

**A G H**

## Języki Opisu Sprzętu

Projekt: Elektroniczny Sejf Hotelowy  
Dokumentacja

Arkadiusz Kasprzak

Jarosław Cierpich

Wydział Fizyki i Informatyki Stosowanej  
Informatyka Stosowana

5 grudnia 2019

# **Spis treści**

<b>1</b>	<b>Wstęp</b>	<b>3</b>
<b>2</b>	<b>Projekt</b>	<b>3</b>
2.1	Założenia projektowe . . . . .	3
2.2	Wymagana funkcjonalność . . . . .	3
<b>3</b>	<b>Dokumentacja użytkownika</b>	<b>5</b>
3.1	Elementy wykorzystywane do interakcji z użytkownikiem . . . . .	5
3.2	Korzystanie z urządzenia . . . . .	6
<b>4</b>	<b>Dokumentacja techniczna</b>	<b>9</b>
4.1	Warstwa Hardware . . . . .	9
4.2	Warstwa Software - architektura . . . . .	11
4.3	Warstwa Software - struktura projektu . . . . .	13
4.4	Warstwa Software - parametry i moduły projektu . . . . .	14
4.4.1	Parametry projektu . . . . .	14
4.4.2	Lista modułów projektu . . . . .	15
4.4.3	Moduł SafeTop . . . . .	16
4.4.4	Moduł Bcd2Decades . . . . .	17
4.4.5	Moduł RseDecoder . . . . .	18
4.4.6	Moduł MasterFsm . . . . .	20
<b>5</b>	<b>Analiza raportu syntezy</b>	<b>22</b>
5.1	Kodowanie stanów . . . . .	22
5.2	Użycie zasobów . . . . .	23
<b>6</b>	<b>Testy</b>	<b>24</b>
6.1	Moduły testujące . . . . .	24
6.1.1	Testy modułu SafeTop . . . . .	25
6.1.2	Testy modułu Bcd2Decades . . . . .	25
6.1.3	Testy modułu RseDecoder . . . . .	26
6.1.4	Testy modułu MasterFsm . . . . .	26
6.1.5	Testy modułu ClockDiv . . . . .	27
6.1.6	Testy modułu DebouncerButtons . . . . .	27
6.1.7	Testy modułu DigitCompare . . . . .	28
6.1.8	Testy modułu OledDriver . . . . .	28
6.1.9	Testy modułu Delay . . . . .	29
<b>7</b>	<b>Możliwe udoskonalenia</b>	<b>29</b>

# 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 obsługiwany 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:

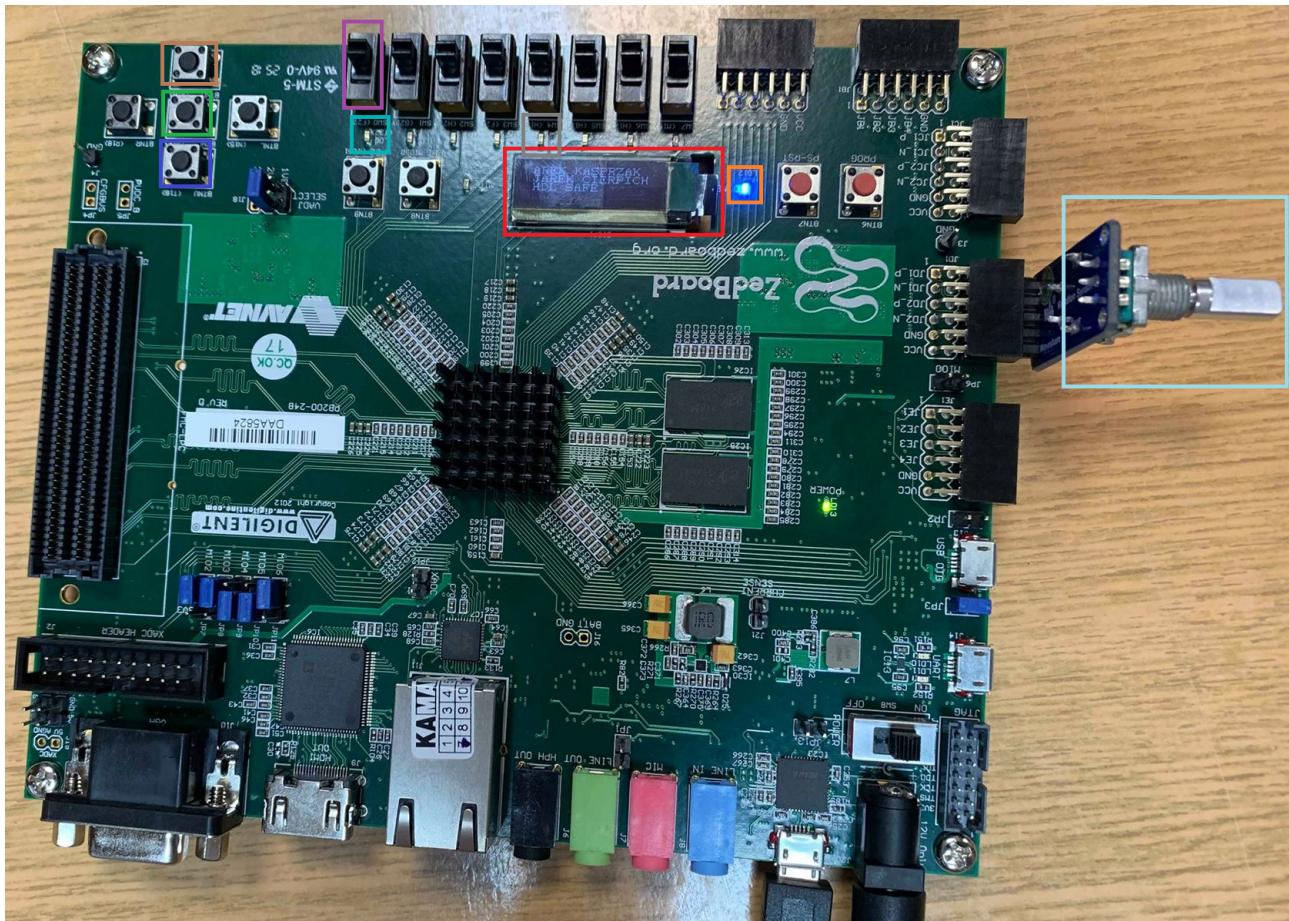
- możliwość wprowadzenia przez użytkownika poprawnego szyfru składającego się z trzech liczb z zakresu [0; 32] za pomocą pokrętła
- możliwość wprowadzenia przez użytkownika niepoprawnego szyfru, co automatycznie przerwać ma proces otwierania sejfu
- możliwość monitorowania przez użytkownika aktualnego stanu otwarcia sejfu - za pomocą diod LED
- możliwość monitorowania przez użytkownika aktualnie wprowadzonej 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

Rozdział przedstawia czynności, które należy wykonywać, aby poprawnie korzystać z urządzenia oraz oczekiwane działanie.

