

# Czytnik kart SD w języku VHDL na Spartanie 3E

Układy cyfrowe i systemy wbudowane 2  
projekt  
semestr letni 2012/2013  
prowadzący  
dr inż. Jarosław Sugier

Autorzy:  
Rafał Zajfert 184743  
Artur Zochniak 184725

# Spis treści

[Cel i zakres problemu](#)

[Teoria problemu](#)

[Karta SD \(ang. Secure Digital\)](#)

[Interfejs SPI \(ang. Serial Peripheral Interface\)](#)

[Opis sprzętu](#)

[Opis projektu](#)

[Hierarchia plików](#)

[Plik schemat.sch](#)

[Moduł sdcard](#)

[Implementacja modułu SDCard](#)

[Definicja jednostki](#)

[Implementacja maszyny stanów](#)

[Proces odpowiedzialny za zmianę stanu](#)

[Pozostałe procesy odpowiedzialne za wyprowadzanie sygnałów](#)

[Implementacja wysyłania komendy CMD0](#)

[Implementacja wysyłania komendy CMD1](#)

[Symulacja](#)

[Wyniki symulacji](#)

[Inicjalizacja karty komendą CMD0](#)

[Wysłanie komendy CMD1](#)

[Wysyłanie komendy CMD17 - odczyt danych z karty](#)

[Odbieranie kolejnych bajtów w bloku](#)

[Informacje o implementacji i osiągi czasowe](#)

[Użycie zasobów układu FPGA](#)

[Instrukcja użytkownika](#)

[Jak podłączyć](#)

[Budowanie pliku binarnego](#)

[Obsługa urządzenia](#)

[Funkcje klawiszy](#)

[Podsumowanie](#)

# Teoria

## Cel i zakres problemu

Celem projektu było zaimplementowanie programu umożliwiającego odczyt danych z karty pamięci SD. Celem było odczytywanie danych bajt po bajcie, co m. in. umożliwiało poznanie kluczowych elementów systemu plików FAT.

Obsługa karty SD protokołem SPI ([1], str. 153) wiąże się z rozwiązaniem następujących problemów:

- dobranie pracy sygnału zegarowego, aby nie stracić i/lub nie przekłamać żadnych wartości,
- inicjalizacja karty, na co składa się:
  - doprowadzenie zasilania do karty,  
podanie stanu niskiego na linię CS,
  - odczekanie odpowiedniej ilości taktów, aby karta wyzerowała wartości wewnętrznych buforów,  
około 80 taktów pracy karty
  - przesłanie komendy wykonującej programowy reset CMD0, równoległe z niskim potencjałem na linii wyboru urządzenia, co włączy komunikację protokołem SPI, po wejściu w ten tryb nie ma możliwości powrotu do domyślnego sposobu komunikacji z kartą innego niż wyłączenie zasilania,

[illegible]

- odebranie odpowiedzi od karty o jej gotowości, odpowiedź 8-bitowa klasy R1: 0x01=b00000001,
- wysłanie komendy CMD41

[illegible]

- odebranie odpowiedzi klasy R1 (0x01=b00000001)
- wysłanie żądania odczytania bloku danych komendą CMD17

której sposób konstrukcji przebiega następująco:

x"FF" & x"51" & address & x"FF", gdzie address to 32-bitowy adres 512-bajtowego bloku do odczytania.

- odebranie 512 bajtów x 8 bitów=4096 bitów,
- wyświetlenie każdego bajtu na monitorze złączem VGA korzystając z dostarczonego przez prowadzącego modułu.

# Teoria problemu

## Karta SD (ang. Secure Digital)

**SD** jest jednym ze standardów kart pamięci opracowany przez firmy Panasonic, SanDisk i Toshiba. Pierwsze nośniki danych tego typu pojawiły się pod koniec 2000 roku. Karty SD charakteryzują się niewielkimi wymiarami (24 × 32 × 2,1 mm) i masą (ok. 2 gramów).

Produkowane obecnie karty SD charakteryzują się dużą prędkością zapisywania i odczytywania danych, a ich pojemność sięga 4 GB. Są używane głównie jako pamięci masowe w cyfrowych aparatach fotograficznych, kamerach, odtwarzaczach MP3, komputerach itp.

Dostępne na rynku karty SD obsługują dwa standardy komunikacyjne: dedykowany SDBus oraz SPI. Interfejs natywny (SDBus) oferuje duże możliwości i dużą szybkość pracy, kosztem wzrostu stopnia skomplikowania obsługi interfejsu. Z tego powodu w kartach SD dostępna jest również komunikacja za pomocą o wiele prostszej w obsłudze magistrali SPI, lecz z nieco okrojonymi możliwościami. Jeśli tylko aplikacja nie wymaga wyjątkowo dużej szybkości przesyłania danych, to nie ma jakiegokolwiek sensu implementacja obsługi interfejsu SDBus, wystarczy praca z magistralą SPI. Sposób podłączenia karty SD przez magistralę SPI do mikrokontrolera został przedstawiony na rys.2. Podobny układ został wykorzystany na płytce Spartan-3E Starter Kit.

## Interfejs SPI (ang. Serial Peripheral Interface)

**SPI** - szeregowy interfejs urządzeń peryferyjnych. Jeden z najczęściej używanych interfejsów komunikacyjnych pomiędzy systemami mikroprocesorowymi a układami peryferyjnymi takimi jak: przetworniki ADC/DAC, układy RTC, pamięci EEPROM, pamięci flash, karty MMC/SD/ itp.

Komunikacja odbywa się synchronicznie za pomocą 3 linii:

- MOSI (ang. Master Output Slave Input) - dane dla układu peryferyjnego
- MISO (ang. Master Input Slave Output) - dane z układu peryferyjnego
- SCLK (ang. Serial CLock) - sygnał zegarowy (taktujący)

Do aktywacji wybranego układu służy dodatkowo linia CS (ang. chip select - wybór układu) lub adresacja układów.

W celu uruchomienia obsługi odczytu danych z karty SD wystarczy znajomość budowy komend oraz ich kilku typów, jakie obsługują karty SD. Każda komenda, która ma być wysłana do karty SD, składa się z sześciu bajtów, Struktura została przedstawiona w tab.1.

