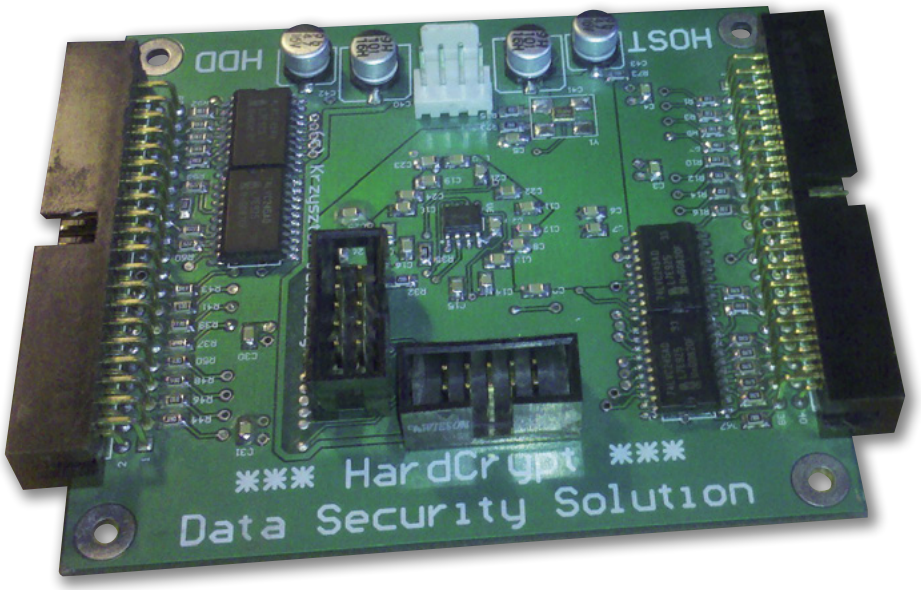
Szyfrator dysków

Sprzętowe szyfrowanie danych na dysku twardym

AVT 5309

Współcześnie, wraz z postępującym rozwojem technologii informacyjnej, coraz większy nacisk kładzie się na zabezpieczenie danych. Głównym zadaniem kryjącym się pod hasłem „bezpieczeństwo" jest dążenie do zachowania poufności przetwarzanych informacji. Naturalnym, nasuwającym się rozwiązaniem jest ich szyfrowanie.

Rekomendacje: *urządzenie przyda się do utajniania danych np. na dysku – kopii, tak aby były niedostępne dla osób postronnych.*



Mało kto zdaje sobie sprawę, że ustawodawca również nałożył na nas obowiązek bezpiecznego przechowywania danych. Mówi o tym rozporządzenie Ministra Spraw Wewnętrznych i Administracji (Dz. U. z 2004 r. Nr 100, poz. 1024) z dnia 29 kwietnia 2004 r. Określa ono warunki techniczne i organizacyjne, którym powinny odpowiadać urządzenia i systemy informatyczne służące do przetwarzania danych osobowych. Wspomina się w nim również o obowiązku stosowania ochrony kryptograficznej. W artykule zaproponuję zbudowanie sprzętowego systemu szyfrowania danych, który będzie całkowicie transparentny dla systemu operacyjnego oraz do swojego działania nie będzie wykorzystywał zasobów sprzętowych komputera. Kompletny projekt jest oparty na układzie programowalnym FPGA Cyclone II firmy Altera. Współpracuje z dyskami twardymi IDE/ATA oraz większością płyt głównych obsługujących tryby UltraDMA. Całość składa się z płyty bazowej oraz podłączanego modułu czytnika kart zawierających klucze szyfrowania. Zaimplementowany algorytm szyfrujący to rozszerzony do 128 bitów algorytm DES (istnieje możliwość modyfikacji układu w celu implementacji algorytmu AES – wymagane jest jedynie zastosowanie układu FPGA dysponującego większymi zasobami). Klucz szyfrowania danych przechowywany może być na karcie pamięci SD lub popularnej karcie chipowej SLE4428.