#### 3.1 Elementy wykorzystywane do interakcji z użytkownikiem



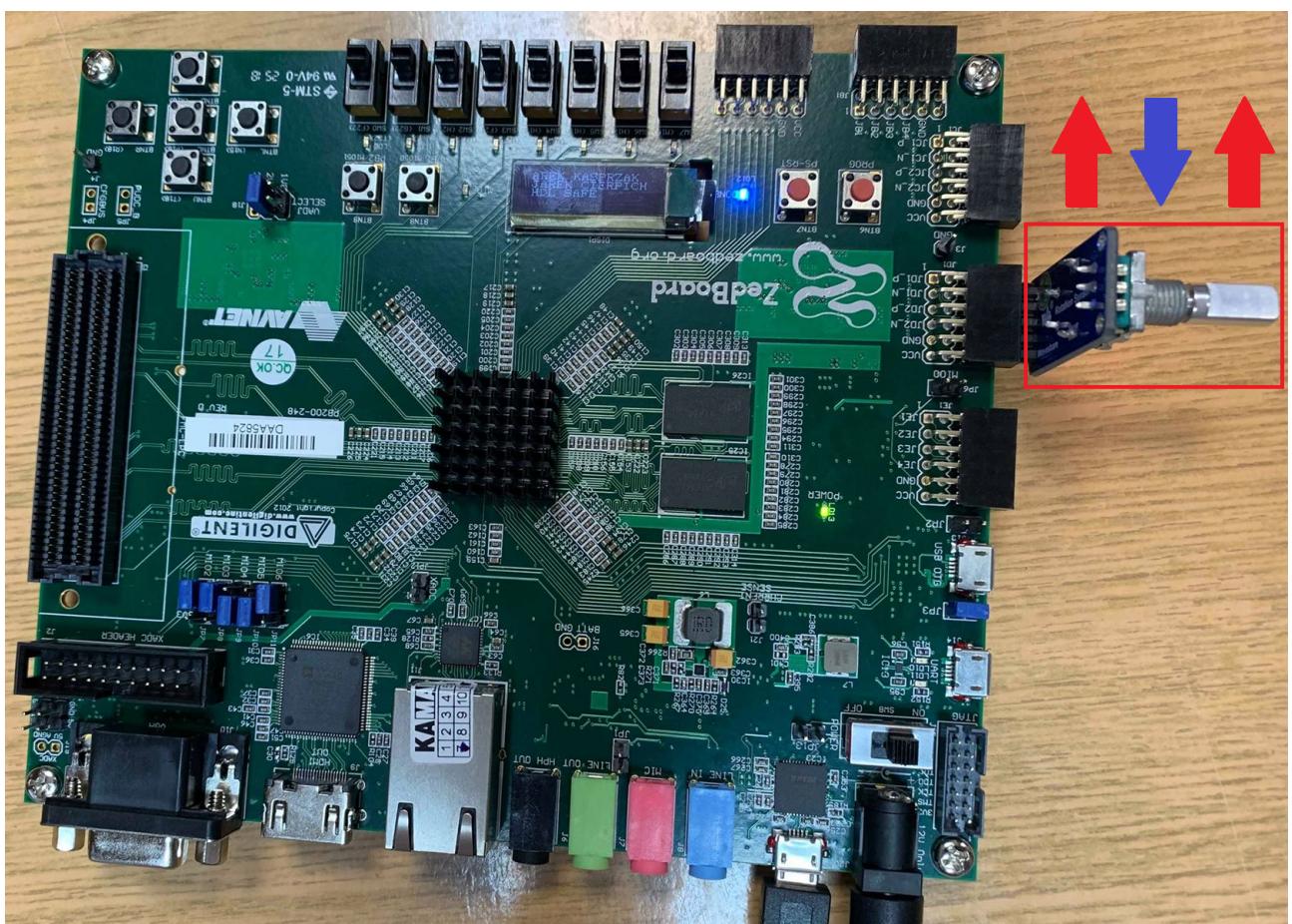
Rysunek 1: Elementy wykorzystywane do interakcji z użytkownikiem

- kolorem **czerwonym** oznaczony został ekran OLED
- kolorem **niebieskim** oznaczony został przycisk rozpoczynający otwieranie sejfu
- kolorem **zielonym** oznaczony został przycisk resetujący cały układ
- kolorem **brązowym** oznaczony został przycisk zamkający sejf
- kolorem **fioletowym** oznaczony został przełącznik odblokowujący funkcję zamykania sejfu
- kolorem **morskim** oznaczona została dioda sygnalizująca ruch rygla
- kolorem **pomarańczowym** oznaczona została dioda sygnalizująca gotowość układu do pracy, tj. wgranie oprogramowania na płytke
- kolorem ciemno szarym oznaczona została dioda sygnalizująca otwarcie sejfu
- kolorem **jasno-niebieskim** oznaczono pokrętło służące do wprowadzania kodu otwarcia

### **3.2 Korzystanie z urządzenia**

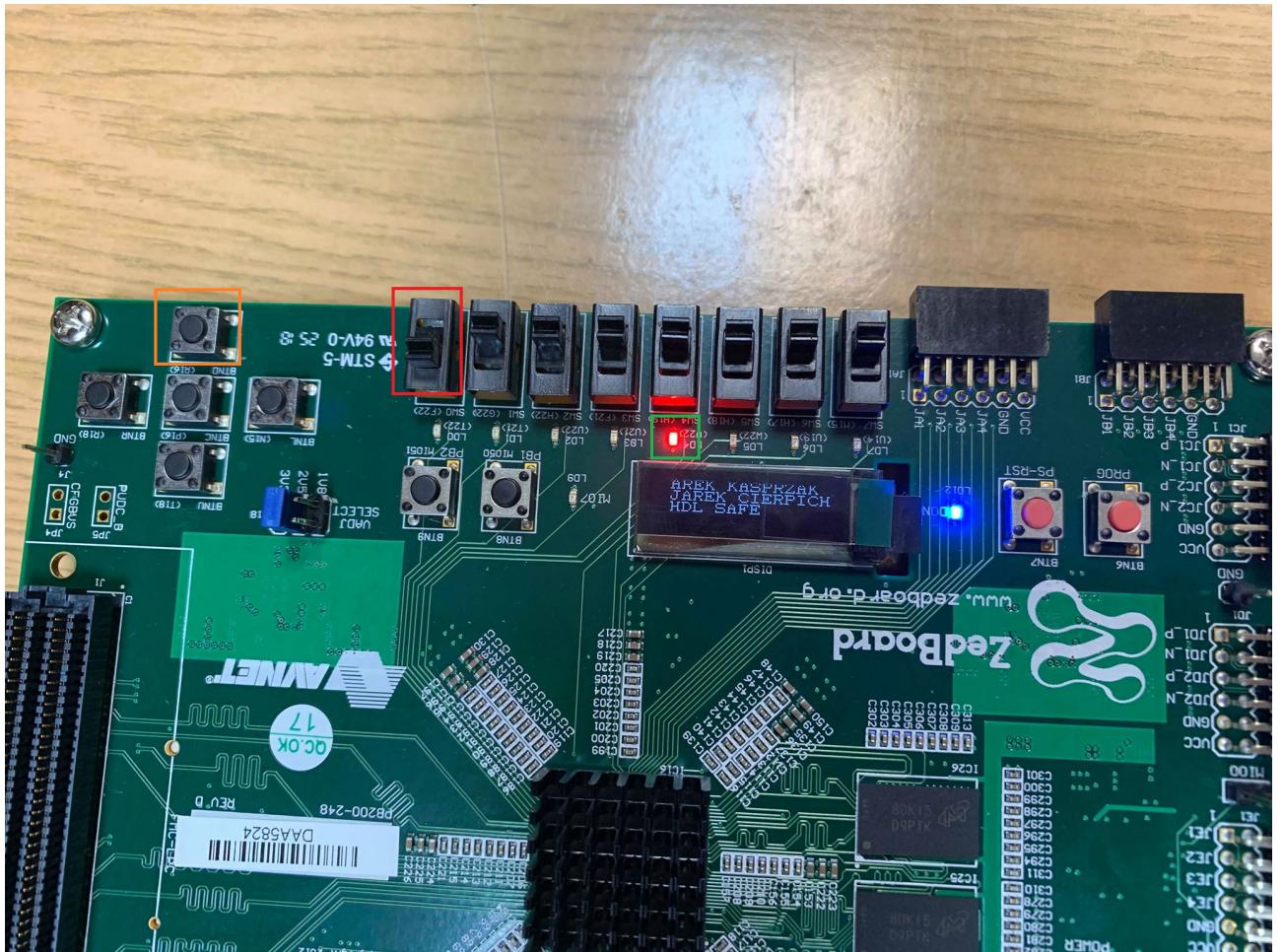
- Poprawne wgranie kodu na urządzenie można rozpoznać po zapalonej diodzie sygnalizującej wgranie oprogramowania na płytę oraz po napisie "Arek Kasprzak Jarek Cierpich HDL SA-FE" na ekranie OLED
- W przypadku niepoprawnego działania należy wykorzystać przycisk odpowiadający za reset
- W celu rozpoczęcia otwierania należy wcisnąć przycisk do tego przeznaczony. Powodzenie akcji będzie można rozpoznać po zmianie napisu na ekranie OLED na: "CODE: 00". Napis ten wskazuje nam na aktualnie wprowadzoną część kodu, kod składa się z trzech dwucyfrowych liczb z zakresu od 0 do 32.