0	1	bit 5...bit 0	bit 31...bit 0	bit 6...bit 0	1
---	---	---------------	----------------	---------------	---

start bit	host	command	argument	CRC	end bit
-----------	------	---------	----------	-----	---------

tab.1 Struktura komendy

Pierwszym bajtem jest zawsze kod komendy, kolejne cztery bajty to jej argument. Na końcu jest przesyłany bajt sumy kontrolnej CRC. O ile w trybie pracy z interfejsem SDBus CRC jest sprawdzane, to przy komunikacji za pomocą magistrali SPI, suma kontrolna jest przez kartę ignorowana. Tylko w trakcie przesyłania komendy CMD0, przełączającej tryb pracy z SDBus na SPI jest wymagany bajt CRC. Nie trzeba go w żaden sposób obliczać, ponieważ jest to stała wartość i wynosi 0x95. W tab.2 umieszczono komendy obsługiwane w trybie pracy z magistralą SPI wraz z opisem argumentów. Oprócz standardowych komend CMD karty SD mogą wykorzystywać jeszcze tak zwane komendy aplikacji (ACMD). Wysłanie komendy aplikacji wymaga uprzedniego wysłania komendy CMD55, która informuje kartę SD, że następna będzie komenda ACMD.

Komenda	Opis
CMD0	Zerowanie karty, przełączenie w tryb SPI
CMD1	Aktywacja procesu inicjalizacji karty
CMD17	Odczytanie jednego bloku pamięci o długości 512 bitów

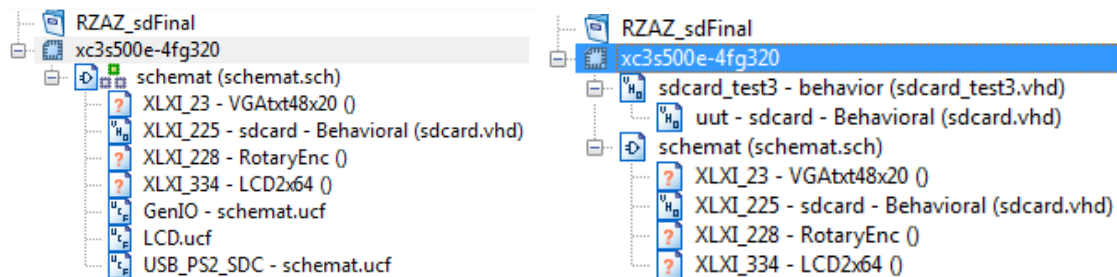
tab.2 Wybrane komendy CMD obsługiwane przez kartę w trybie SPI

## Opis sprzętu

Projekt został zrealizowany na układzie Spartan E3 z podłączonym doń dedykowanym modulem ze slotem na kartę SD oraz wyprowadzonym wyjściem VGA w celu wyświetlania informacji na monitorze CRT.

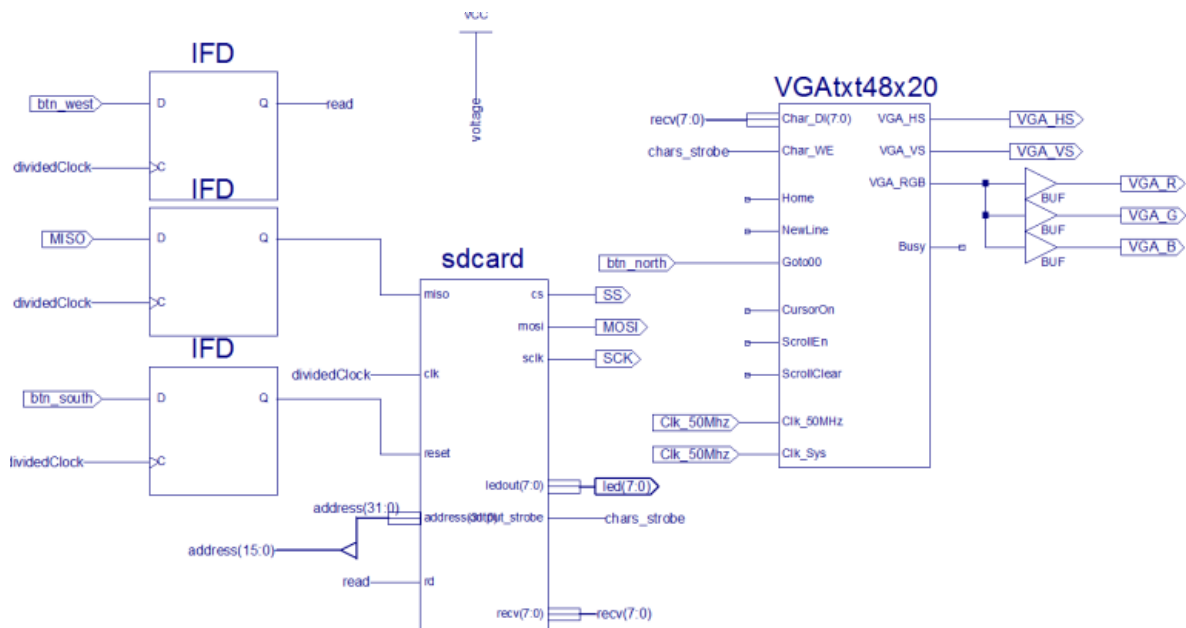
# Opis projektu

## Hierarchia plików

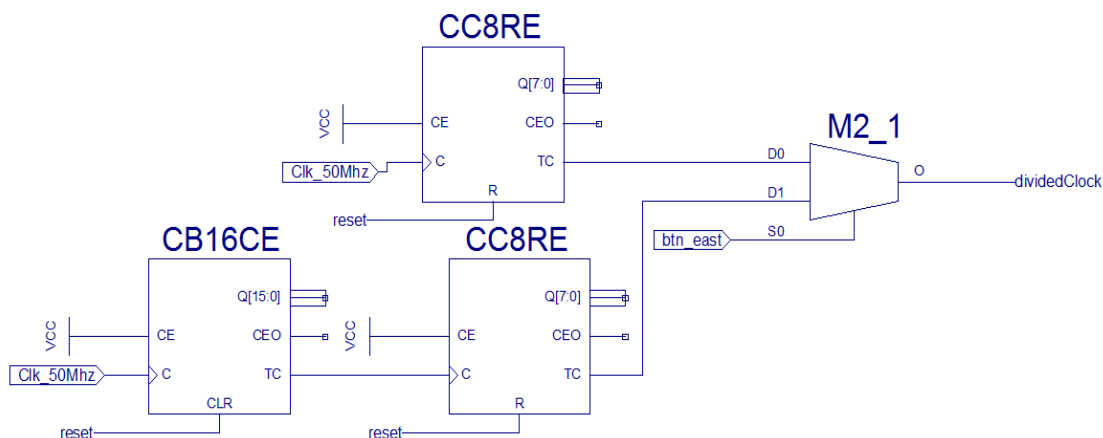


- Projekt zawiera następujące pliki:
- plik .sch zawiera schemat połączeń bezpośrednich do pinów układu Spartan,
- plik sdcard.vhd zawiera implementację symbolu sdcard, który jest odpowiedzialny za komunikację z kartą SD pinami MOSI, MISO, SCK, SS,
- plik sdcard\_test3.vhd zawiera test modułu sdcard (pozwala na obserwowanie jak zachowuje się moduł sdcard na symulowane odpowiedzi z karty),
- pliki RotaryEnc, LCD2x64, \*.ucf to pliki dostarczone wraz z układem Spartan w ramach laboratorium [2].