Projekt zostanie opisany zarówno od strony sprzętowej, jak i opisu sprzętu w języku VHDL. Poruszony będzie problem zapobiegania zjawiskom przepięć i przesłuchów związanych z szybkością narastania zboczy szybkich sygnałów interfejsowych. Omówiona zostanie również implementacja soft-core'owego procesora NIOS II, umożliwiająca programowanie kart chipowych SLE4428. Będzie to pomocne dla Czytelników nie posiadających specjalizowanych programatorów kart SLE4428. W wypadku kart SD klucz szyfrujący stanowi plik umieszczony w systemie plików FAT16, więc generowanie klucza nie powinno stanowić większego problemu. Obsługa odczytu kluczy z obu kart jest realizowana całkowicie sprzętowo (przez FPGA).

Interfejs IDE/ATA

Interfejs IDE/ATA był przez długie lata najczęściej stosowanym standardem połączeń napędów optycznych i dyskowych. Po raz pierwszy został zaproponowany przez firmę Compaq, a jego historia sięga 1983 roku. Od 2003 roku jest jednak powoli wypierany przez magistralę Serial ATA. Nadal jednak jest bardzo rozpowszechnionym i wciąż jeszcze wykorzystywanym standardem. Magistrala ATA jest magistralą równoległą. Połączenie z dyskiem twardym początkowo było wykonywane za pomocą 40-żyłowej taśmy typu AWG40. W miarę rozwoju standardu dążono do uzyskania coraz wyższej przepustowości, co wiązało się ze zwiększaniem częstotliwości sygna-

AVT-5309 w ofercie AVT:
AVT-5309A – płytka drukowana

Podstawowe informacje:
- Szyfrowanie danych podczas ich zapisu, deszyfrowanie podczas odczytu
- Metoda szyfrowania DES z kluczem 128-bitowym
- Możliwość ustalenia własnego, unikatowego klucza zapisywanego w pamięci zewnętrznej (np. na karcie SD)
- Zasilanie z zasilacza komputerowego
- Szybkość transmisji: UDMA66, ale obsługuje też wyższe tryby

Dodatkowe materiały na CD/FTP:
ftp://ep.com.pl, user: 19623, pass: 6c5r20n3
- wzory płytek PCB
- karty katalogowe i noty aplikacyjne elementów oznaczonych w Wykazie elementów kolorem czerwonym

Projekty pokrewne na CD/FTP:
(wymienione artykuły są w całości dostępne na CD)
AVT-5139 Przejściówka USB-ATA z szyfrowaniem danych AES (EP 7-8/2008)

Tabela 1. Rozkład sygnałów na złączu IDE/ATA. W trybie UltraDMA sygnały DIOR_N, DIOW_N, IORDY pełnią nieco inną rolę, ale o tym później.

Pin	Oznaczenie	Opis	Pin	Oznaczenie	Opis
1	RESET_N	Sprzętowa inicjalizacja (RESET) dysku	2	GND	Masa
3..18	DD7..DD0	Magistrala danych dysku	3..18	DD15..DD8	Magistrala danych dysku
19	GND	Masa	20		Złącze nie wykorzystane
21	DMARQ	Żądanie transferu DMA	22	GND	Masa
23	DIOW_N	Sygnał zapisu danych z magistrali	24	GND	Masa
25	DIOR_N	Sygnał odczytu danych z magistrali	26	GND	Masa
27	IORDY	Sygnał gotowości urządzenia	28	CSEL	Wybór napędu master/slave
29	DMACK_N	Potwierdzenie żądania DMA przez hosta	30	GND	Masa
31	INTRQ	Żądanie obsługi przerwania	32	IOCS16	Obecnie sygnał przestarzały
33	DA1	Linia adresowa używana do adresowania rejestrów napędu	34	PDIAG_N	Sygnalizuje zakończenie inicjalizacji drugiego dysku
35	DA0	Linia adresowa używana do adresowania rejestrów napędu	36	DA2	Linia adresowa używana do adresowania rejestrów napędu
37	CS0_N	Umożliwia komunikację z rejestrami komend	38	CS1_N	Umożliwia komunikację z rejestrami kontrolnymi
39	DASP_N	Informuje pierwszy dysk fizyczny (master) o obecności drugiego	40	GND	Masa

łów interfejsu. Dla poprawienia charakterystyki i zminimalizowania przesłuchów między liniami, zastosowano taśmę 80-żyłową, która jest stosowana począwszy od wprowadzenia do standardu trybu UltraDMA 66. Dodatkowe żyły tej taśmy połączone są z masą. Rozkład sygnałów na złączu prezentuje **tabela 1**.