- Kod powinien zostać wprowadzony za pomocą pokrętła w następujący sposób:
  - Pierwszą liczbę kodu należy wprowadzać przekręcając pokrętło zgodnie ze wskazówkami zegara. Wartość podawana na ekranie będzie się wtedy zwiększać.
  - Aby przejść do wybierania kolejnej części kodu należy zmienić kierunek kręcenia w momencie, gdy wybrana jest odpowiednia wartość (Jeżeli nasz kod składa się z liczby 9, 6 oraz 5, to zaczynamy kręcić w przeciwnym kierunku, gdy na ekranie znajduje się 9). Wartość podawana na ekranie będzie się wtedy zmniejszać.
  - Po osiągnięciu kolejnej części kodu (w podanym przykładzie 6) zaczynamy kręcić ponownie zgodnie ze wskazówkami zegara.
  - Przekręcamy pokrętło aż na ekranie będzie wyświetlana ostatnia część naszego kodu (w podanym przykładzie 5)
  - W przypadku niepowodzenia w wprowadzeniu kodu należy ponowić proces rozpoczęcia otwierania sejfu



Rysunek 2: Otwieranie sejfu

- w przypadku wprowadzenia odpowiedniego kodu na chwilę zapali się dioda sygnalizująca ruch rygla oraz zapali się dioda sygnalizująca otwarcie sejfu
- w celu ponownego zamknięcia sejfu należy przestawić przełącznik uruchamiający tą funkcję oraz kliknąć odpowiedni przycisk. Poprawne zamknięcie sejfu będzie widoczne poprzez zgaszenie się diody sygnalizującej otwarty sejf.



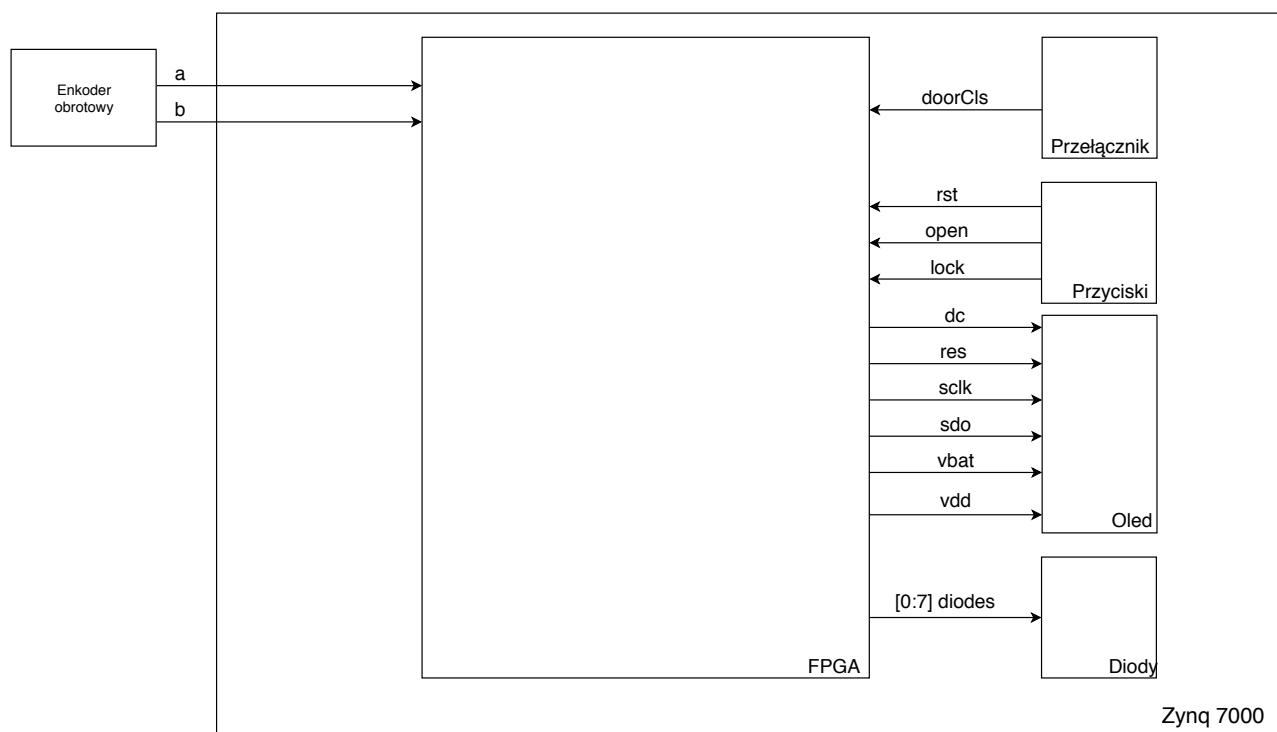
Rysunek 3: Zamknięcie sejfu

## 4 Dokumentacja techniczna

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

### 4.1 Warstwa Hardware

W tej sekcji zostanie przedstawiona architektura projektu z punktu widzenia **sprzętowego**



Rysunek 4: Przedstawia elementy sprzętowe użyte w projekcie oraz sygnały wymieniane między nimi.

Elementy użyte w realizacji projektu:

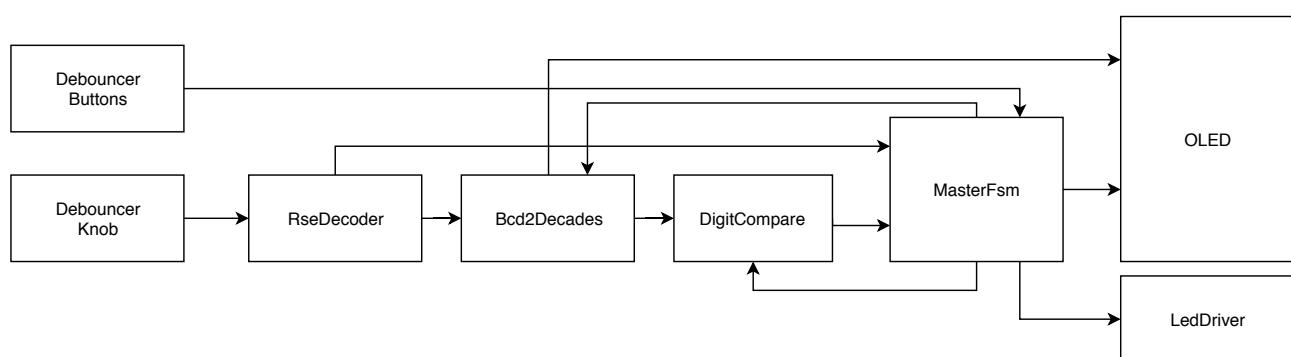
- **Enkoder obrotowy**, który za pomocą wysłanych sygnałów dostarcza informacji dotyczących obrotu oraz jego kierunku. Nie jest on częścią płytki Zynq 7000. Został podłączony za pomocą portów JD1\_N oraz JD2\_P
- **FPGA** czyli programowalny układ logiczny. Steruje on całym układem. Element odpowiedzialny za logikę układu
- **Przełącznik SW0** odpowiedzialny za dostarczanie sygnału odblokowania funkcji zamykania sejfu
- **Przyciski BTNC, BTNU oraz BTND** odpowiedzialne za dostarczanie sygnałów zamykania, otwierania oraz resetowania układu
- **Ekran OLED** który jest częścią płytki Zynq 7000. Wymaga on specjalnej procedury inicjalizującej. Informacje wysyłane są za pomocą protokołu SPI
- **Diody**, służą do sygnalizowania informacji na temat ruchu ryglą oraz stanu sejfu (zamknięty/otwarty)