## Plik schemat.sch

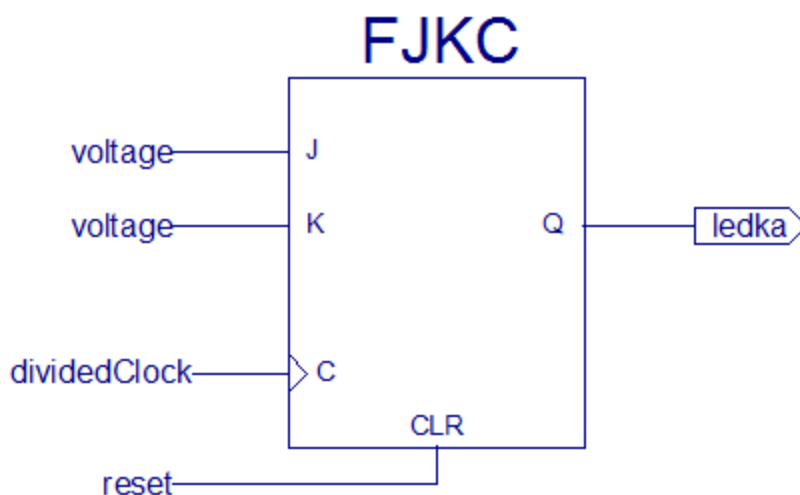


**Schemat główny.** Linia *dividedClock* to sygnał zegara częstotliwości podzielona przez  $2^8$  lub  $2^{24}$  (w zależności od wciśnięcia przycisku btn\_east). Każde wejście pracujące z dużą częstotliwością jest zatrzymywane na czas każdego cyklu pracy układu.

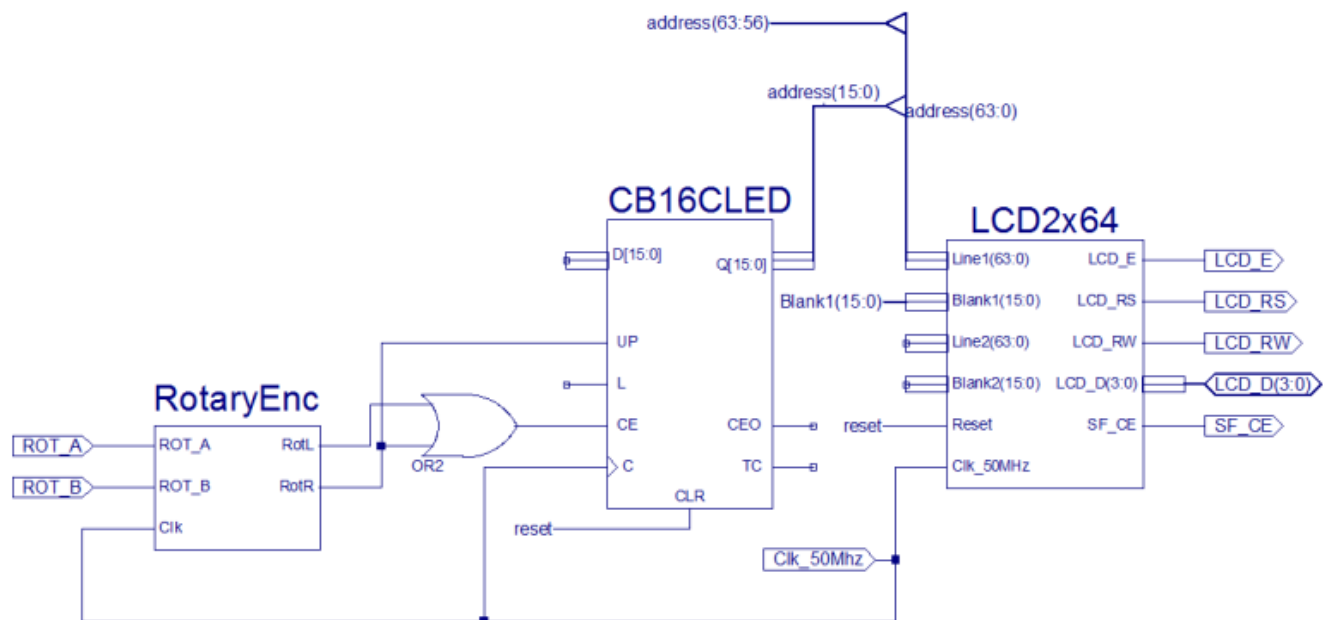


**Fragment schematu realizujący dzielnik częstotliwości demultipleksujący jedną z dwóch częstotliwości w zależności od stanu wciśnięcia przycisku *btn\_east*.**

W celu spowolnienia pracy układu i możliwości zaobserwowania zmiany stanu układu należy przytrzymać przycisk oznaczony jako *btn\_east*. Powoduje to zmniejszenie częstotliwości pracy układu z  $50\text{MHz}/2^8$  na  $50\text{MHz}/2^{(16+8)} \approx 3\text{Hz}$ .



**Fragment schematu informujący o częstości pracy sygnału zegarowego.** Skonfigurowany przerzutnik JK w ten sposób za każdym razem zmienia zapamiętaną wartość - dzięki czemu możliwe jest wizualizowanie pracy zegara.