Realizacja sprzętowa

Projekt oparty jest na idei przechwytywania sygnałów z/do dysku twardego i odpowiedniej ich modyfikacji przy zaistnieniu pewnych warunków. Należy więc wszystkie sygnały „przepuścić" przez nasz urządzenie. Wbrew pozorom nie jest to zadanie proste, ponieważ magistralę IDE/ATA pierwotnie przewidziano do współpracy z urządzeniami w standardzie TTL, a każde kolejne wydanie jej specyfikacji zakładało wsteczną zgodność z tym wymogiem, pomimo że współczesne dyski twarde wykorzystują do transferu danych sygnały CMOS o poziomie wysokim wynoszącym 3,3 V. Na potrzeby dostosowania poziomów napięć, wykorzystano dwukierunkowe układy buforujące typu 74LVC245, które zapewniają prawidłową pracę szyfratora w razie pojawienia się na magistrali napięcia wyższego od napięcia zasilania bloków I/O układu FPGA (3,3 V). To jednak nie jedyny problem, który należało pokonać przy podłączeniu do interfejsu. Kolejne wersje specyfikacji ATA/ATAPI przewidywały wzrosty prędkości transmisji, co wiąże się ze wzrostem szybkości narastania zboczy ($\Delta U/\Delta t$) sygnałów, co z kolei nakazuje konstruktorowi zwrócić szczególną uwagę na zjawiska falowe. Taśmę połączeniową między dyskiem twardym a komputerem można traktować jak typową linię długą, w której zachodzą m.in. zjawiska odbić. Można zaobserwować, że w przypadku odbić, na skutek oscylacji sygnał dwukrotnie przechodzi przez próg przełączenia 74LVC245, co może powodować zaburzenia transmisji. Żeby tego uniknąć zastosowano szeregowy model terminacji linii, sugerowany w specyfikacji ATA/ATAPI. Wartości rezystorów po stronie hosta oraz urządzenia zamieszczono w **tabeli 2**. Ponadto, zgodnie ze specyfikacją ATA/ATAPI niektóre z sygnałów wymagają zewnętrznych rezystorów podciągających lub ściągających.

Tabela 2. Wartości rezystorów terminujących według specyfikacji ATA/ATAPI.

Sygnał	Host	Dysk
DIOR	22 Ω	82 Ω
DIOW	22 Ω	82 Ω
CS0, CS1	33 Ω	82 Ω
DA0, DA1, DA2	33 Ω	82 Ω
DMACK	22 Ω	82 Ω
DD15..DD0	33 Ω	33 Ω
DMARQ	82 Ω	22 Ω
INTRQ	82 Ω	22 Ω
IORDY	82 Ω	22 Ω
RESET	33 Ω	82 Ω