## 4.2 Warstwa Software - architektura

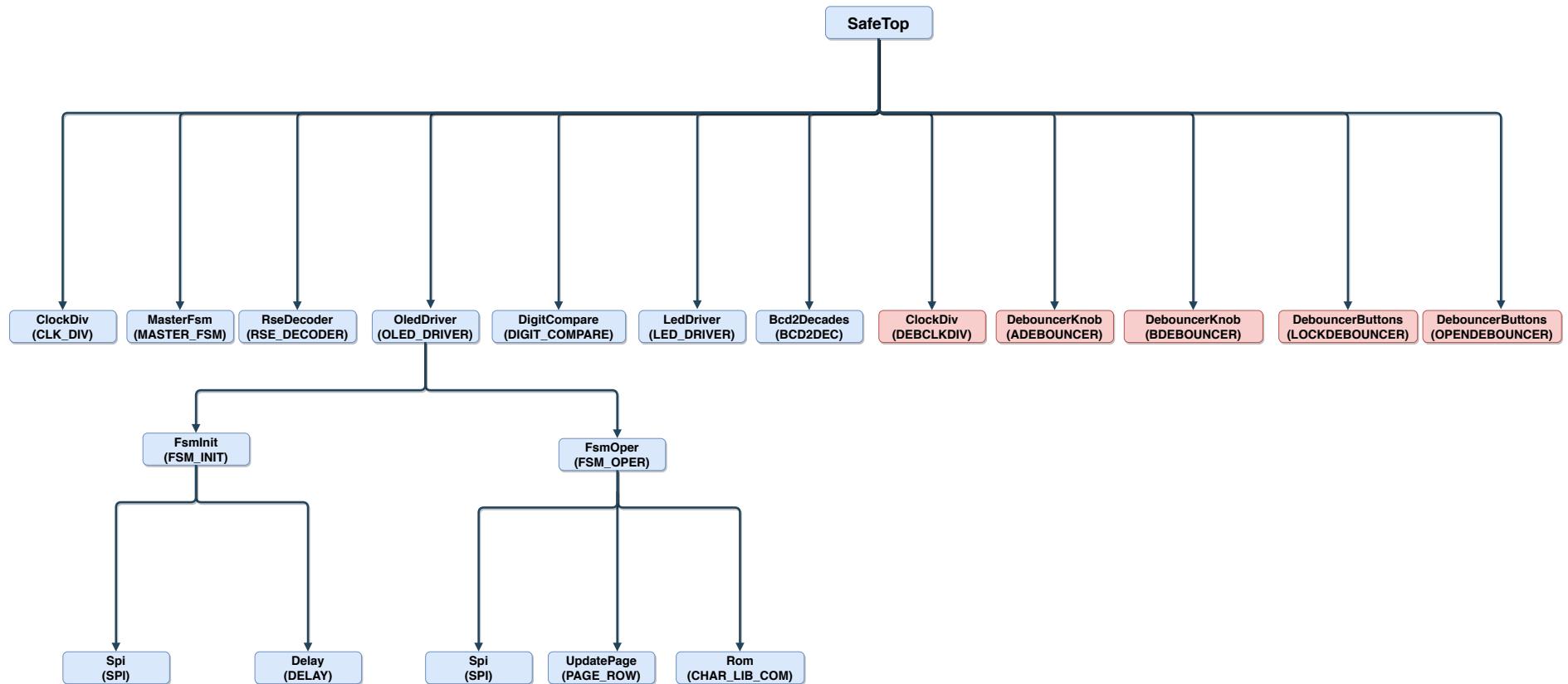
W tej sekcji omówiona zostanie architektura projektu - zarówno ze względu na przepływ danych między komponentami, jak i na ich hierarchię w projekcie.

Rysunek 6 przedstawia hierarchię instancji modułów w projekcie, z wyróżnieniem instancji modułów odpowiedzialnych za tłumienie drgań na przyciskach i pokrętłe - instancje te nie są generowane w czasie symulacji behawioralnej.

Rysunek 5 przedstawia strukturę zastosowanych w projekcie modułów wraz z przepływem danych między nimi.



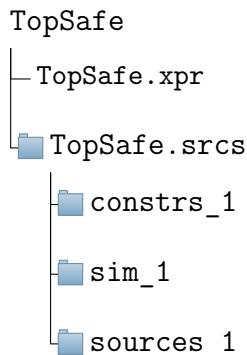
Rysunek 5: Struktura modułów oraz przepływ informacji



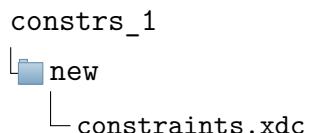
Rysunek 6: Hierarchia instancji modułów zastosowana w projekcie. Nazwy w nawiasach to nazwy instancji, przed nawiasami - nazwy modułów. Na czerwono zaznaczono instancje, które nie są generowane podczas symulacji behawioralnej (wygaszanie drgań)

## 4.3 Warstwa Software - struktura projektu

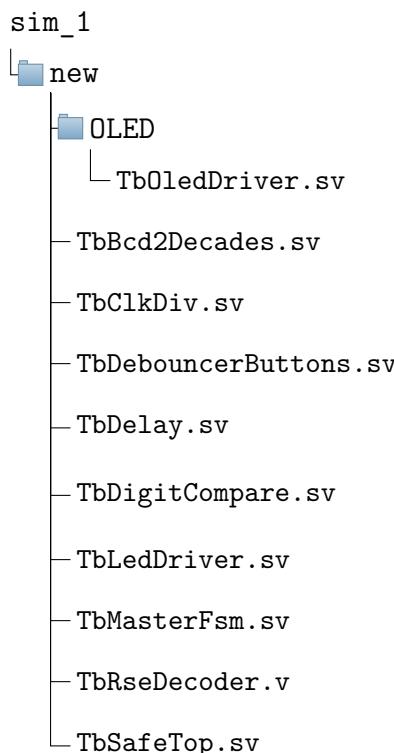
Wysokopoziomowa struktura katalogów i plików przedstawiona została poniżej:



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. Test-bench) i jego strukturę przedstawiono poniżej:



Struktura katalogu **sources\_1** zawierającego kod źródłowy projektu:

```
sources_1
└── new
    ├── Bcd2Decades.sv
    ├── ClockDiv.sv
    ├── DebouncerButtons.sv
    ├── DebouncerKnob.v
    ├── DigitCompare.sv
    ├── LedDriver.sv
    ├── MasterFsm.sv
    ├── safeTop.sv
    └── RseDecoder.sv
└── OLED
    ├── delay.sv
    ├── fsm_init.sv
    ├── fsm_oper.sv
    ├── OledDriver.sv
    ├── pixel_SSD1306.dat
    ├── rom.v
    ├── screens.vh
    ├── spi.sv
    └── update_page.sv
```

## 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 pokrętłe
- **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 najwyższej 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 przedstawia implementację opisywanego rozwiązania. Schemat blokowy przedstawiony na rysunku 7 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)
        );