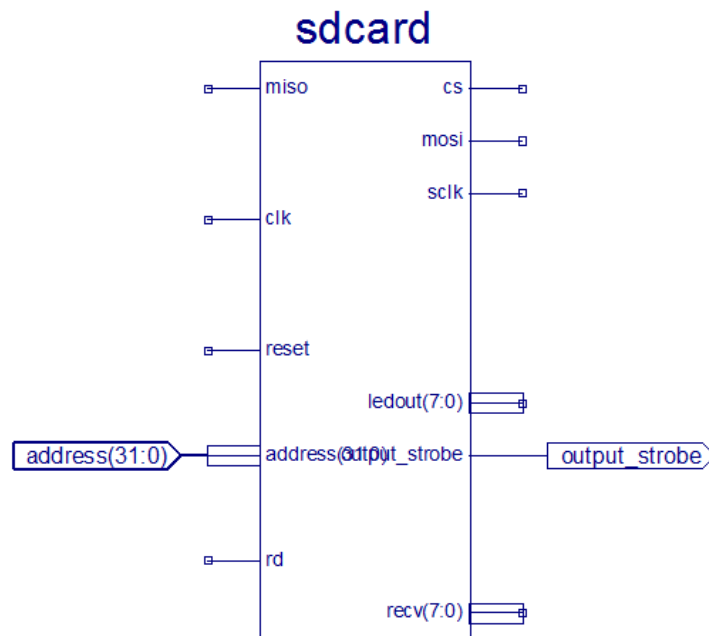
**Fragment schematu realizujący adresację bloku, który zostanie odczytany z karty komendą CMD17.** Przekręcenie pokrętki RotaryEnc w lewo zmniejsza adresu, zaś w prawo - zwiększa go, w wyniku zwiększenia podawany jest stan wysoki na wejście *U* licznika dwukierunkowego *CB16CLED*, co w konsekwencji zwiększa wartość przechowywaną w liczniku. Jeśli pokrętło zostanie przesunięte w prawo - podawany jest stan niski na wejście *U* licznika. Aktualny adres wyświetlany jest na wyświetlaczu *LCD2x64*. Na wyświetlaczu wyświetlane jest również odebrany bajt z karty.

Oto tabela prawd dla licznika *CB16CLED* zaczerpnięta z [3]:

Inputs						Outputs		
CLR	L	CE	C	UP	Dz-D0	Qz-Q0	TC	CEO
1	X	X	X	X	X	0	0	0
0	1	X	↑	X	Dn	Dn	TC	CEO
0	0	0	X	X	X	No change	No change	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

## Moduł sdcard





Wejścia	Wyjścia
<ul style="list-style-type: none"> <li>• MISO- Master Input Slave Output - wejście, którym synchronicznie (do stanu wysokiego zegara CLK) przekazywane są bity z karty SD interfejsem SPI,</li> <li>• CLK - sygnał zegarowy regulujący pracę modułu. Sygnał ten ma bezpośredni wpływ na sygnał sclk wyprowadzany bezpośrednio do karty.</li> <li>• RESET - asynchroniczny, jednokrotny impuls resetuje pracę układu (rozpoczyna inicjalizację od początku),</li> <li>• ADDRESS - magistrala wykorzystywana do wprowadzenia 32-bitowego adresu bloku do odczytania z karty,</li> <li>• RD - jednokrotny synchroniczny impuls stanu wysokiego podany podczas stanu wysokiego sygnału clk w stanie zainicjalizowanej karty powoduje odczytanie bloku o adresie address,</li> </ul>	<ul style="list-style-type: none"> <li>• CS (ang. chip select) - aktywacja karty - pin podłączany bezpośrednio do pinu czytnika kart SD,</li> <li>• MOSI - Master Output Slave Input - wyjście wykorzystywane do przekazywania bitów interfejsem SPI do karty SD, podłączane bezpośrednio do wyprowadzenia układu,</li> <li>• LEDOUT - 8-bitowa magistrala informująca o tym, w jakim stanie znajduje się czytnik kart SD,</li> <li>• OUTPUT_STROBE- podczas odczytywania kolejnych bajtów z karty na to wyjście generowany jest impuls informujący o pojawieniu się nowych danych na magistrali recv,</li> <li>• RECV - 8-bitowa magistrala, z której odczytać można bity wysłane podczas procedury READ_SINGLE_BLOCK [1, str. 156] (bity będące odpowiedzią karty na komendę CMD17). Należy pamiętać, że najniższy bit w tej magistrali to bit odebrany najpóźniej.</li> </ul>

## Implementacja modułu SDCard

### Definicja jednostki

```
39 entity sdcard is
40 port (
41     cs : out std_logic := '1';
42     mosi : out std_logic := '1';
43     miso : in std_logic := '1';
44     rd : in std_logic := '0';
45     sclk : out std_logic := '0';
46
47     ledout : out std_logic_vector(7 downto 0);
48     address : in std_logic_vector(31 downto 0);
49     output_strobe : out std_logic := '0';
50     recv : out std_logic_vector(7 downto 0) := (others => '1');
51     clk : in std_logic; -- podczas pracy z kartą - na każdy takt
52     --pracy karty SD przypadają dwa takty pracy tego modułu
53     reset : in std_logic
54 );
55 end sdcard;
```

### Implementacja maszyny stanów

```
type states is (
    Rst,
    waitingForErasingBuffersEtc,
    Initialize,
    CMD0_sending,
    CMD0_finished,
    waitingFor0,
    waitingFor1,
    CMD1_loading,
    CMD1_sending,
    CMD1_finished,
    readyForReading8Bits,
    initialion_success,
    idle,
    cmd17_load,
    cmd17_sending,
    readingDataBlock,
    SetStrobe0,
    finished
);
```

Stan	Działanie stanu
Rst	resetuje wszystkie sygnały wewnętrzne i wznowia pracę modułu, w tym ustawianie linii CS na stan wysoki, zapalenie diody o wadze 2^0

waitingForErasingBuffersEtc	stan oczekiwania przez 20 taktów pracy karty
Initialize	ustawia linię CS na stan niski (wybór danej karty) oraz odczekuje umowne 160 taktów pracy karty w celu umożliwienia jej inicjalizacji wewnętrznych rejestrów, podaje również komendę CMD0 do sygnału wewnętrznego implementowanego jako rejestr przesuwany w lewo, odpowiedzialnego za podawanie kolejnych bitów komendy
CMD0_sending	wysyłanie komendy CMD0 (wykorzystywanie rejestru przesuwanego w lewo)
CMD0_finished	stan zapalający diodę o wadze $2^1$ oraz przechodzący w stan oczekiwania na stan niski na linii MISO
waitingFor0	oczekiwanie do momentu pojawienia się na linii MISO stanu niskiego
waitingFor1	oczekiwanie do momentu pojawienia się na linii MISO stanu wysokiego, który jednocześnie jest końcem odpowiedzi R1 (0x01)
CMD1_loading	załadowanie komendy CMD1 do rejestru przesuwanego
CMD1_sending	podawanie kolejnych bitów 48-bitowej komendy CMD1 na linię MOSI interfejsu SPI
CMD1_finished	stan oczekujący na przejście linii MISO w stan niski (co jest równoznaczne z rozpoczęciem podawania odpowiedzi przez kartę)
ready4Reading8Bits	wczytanie kolejnych 7 bitów po przejściu karty w stan niski. Jeśli ostatni bit był stanem wysokim, to karta zainicjalizowała się z sukcesem i zgłosiła gotowość do wykonywania kolejnych komend. Jeśli ostatnim bitem nie był bit 1, to należy ponowić wysyłanie komendy CMD1
initialization_success	zapalenie diody o wadze $2^2$ oraz przejście do stanu oczekiwania
idle	stan oczekiwania na zdarzenie żądania odczytu z bloku z karty
cmd17_load	stan wczytania komendy CMD17 z argumentem pobranym z wejścia <i>address</i> modułu <i>sdcard</i> . Zapalona zostaje dioda o wadze $2^3$ .
cmd17_sending	przesyłanie komendy cmd17 do karty SD; ustawienie długości odpowiedzi karty na 512 bajtów (domyślna długość odczytywanego bloku); zostaje zapalona dioda o wadze $2^4$ .
readingDataBlock	odczytywanie pojedynczego bajtu (8 bitów). Po zakończeniu odczytywania praca karty niwstrzymana na około 8 taktów, a na wyjście modułu <i>sdcard</i> output_strobe podawany jest stan wysoki informujący o dostępnym odczytanym bajcie na wyjście <i>recv</i> . Zostaje zapalona dioda o wadze $2^5$ .