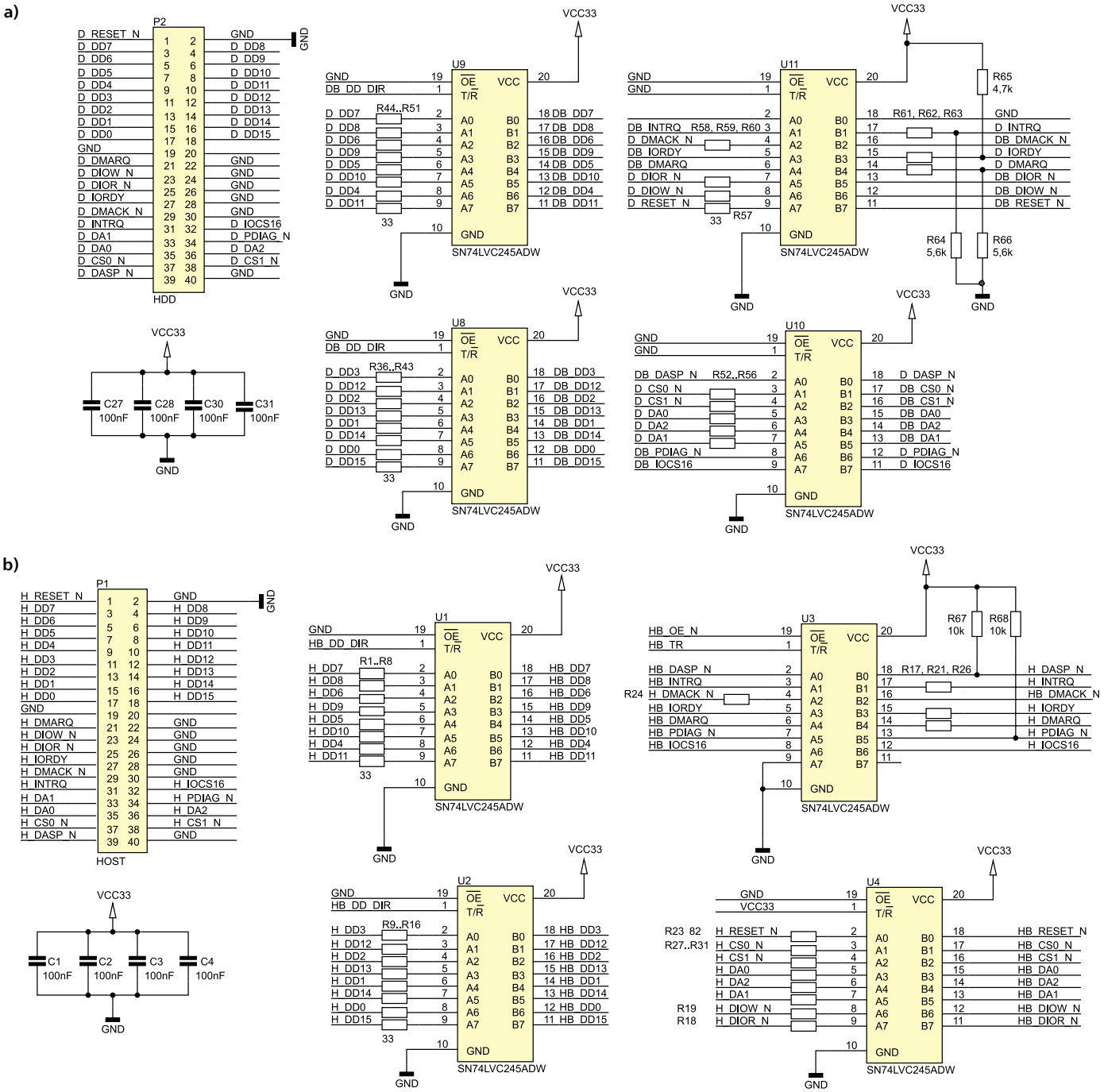
Schemat dołączenia układów buforujących pokazano na **rysunku 1**. Podczas projektowania PCB dla tego obwodu należało zwrócić uwagę również na długość ścieżek. Dopuszczalna przez specyfikację różnica długości ścieżek między liniami danych (DD15..DD0) i sygnałami strobującymi (*DIOW_N*, *DIOR_N*, *IORDY*) wynosi 1,27 cm. Gdyby nie spełniono tego wymagania, mogłoby się zdarzyć, że dane zostałyby zatrzaśnięte w buforze jeszcze przed ich ustabilizowaniem się.

Schemat dołączenia układu FPGA pokazano na **rysunku 2**. Układ FPGA zastosowany w urządzeniu wymaga doprowadzenia dwóch napięć zasilania o wartości 1,2 V i 3,3 V (schemat zasilacza zaprezentowano na **rysunku 3**). Komplikuje to nieco prowadzenie ścieżek na dwuwarstwowym obwodzie drukowanym, dlatego zastosowano pseudopłaszczyzny zasilania na warstwach *top* i *bottom*, wokół ścieżek sygnałowych.

Rysunek 4 przedstawia schemat modułu czytnika kart chipowych oraz SD dołączanego do urządzenia szyfrującego, natomiast rysunek 5 schemat dołączenia jego złącza oraz generatora zegarowego. Jego budowa nie jest skomplikowana. Komentarza wymagać mogą jedynie diody D1...D3. Zastosowano je, ponieważ karta SLE4428 wymaga napięcia zasilania 5 V, natomiast banki FPGA zasilane są napięciem 3,3 V. Diody w parze z rezystorami obniżają poziom sygnałów wychodzących z karty chipowej.