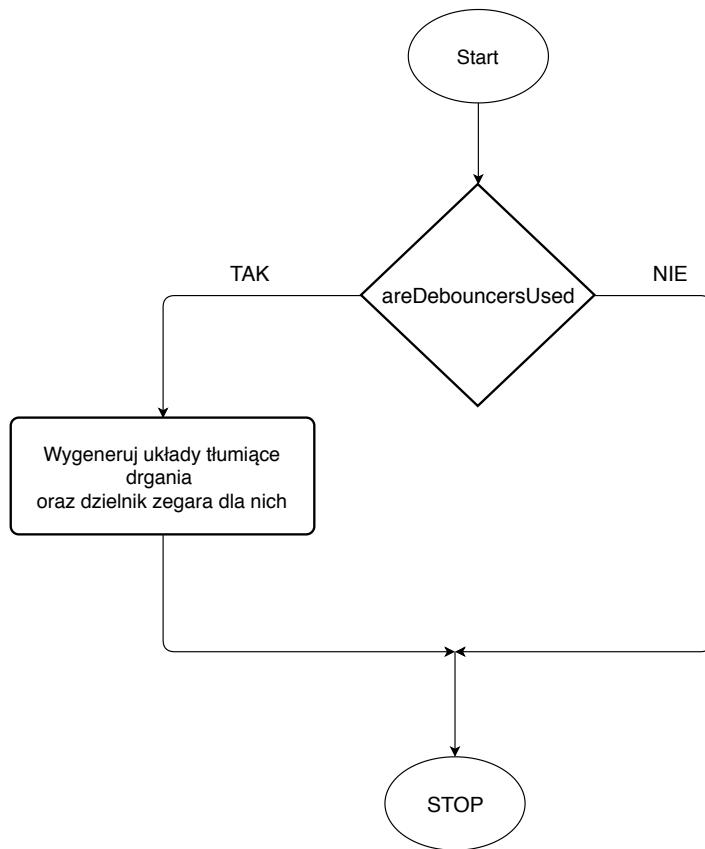
        // a signal debouncer
        DebouncerKnob #(.N(3)) ADEBOUNCER (
            .clk(debSlowClk), .rst(rst), .in(a), .out(aKnob)
        );

        // 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 7: 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 przedstawiona 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

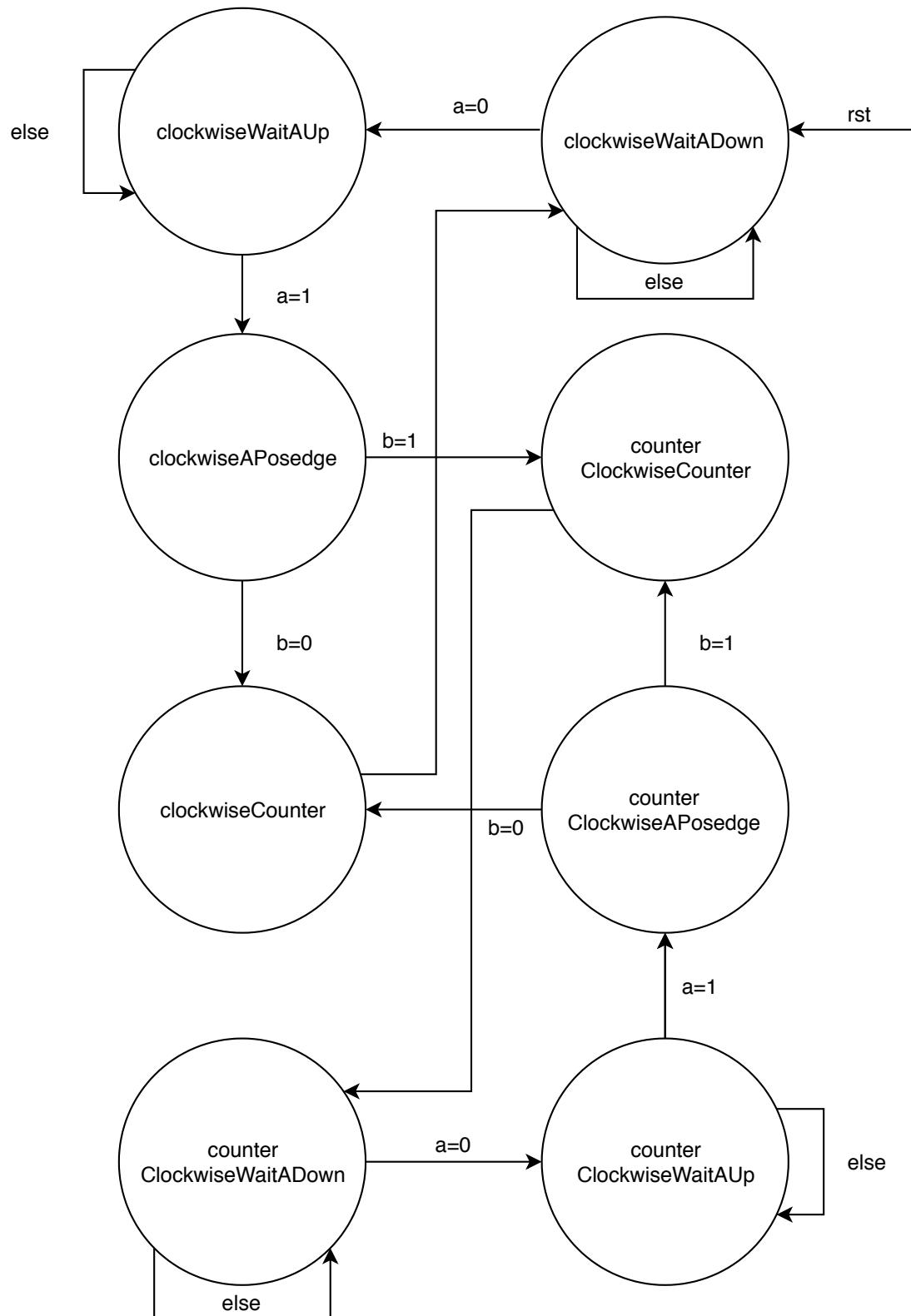
```

Tabela 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

Moduł **RseDecoder** odpowiada za odpowiednie przetwarzanie sygnału dostarczonego przez enkoder obrotowy. Sygnał dostarczany jest wcześniej odpowiednio debounceowany w celu eliminacji drgań. Logika modułu opiera się na odpowiedniej maszynie stanów przedstawionej na Rysunku 8. Moduł sygnalizuje ruch ryglą oraz jego kierunek osiągając stany *clockwiseCounter* oraz *counterClockwiseCounter*. Zmiana kierunku jest sygnalizowana poprzez przejścia z *clockwisePosedge* na *counterClockwiseCounter* oraz *counterClockwisePosedge* na *clockwiseCounter*.



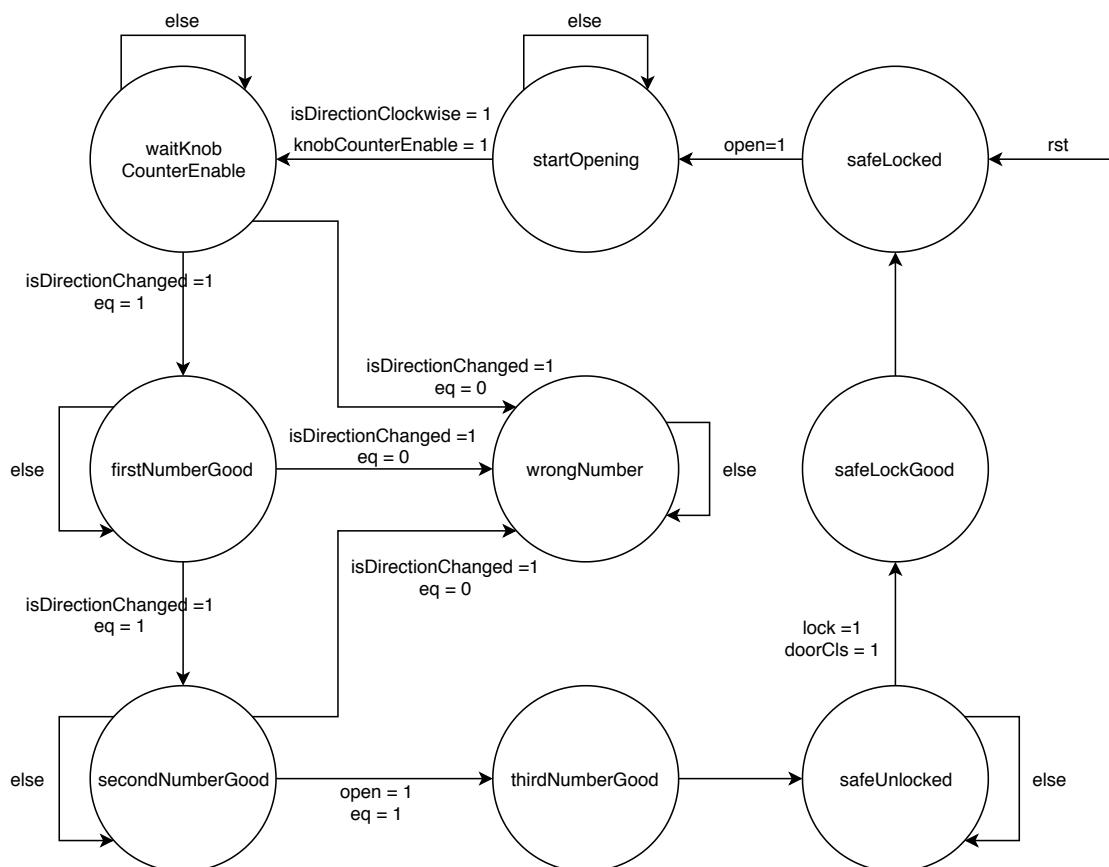
Rysunek 8: Maszyna stanów RseDecoder

Wyjściami modułu są następujące sygnały:

- **knobCounterEnable** - sygnał informujący o odczycie obrotu
- **isDirectionClockwise** - sygnał informujący czy odczytany obrót jest zgodny z kierunkiem wskazówek zegara (1) czy też nie (0)
- **isDirectionChanged** - sygnał informujący o zmianie kierunku obracania względem wcześniejszej zarejestrowanego kierunku

#### 4.4.6 Moduł MasterFsm

Moduł **MasterFsm** odpowiada za główną logikę urządzenia. Decyduje on o otwarciu sejfu. Głównym jego członem jest maszyna stanów, która jest przedstawiona na Rysunku 9. Znajdują się w niej przejścia między stanami sprawdzania poprawności kolejnych części szyfru, otwarcia/zamknięcia sejfu. Poprawność szyfru jest sprawdzana w momencie zmiany kierunku sygnalizowanej przez **RseDecoder**, sprawdzenia dokonuje moduł **DigitCompare** zwracając informacje do **MasterFsm**. Przejścia w module **MasterFsm** zależą również od przycisków (sygnały open, lock) oraz od przełącznika (sygnał doorCls).



Rysunek 9: Maszyna stanów MasterFsm

Wyjściami modułu są sygnały:

- **enableCounter** - uruchomienie licznika
- **clearCounter** - wyzerowanie licznika
- **blank** - wyczyszczenie wyświetlacza
- **triggerLock** - uruchomienie ryglą
- **isLockBeingOpened** - kierunek ruchu ryglą (otwierający/zamykający)
- **numberSelector** - wskazuje która liczba szyfru jest aktualnie obsługiwana

## 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**

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

Stan (State)	Nowe kodowanie	Poprzednie kodowanie
safeLocked	000000001	00
startOpening	000000010	0001
waitKnobCounterEnable	000000100	00010
firstNumberGood	000001000	00011
secondNumberGood	000010000	000100
thirdNumberGood	000100000	000101
safeUnlocked	001000000	000111
safeLockGood	010000000	0001000
wrongNumber	100000000	000110

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

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

Stan (State)	Nowe kodowanie	Poprzednie kodowanie
idle	000	00
ClearDC	001	0001
SendCmd	010	00010
Transition1	011	000100
Transition2	100	000101
Transition5	101	000110
SetDC	110	000111

## 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.

Tabela 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.

## 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;

    // wejścia
    reg clk, rst;
    reg in1;

    // wyjścia
    reg [3:0] out1;

    // ...

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

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

    // generacja sygnałów wejściowych
    // 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

// ...

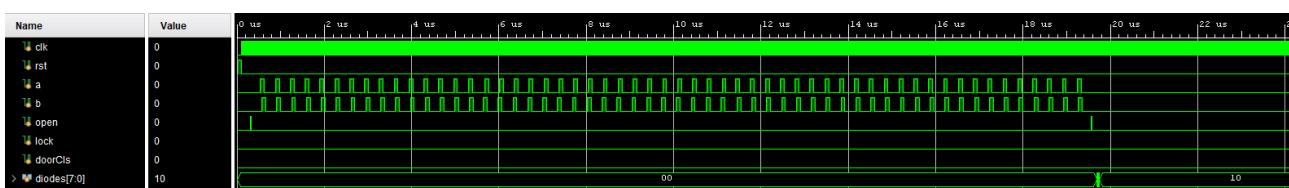
endmodule