SetStrobe0	stan resetujący wyjście <i>output_strobe</i> , oraz w przypadku jeśli są jeszcze bajty do odczytania - wracający do stanu poprzedniego, jeśli odczytano już wszystkie bajty - to przechodzi do następnego stanu. Dioda o wadze $2^5$ gaśnie.
finished	stan w którym zapalona zostaje dioda o wadze $2^6$ .

Dzięki przejściu do trybu spowolnionej pracy (wciśnięcie przycisku btn\_east sterującego dzielnikami częstotliwości) możliwe jest zaobserwowanie przechodzenia pomiędzy poszczególnymi stanami.

### Proces odpowiedzialny za zmianę stanu

```

changestate: process (clk, reset)
begin
    if reset='1' then
        state <= Rst;
    elsif rising_edge(clk) then
        state <= next_state;
    end if;
end process;

```

### Pozostałe procesy odpowiedzialne za wyprowadzanie sygnałów

```

318     sclk <= sclk_sig; -- wyprowadzenie sygnału pracy karty na wyjście
319     mosi <= (cmd_out(47) and cmd_mode) or (read_cmd(55) and (not cmd_mode));
320     -- demultipleksacja komendy 48-bitowa lub 56-bitowa
321     ledout <= status; -- wyprowadzenie aktualnego stanu na diody
322     recv <= received; -- wyprowadzenie danych odczytanych z karty

```

## Implementacja wysyłania komendy CMD0

```
when CMD0_sending =>
    if(bit_counter = 1) then
        next_state<=CMD0_finished;
    else
        if (sclk_sig = '1') then
            bit_counter <= bit_counter-1;
            cmd_out <= cmd_out(46 downto 0) & '0'; --rejestr przesuwany w prawo
        end if;
    end if;
    sclk_sig <= not sclk_sig;

when CMD0_finished =>
    status(0) <= '1';
    next_state <= waitingFor0;

when waitingFor0 =>
    if (sclk_sig = '1') then
        --to moglibysmy zrobic w ten sposob, ze process(miso) jesli w stanie waitingFor0
        if (miso='0') then
            status(1) <= '1';
            next_state <= waitingFor1;
        end if;
    end if;
    sclk_sig <= not sclk_sig;

when waitingFor1 =>
    if (sclk_sig = '1') then
        if (miso='1') then
            next_state <= CMD1_loading;
        end if;
    end if;
    sclk_sig <= not sclk_sig;
```

## Implementacja wysyłania komendy CMD1

```
when CMD1_loading =>

    cmd_out <= x"4100000000095";
    bit_counter <= conv_std_logic_vector(48, 32);
    next_state <= CMD1_sending;

when CMD1_sending =>
    if (sclk_sig = '1') then
        if(bit_counter=1) then
            next_state<=CMD1_finished;
        else
            bit_counter <= bit_counter-1;
            cmd_out <= cmd_out(46 downto 0) & '0'; --rejestr przesuwany w prawo
        end if;
    end if;
    sclk_sig <= not sclk_sig;

when CMD1_finished =>
    if (sclk_sig = '1') then
        if(miso = '0') then
            bit_counter <= conv_std_logic_vector(8, 32);
            next_state <= readyForReading8Bits;
        end if;
    end if;
```



Symulacja składa się z dwóch faz.

Pierwszą jest podawanie stałego sygnału wysokiego na linii MISO (w rzeczywistości karta zachowuje się również w ten sposób - obrazuje to stan przed zainicjalizowaniem karty).

Zegar zmienia swoją wartość co 10us, linia reset jest wzbudzona na 10ms po 100ms od momentu rozpoczęcia symulacji, żądanie odczytu pojedynczego bloku następuje po 550ms od rozpoczęcia symulacji. To opóźnienie zostało dobrane eksperymentalnie (na podstawie analizy wcześniejszej pracy modułu).

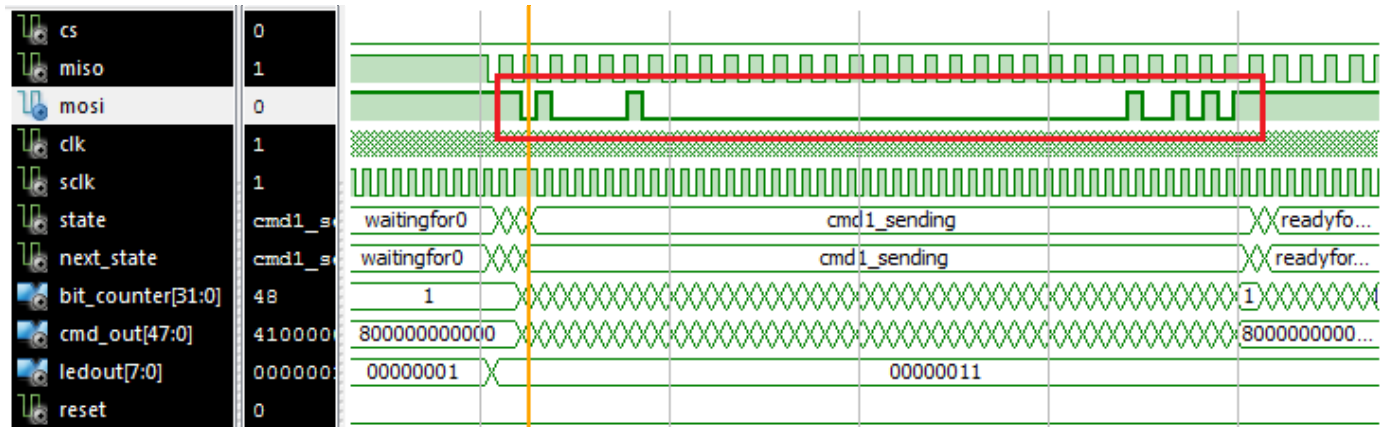
## Wyniki symulacji

Poniżej przedstawiamy wyniki przeprowadzonych symulacji. Można na nich zaobserwować jak działa interfejs SPI oraz zapoznać się ze sposób wymiany informacji.