Opis logiczny sprzętu

Strukturę sprzętu opisanego w układzie FPGA pokazano na rysunku 6. Przedstawia on tzw. „top level" (najwyższy poziom) i jest zrealizowany nietypowo –

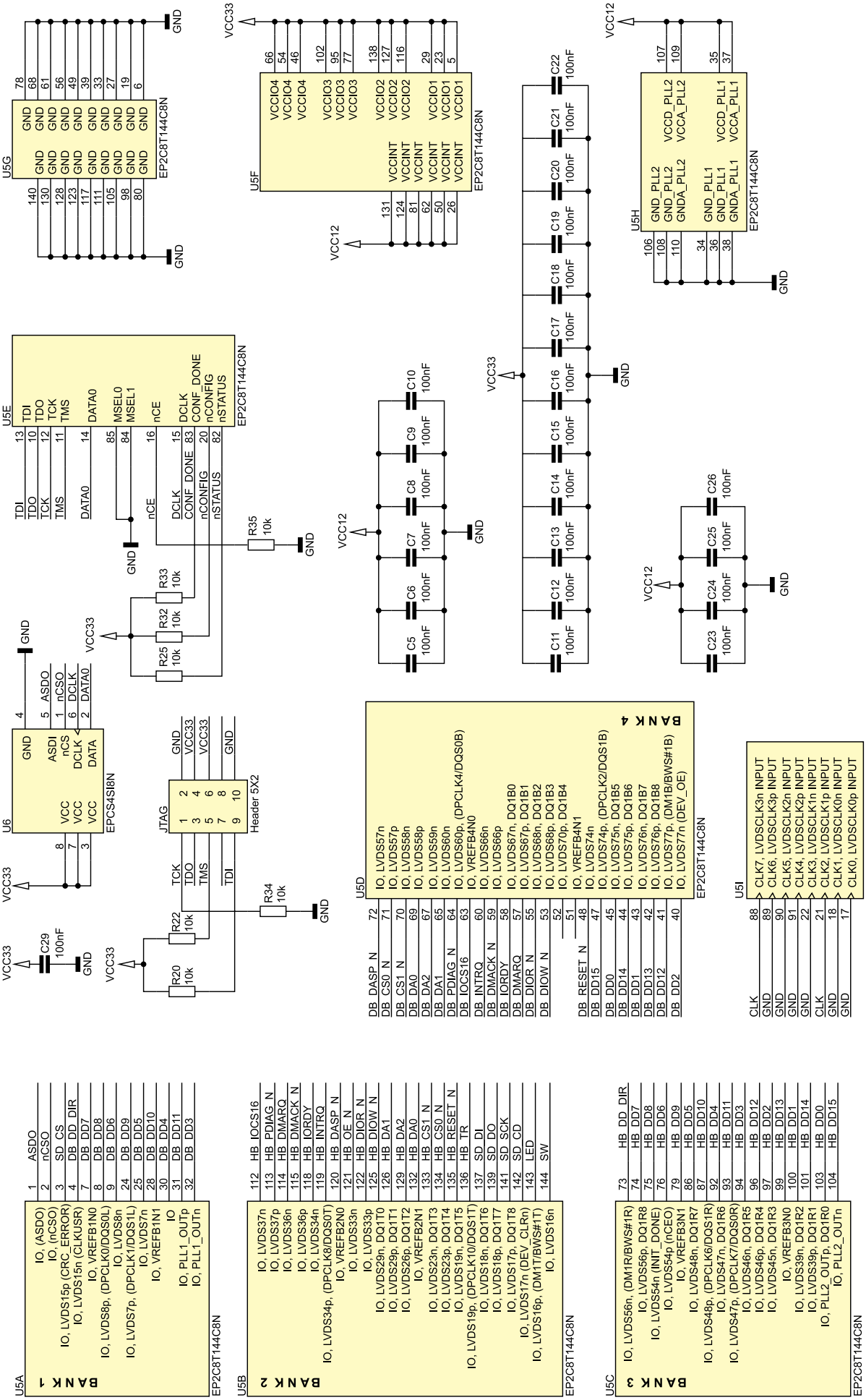


Rysunek 1. Schemat podłączenia układów buforujących sygnały: a) dysk twardy/układ FPGA, b) host/układ FPGA