```

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

### 6.1.1 Testy modułu SafeTop

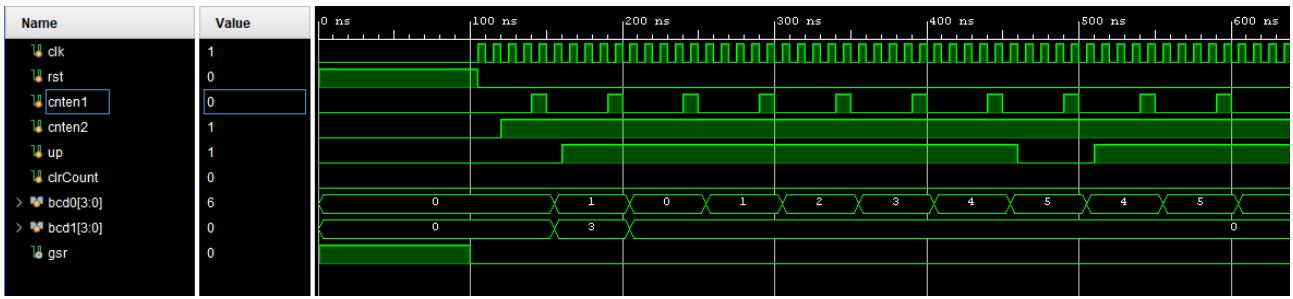
Test modułu SafeTop zakłada wyłączenie generacji debouncerów w module. Rysunek 10 przedstawia wartości sygnałów uzyskane podczas testu. Podczas testu wygenerowana została odpowiednia kombinacja sygnałów wejściowych  $a$  i  $b$  (sygnały pochodzące z pokrętła), by wprowadzić poprawny szyfr sejfu. Po 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 oczekiwany.