Signal	Value	Waveform
cs	0	[High]
miso	1	[High]
mosi	1	[High]
clk	1	[High]
sclk	0	[High]
state	cmd0_finished	[High]
next_state	waitingfor0	[High]
bit_counter[31:0]	1	[High]
cmd_out[47:0]	8000000000000000	[High]
ledout[7:0]	00000001	[High]
reset	0	[High]

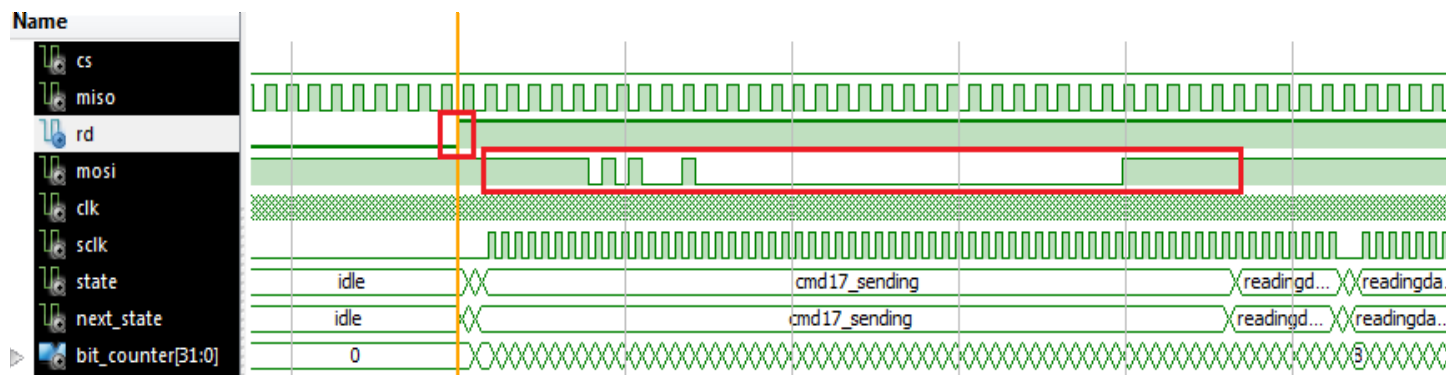
Na przebiegu czasowym linii MOSI widać, że przekazywane są kolejne bity komendy CMD0.

## Wysłanie komendy CMD1



Komenda CMD1 różni się tylko jednym bitem od CMD0. Przesyłanie komendy CMD1 zostało zaznaczone na rysunku.

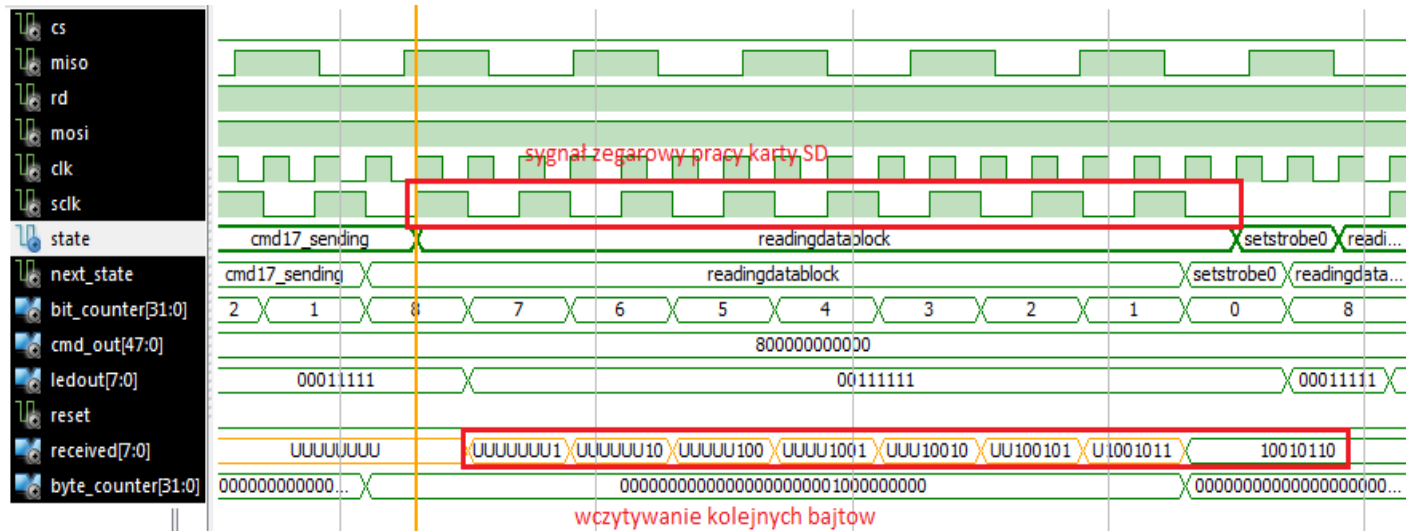
## Wysyłanie komendy CMD17 - odczyt danych z karty



Na rysunku zaznaczono całą komendę żądania przesłania bloku pamięci (komenda zawiera w sobie bajty 0xFF na początku i na końcu).



## Odbieranie kolejnych bajtów w bloku



Odbieranie bajtów zaimplementowane jest przy pomocy rejestru przesuwneego. Został zaznaczony na linii *received[7:0]*.

## Informacje o implementacji i osiągi czasowe

Timing summary:

-----

Timing errors: 0 Score: 0 (Setup/Max: 0, Hold: 0)


Constraints cover 3698 paths, 0 nets, and 1420 connections

Design statistics:

Minimum period: 8.696ns{1} (Maximum frequency: 114.995MHz)

Maksymalna częstotliwość z jaką może pracować układ to 115MHz, jednak zgodnie z dokumentacją [1, str. 3], karta w trybie domyślnym może pracować z częstotliwością do 25MHz, w związku z czym w projekcie stosowane są dzielniki częstotliwości. Nie jest możliwe również wykorzystanie maksymalnej częstotliwości z jaką mógłby pracować układ dlatego, że zainstalowany kwarc w układzie Spartan 3E taktuje z częstotliwością 50MHz.