w postaci schematu środowiska Altera Quartus. Wygodniejszym podczas rozwijania projektu oraz bardziej uniwersalnym sposobem jest jego opis poprzez plik VHDL/Verilog. Ten pierwszy sposób ma jednak zaletę w przypadku prezentowania gotowej całości – schemat blokowy jest bowiem zdecydowanie bardzie czytelny.

io_ctrl.vhd. Jest najprostszym blokiem funkcjonalnym układu i ma za zadanie jedynie zsynchronizowanie sygnałów zewnętrznych z dysku twardego oraz hosta z sygnałem zegarowym, taktującym logikę. Sygnały te należy „zatrzasnąć" w przerzutnikach D, aby uniknąć problemu z odczytem stanów metastabilnych. Sygnały wychodzące z tego bloku są już zgodne z domeną zegarową naszej logiki i możemy je odtąd „bezpiecznie" analizować.

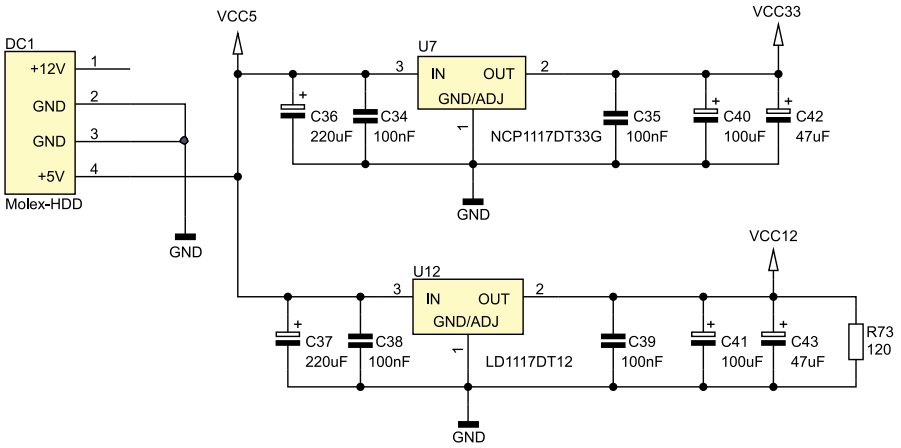
data_bus_ctrl.vhd. Jak pamiętamy, w układzie zastosowano dwukierunkowe układy buforujące (74LVC245) na magistrali danych dysku twardego. Należy wobec tego zadbać o odpowiednie sterowanie kierunkiem buforów. To zadanie pełni omawiany blok logiczny. Proces sterowania kierunkiem wbrew pozorom nie jest zadaniem banalnym, zależy on bowiem od tego w jakim trybie aktualnie znajduje się dysk twardy. Podstawowym trybem komunikacji dla dysku twardego PATA jest tryb PIO. W ten sposób następuje zapis i odczyt rejestrów wewnętrznych. Można również żądać dostępu do sektorów dysku w trybie PIO, jednak jest to obecnie niepraktykowane ze względu na niską wydajność tego sposobu transferu danych. Wykrywanie kierunku transmisji jest w tym przypadku proste: gdy zostanie wykryty stan niski na linii hosta DIOR_N, oznaczającej odczyt danych z dysku, należy bufor dysku twardego ustawić jako wejście (następuje przenoszenie sygnałów z magistrali danych dysku do układu FPGA), a bufor hosta jako wyjście (następuje przenoszenie sygnałow z układu FPGA na magistralę danych hosta). Wykrycie żądania zapisu na linii DIOW_N wymusza sytuację odwrotną. Niestety w przypadku transferów w trybach UltraDMA, sygnały DIOW_N oraz DIOR_N przyjmują inną funkcję i nie można kontrolować



