binary2BCD.pdf

[3] Opracowanie algorytmu konwersji wartości binarnych od BCD ze strony:
https://my.eng.utah.edu/ nmcdonal/Tutorials/BCDTutorial/BCDConversion.html