## Użycie zasobów układu FPGA

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	439	9,312	4%	
Number of 4 input LUTs	725	9,312	7%	
Number of occupied Slices	463	4,656	9%	
Number of Slices containing only related logic	463	463	100%	
Number of Slices containing unrelated logic	0	463	0%	
Total Number of 4 input LUTs	817	9,312	8%	
Number used as logic	722			
Number used as a route-thru	92			
Number used as Shift registers	3			
Number of bonded <a href="#">IOBs</a>	31	232	13%	
IOB Flip Flops	3			
Number of RAMB16s	2	20	10%	
Number of BUFGMUXs	2	24	8%	
Number of RPM macros	19			
Average Fanout of Non-Clock Nets	3.21			

kryterium	użyto	dostępne	wykorzystanie
zajętość LUT (look-up table) - tablica wartości funkcji	725	9312	9%
plastry	439	9312	4%
RAM	2	20	8%

## Instrukcja użytkownika

### Jak podłączyć

W celu podłączenia czytnika kard do układu Spartan 3E należy połączyć następujące piny z odpowiednimi pinami interfejsu SPI.



## SD/MMC Pinout

- 1 - !CS
- 2 - MOSI (DI, Data In)
- 3 - Vss (GND)
- 4 - Vdd + 3.3VDC
- 5 - SCK (CLK, clock)
- 6 - Vss (GND)
- 7 - MISO (DO, Data Out)

Źródło: [4].

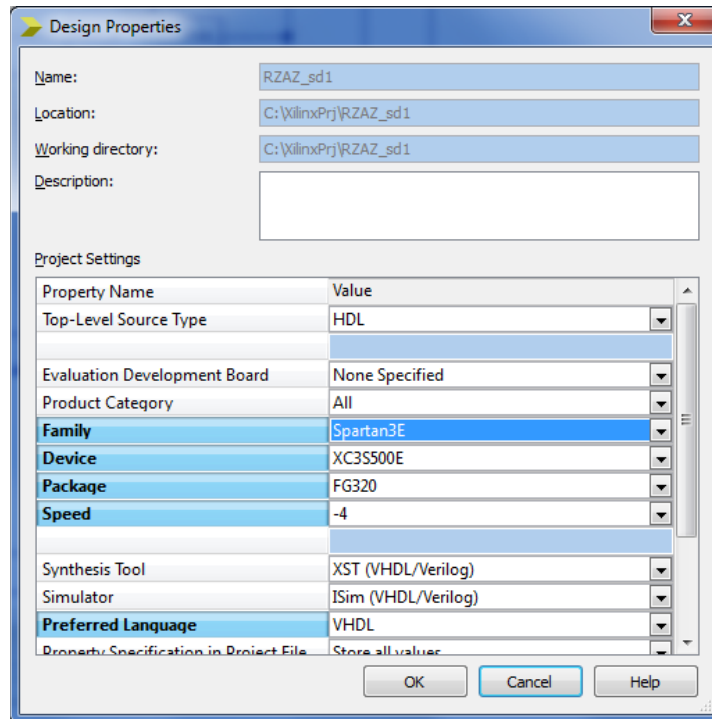
**Ilustracja obrazująca piny karty SD.**

Oznaczenie na układzie Spartan 3E	Oznaczenie w protokole SPI	Nr pinu karty SD
B6	SS - Slave Select	1
E7	Sck - Slave Clock	5
A6	MISO - Master Input Slave Output	7
F7	MOSI - Master Output Slave Input	2

## Budowanie pliku binarnego

Projekt został wykonany w programie Xilinx 14.2. Zaleca się korzystanie z tej wersji w celu uniknięcia problemów z kompatybilnością.

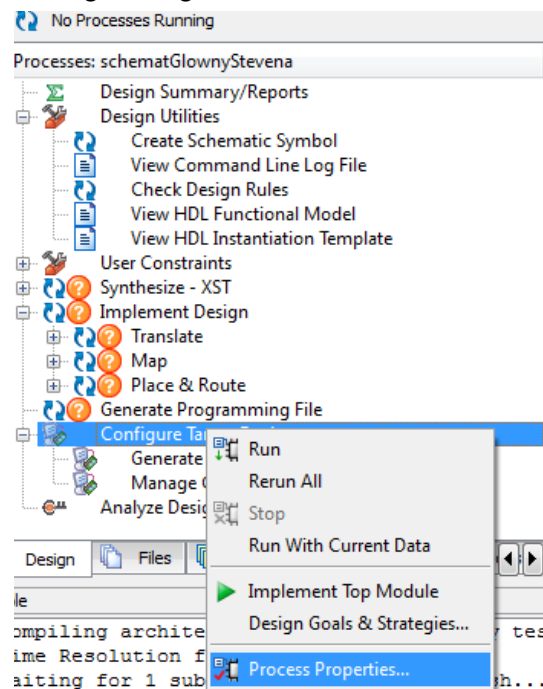
Projekt należy skonfigurować dla następującego urządzenia:



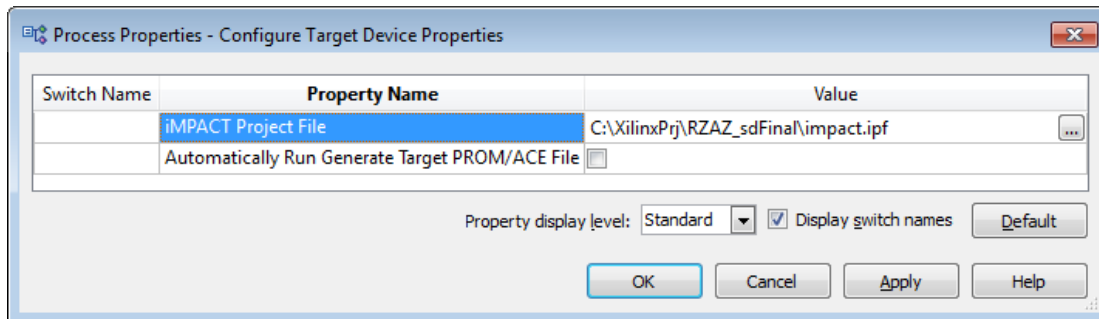
Błękitnym kolorem zaznaczono wiersze, na które należy zwrócić szczególną uwagę.

### **Automatyzacja wgrywania oprogramowania na urządzenie**

Możliwe jest skonfigurowanie środowiska tak, aby program był automatycznie wgrywany na Spartana 3E. W tym celu należy kliknąć prawym na "Configure Target Device" pojawiającą się po zaznaczeniu pliku schematu głównego.