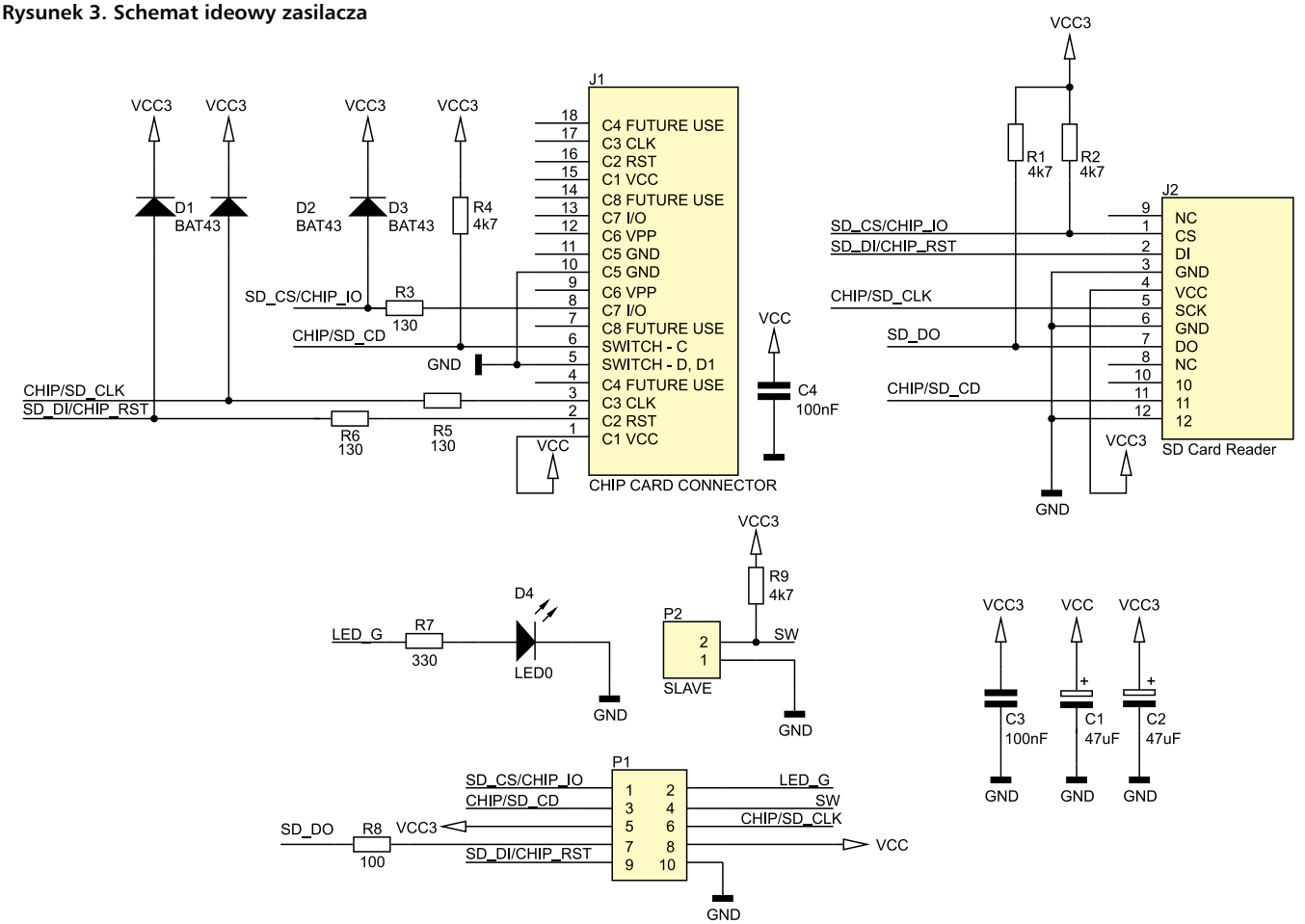
Rysunek 2. Schemat podłączenia układu FPGA