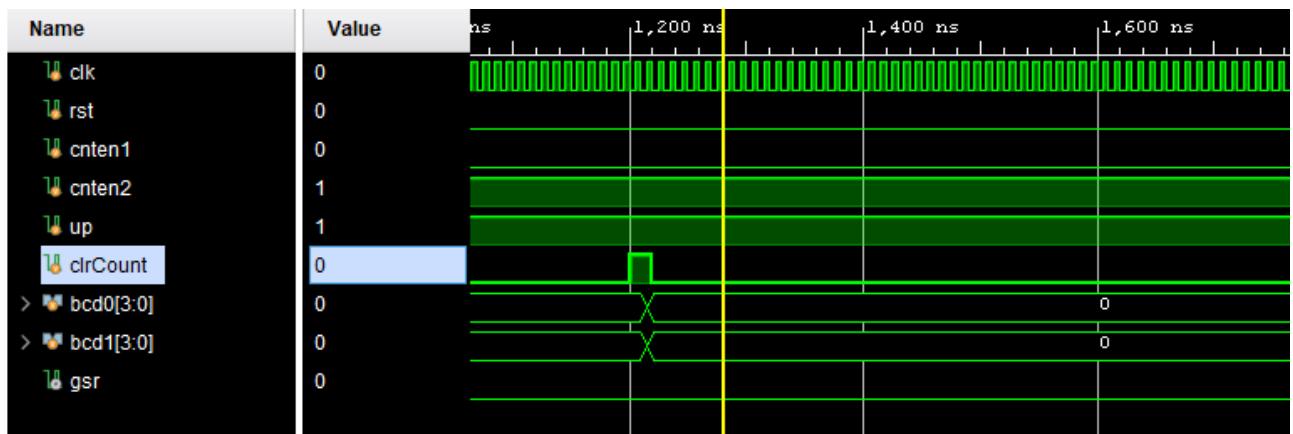
Rysunek 10: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły SafeTop

### 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 11 przedstawia pomyślnie przeprowadzony test opisanej funkcjonalności. Możliwe jest również wyzerowanie licznika za pomocą sygnału *clrCount* - dokładnie takie działanie można zaobserwować na rysunku 12.



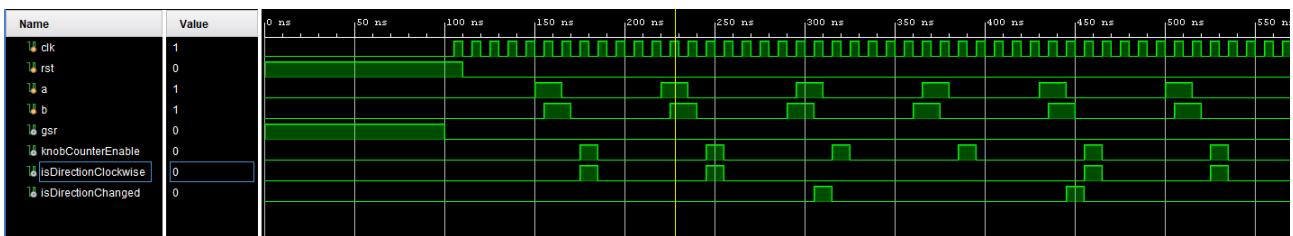
Rysunek 11: Wartości sygnałów wejściowych i wyjściowych podczas testu modułu Bcd2Decades



Rysunek 12: Reakcja modułu Bcd2Decades na stan wysoki sygnału clrCount.

### 6.1.3 Testy modułu RseDecoder

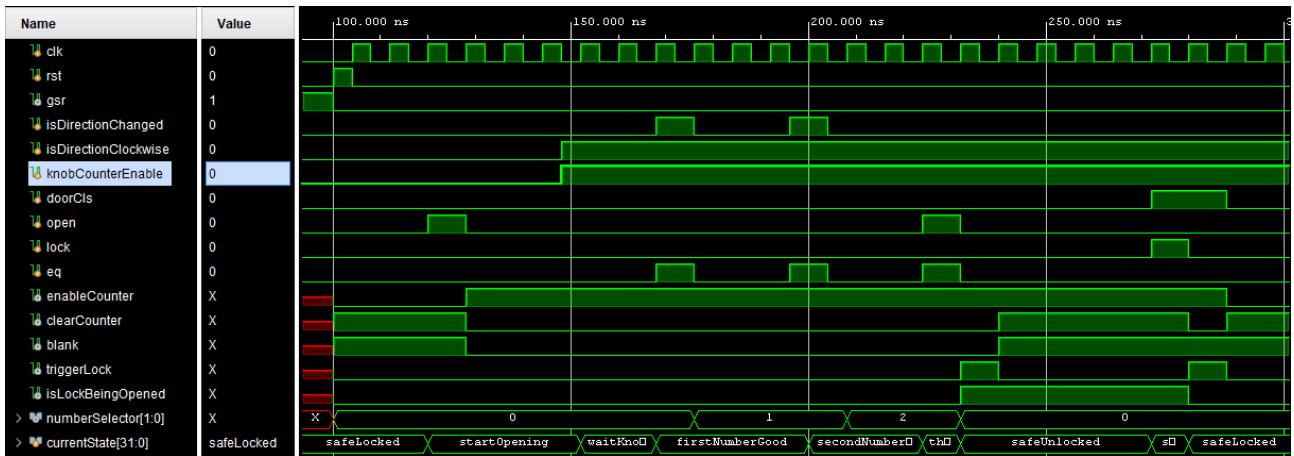
Test moduły **RseDecoder** polega na wygenerowaniu sygnałów a i b o odpowiedniej długości oraz kolejności. Są one generowane w taki sposób, aby sygnalizować dwa razy ruch zgodny z kierunkiem wskazówek zegara, dwa razy ruch przeciwny do kierunku wskazówek zegara oraz dwa razy ruch zgodny z kierunkiem wskazówek zegara. Moduł testowy ukazany na Rysunku 13 wykazuje, że **RseDecoder** poprawnie generuje sygnały wyjściowe dla zadanych sygnałów wejściowych.



Rysunek 13: Moduł testowy dla RseDecoder

### 6.1.4 Testy modułu MasterFsm

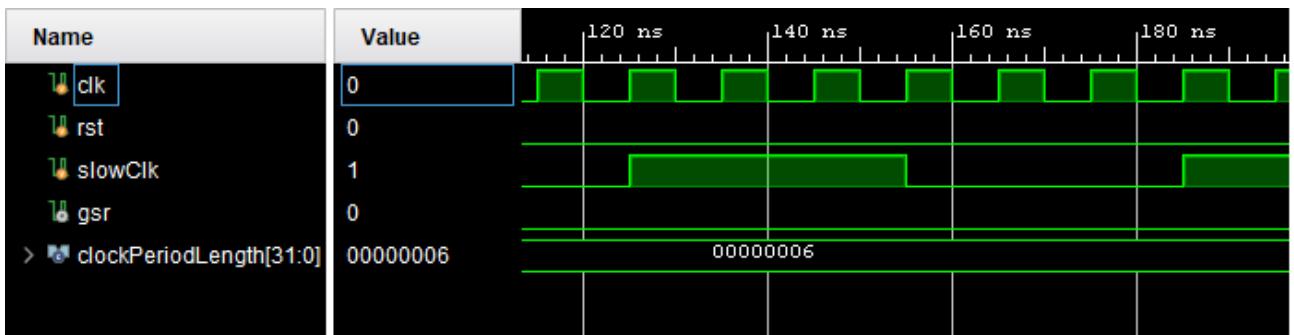
Test modułu **MasterFsm** polega na zaaplikowaniu odpowiednich sygnałów wejściowych, tak, aby maszyna stanów przeszła przez cały proces dla odpowiedniego otwierania sejfu. Zgodnie z Rysunkiem 14 moduł poprawnie reaguje na wszystkie sygnały na co wskazują m.in. stany przyjmowane przez moduł.