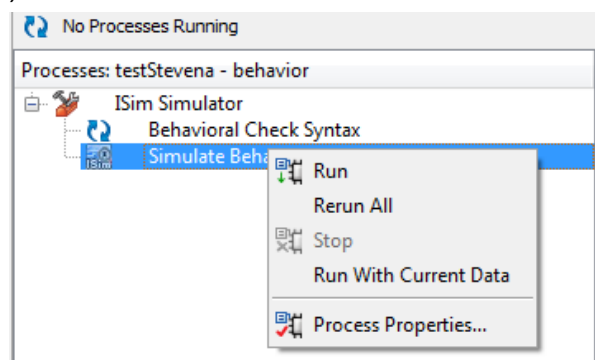
a następnie wybrać



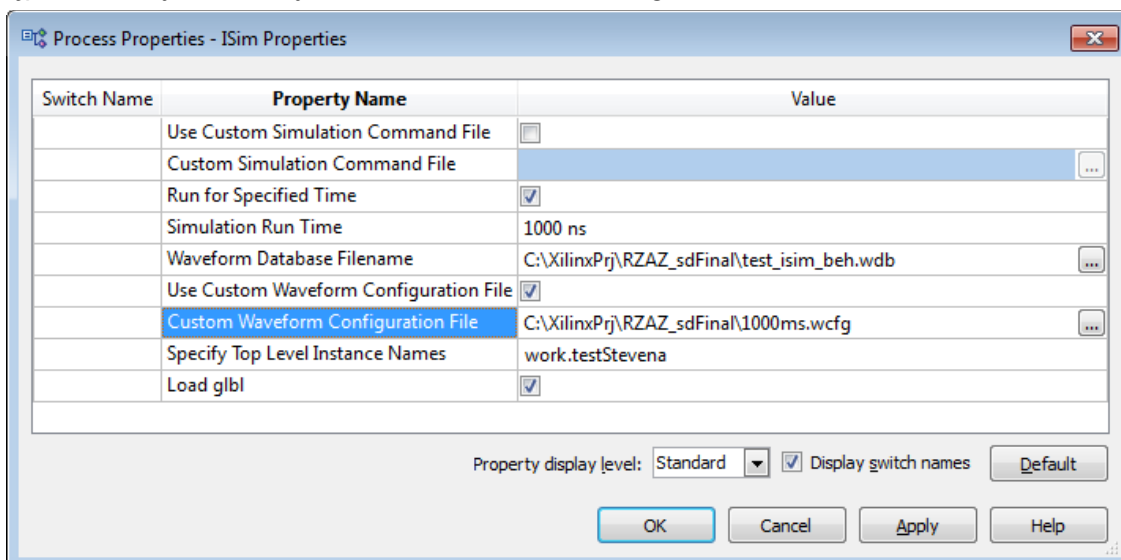
Jeśli pod wskazaną ścieżką znajdzie się projekt zapisany z programu iMPACT, to wgrzywanie programu do układu FPGA będzie odbywać się automatycznie.

### Konfiguracja automatycznego ładowania czasu trwania symulacji oraz podglądanych sygnałów

Możliwe jest również skonfigurowanie domyślnego środowiska do symulacji pracy układów FPGA ISIM, aby wczytywał wcześniej zapisaną konfigurację. W celu osiągnięcia automatyzacji ładowania przebiegów czasowych należy kliknąć na zakładkę symulacja w oknie programu, następnie należy zaznaczyć plik ze źródłem programu symulacyjnego, w polu poniżej należy kliknąć prawym przyciskiem myszy na symulację behawioralną i wybrać preferencje procesu (ang. process properties).



Następnie należy zaznaczyć "custom waveform configuration file"



oraz wskazać wcześniej zapisany plik z przebiegiem czasowym z programu ISIM.

## Obsługa urządzenia

Po zaprogramowaniu układu można wsunąć kartę SD do czytnika kart. W celu odczytania bloku pamięci należy

- podłączyć monitor złączem VGA,
- podłączyć czytnik kart SD do odpowiednich pinów (patrz rozdział “Jak podłączyć”),
- ustawić odpowiedni adres za pomocą pokrętła ROT,
- wcisnąć przycisk btn\_south (reset układu),
- poczekać aż zapalą się 3 diody led o wagach  $2^0$ ,  $2^1$ ,  $2^2$  (sygnalizuje to poprawną inicjalizację karty),
- wcisnąć btn\_west w celu zażądania odczytania bloku o adresie ustawionym RotaryEncoderem,
- obserwować dane prezentowane na ekranie.

## Funkcje klawiszy

	<i>btn_north</i> - ustawienie kursora na współrzędnych (0,0). Wciskanie tego klawisza jest wymagane za każdym razem kiedy użytkownik życzy sobie, aby na ekran zostały wypisane nowe dane	
<i>btn_west</i> - zgłoszenie żądania odczytu bloku danych z karty SD		<i>btn_east</i> - tryb spowolniony. Zmniejsza częstotliwość pracy układu $2^{16}$ razy.
	<i>btn_south</i> - zresetowanie czytnika kart SD.	

## Podsumowanie

Napisanie oprogramowania komunikującego się z kartą SD nie jest trywialnym problemem, jednak nie jest to też zadanie o wysokim stopniu skomplikowania. Jest co najmniej kilka sposobów w jaki można zrealizować odczyt danych z karty SD. Po zdobyciu doświadczenia podczas realizacji tego projektu, mając możliwość zrealizować go jeszcze raz postaralibyśmy się:

- aby implementacja maszyny stanów oparta była na przeskakiwaniu do stanu, który działa analogicznie do funkcji, a po wykonaniu swojej pracy wraca do stanu, z którego został wywołany,
- wykorzystalibyśmy komendy, które umożliwiają:
  - pobranie informacji o stanie inicjalizacji karty,
  - poznanie rozmiaru nośnika,
  - zmianę odczytywanego bloku,
  - odczytywanie danych z karty o pojemności większej niż pojemność systemu plików FAT - 4GB.

# Repozytorium projektu

Projekt jest dostępny pod adresem [https://github.com/arjamizo/vhdl\\_sd\\_card\\_reader\\_spartan](https://github.com/arjamizo/vhdl_sd_card_reader_spartan).

## Bibliografia

1. Dokumentacja kart SD firmy SanDisk  
[https://www.sdcard.org/downloads/pls/simplified\\_specs/part1\\_410.pdf](https://www.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf)
2. <http://www.zsk.ict.pwr.wroc.pl/zsk/dyd/intinz/uc/>  
([http://www.zsk.ict.pwr.wroc.pl/zsk\\_ftp/fpga/](http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/))
3. Xilinx ISE 14.2 Help - Symbol Info - pomoc srodowiska Xilinx 14.2
4. <http://www.bot-thoughts.com/2010/02/logging-data-to-sd-cards.html>