Rysunek 14: Moduł testowy dla MasterFsm

### 6.1.5 Testy modułu ClockDiv

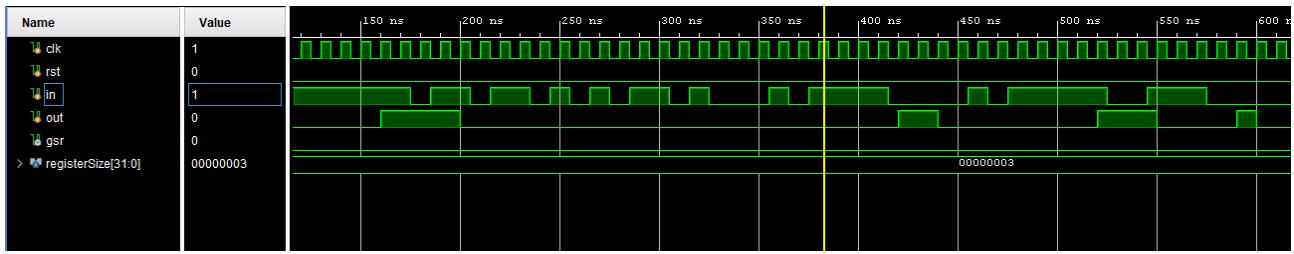
Test modułu ClockDiv jest bardzo prosty. Jego rezultaty przedstawione zostały na rysunku 15. Na wejście modułu podawany jest zegar. Długość okresu wynikowego zegara (*slowClk*) ustalona została za pomocą parametru na sześć okresów zegara wejściowego. Test wskazuje więc, że działanie modułu jest zgodne z oczekiwaniami - okres sygnału *slowClk* jest sześć razy dłuższy od okresu sygnału *clk*.



Rysunek 15: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły ClockDiv

### 6.1.6 Testy modułu DebouncerButtons

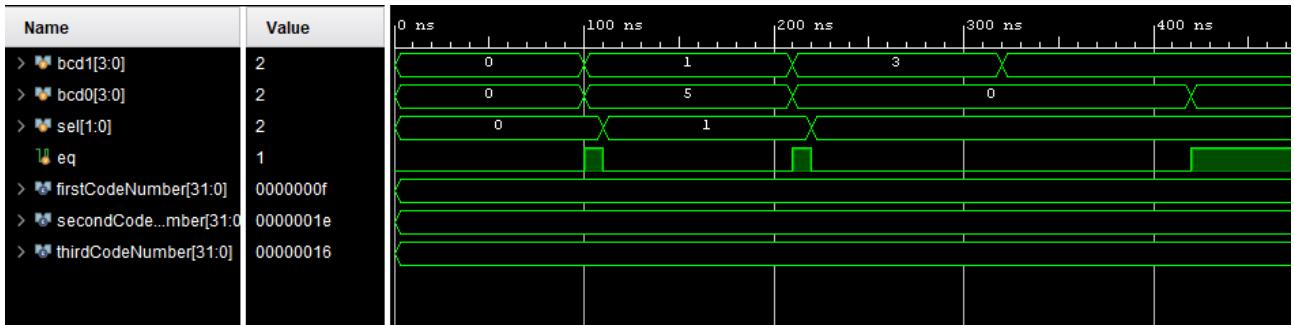
Rysunek 16 przedstawia przebieg symulacji modułu DebouncerButtons. Moduł ten próbuje na każdym rosnącym zboczu zegara wartość na linii *in*. Jeśli w ciągu 3 kolejnych próbek linia utrzymuje stan wysoki, na wyjściu *out* generowany jest sygnał o wartości 1. Obserwacja przebiegów pozwala stwierdzić, że moduł zachowuje się zgodnie z oczekiwaniami.



Rysunek 16: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły DebouncerButtons

### 6.1.7 Testy modułu DigitCompare

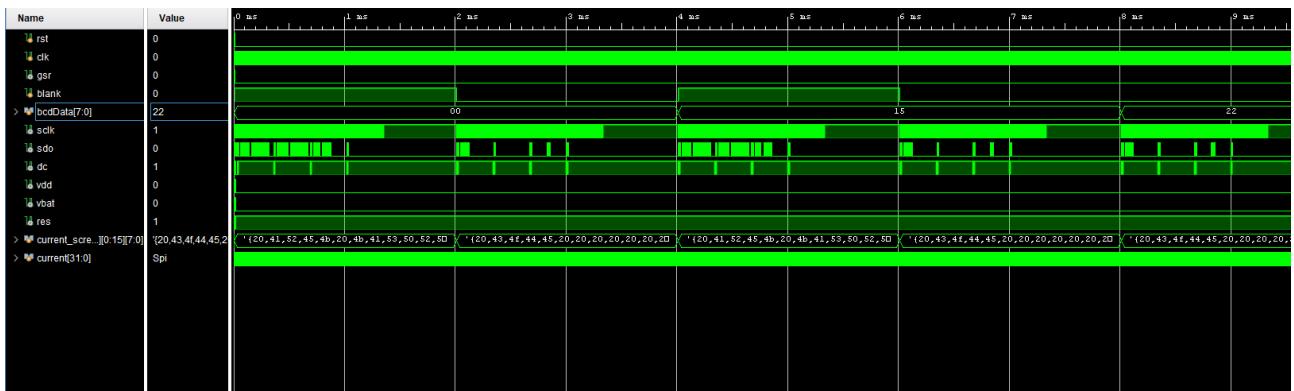
Test modułu DigitCompare polegał na podawaniu na wejście modułu różnych możliwych wartości szyfra i porównywaniu ich z prawdziwym szyfrem (tutaj 15, 30, 22). W momencie, gdy podana na wejście liczba (w postaci BCD - sygnał *bcd1* to cyfra dziesiątek, a *bcd0* to jedności) jest zgodna z aktualnie sprawdzaną wartością szyfra, sygnał *eq* powinien zmienić swój stan na wysoki. Rysunek 17 przedstawia przebieg testu. Wynik symulacji wskazuje na poprawne działanie modułu.



Rysunek 17: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły DigitCompare

### 6.1.8 Testy modułu OledDriver

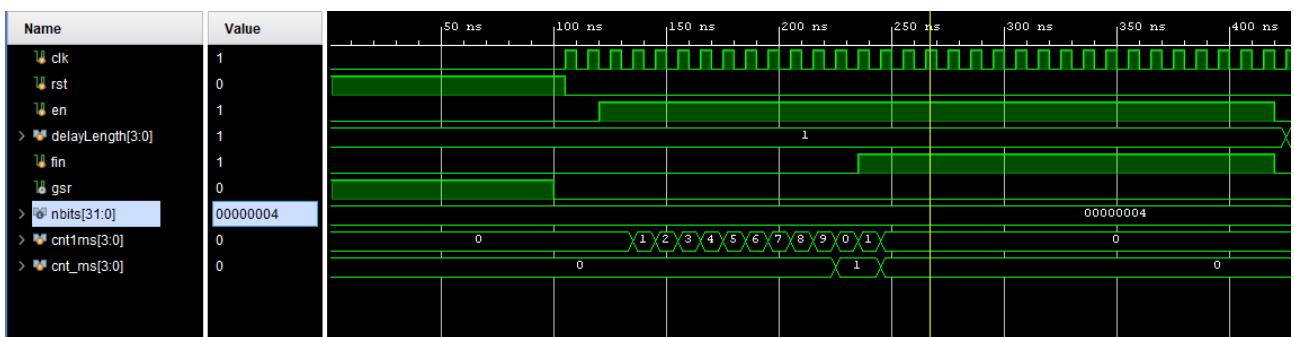
Moduł OledDriver reaguje na zmianę wartości sygnału *blank* oraz wektora *bcdData*. Test polegał więc na oczekaniu, aż zakończy się proces inicjalizacji wyświetlacza OLED, a następnie na zmianie jednego z tych dwóch sygnałów. W momencie zmiany przewidywanym zachowaniem jest nastąpienie transmisji SPI, co powinno być widoczne na linii *sdo*. Przebieg symulacji (rysunek 18) potwierdza przewidywania - moduł działa poprawnie.



Rysunek 18: 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 19 - jest on zgodny z oczekiwaniami.



Rysunek 19: Wartości sygnałów wejściowych i wyjściowych podczas testu moduły Delay

## 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/](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>