kierunku na ich podstawie. Do bloku data_bus_ctrl doprowadzono zatem dodatkowe sygnały logiczne informujące, iż aktualnie urządzenie jest w trakcie transferu UDMA wejściowego, wyjściowego lub jest w fazie, w której na magistralę powinno przekazać CRC danych. Są to kolejno sygnały UDMA_IN, UDMA_OUT, UDMA_CRC. Sterowanie kierunku w języku opisu sprzętu zrealizowane jest w postaci typowego automatu stanów, gdzie stan kolejny jest wyjściem enkodera priorytetowego, uwzględniającego opisane wyżej sygnały. Dodatkowo do bloku doprowadzono sygnał HIGHZ wymuszający stan wysokiej impedancji na magistrali hosta. Jest to sygnał, który można wykorzystać przy rozwijaniu urządzenia, tak aby na tym samym kanale IDE mogło współpracować z drugim dyskiem twardym bądź napędem CD/DVD. Wtedy w momencie, gdy drugi dysk będzie korzystał z magistrali należy to umożliwić przez wprowadzenie buforów w stan wysokiej impedancji za pomocą sygnału HIGHZ.

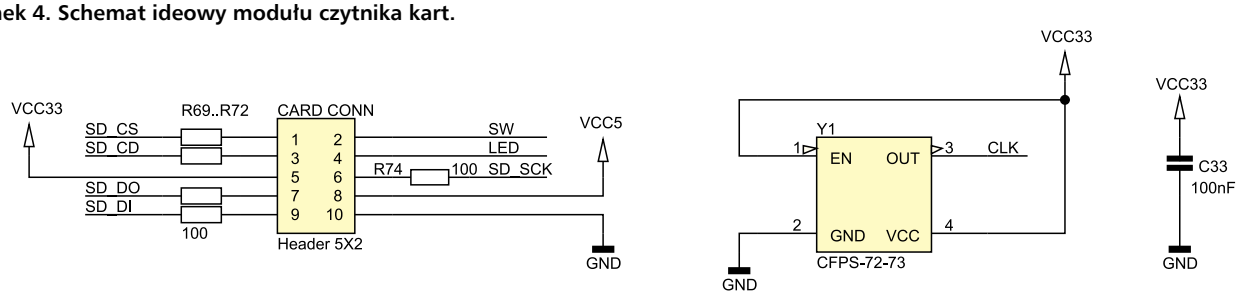
main_ctrl.vhd. To właśnie w tym bloku skupia się filozofia działania całego układu. Na początku warto wspomnieć o ograniczeniach. Otóż logika sterująca całym procesem szyfrowania jest dość rozbudowana, wobec czego szczytem możliwości dla układu EP2C8 było taktowanie jej sygnałem zegarowym 144 MHz. Ogranicza to nas do obsługi maksymalnie trybu UltraDMA 66, który zapewnia transfer danych do



Rysunek 3. Schemat ideowy zasilacza



Rysunek 4. Schemat ideowy modułu czytnika kart.



Rysunek 5. Schemat ideowy generatora kwarcowego oraz złącza czytnika kart

Listing 1. Przechwytywanie zawartości rejestrów dysku twardego w języku VHDL.

```
process(CLK,RESET)
begin
  if RESET='1' then
    cmd_reg<=(others=>'0');
    features_reg<=(others=>'0');
    sector_cnt_reg<=(others=>'0');
    devhead_reg<=(others=>'0');
    cylhigh_reg<=(others=>'0');
    cyllow_reg<=(others=>'0');
    secnum_reg<=(others=>'0');
    cylhigh_reg_last<=(others=>'0');
    cyllow_reg_last<=(others=>'0');
    secnum_reg_last<=(others=>'0');
    dma_rd<='0';
    dma_wr<='0';
    elsif CLK'event and CLK='1' then
      cmd_exec<='0';
      if cs_n="10" then
        if diow_n='1' and diow_n_last='0' then
          case da is
          when „111"=>
          cmd_reg<=DATA_FROM_HOST(7 downto 0);
          if DATA_FROM_HOST(7 downto 0)=DMA_READ or DATA_FROM_HOST(7 downto
0)=DMA_READ_EXT then
            dma_rd<='1';
          else
            dma_rd<='0';
          end if;
          if DATA_FROM_HOST(7 downto 0)=DMA_WRITE or DATA_FROM_HOST(7 downto
0)=DMA_WRITE_EXT then
            dma_wr<='1';
          else
            dma_wr<='0';
          end if;
          cmd_exec<='1';
          when „001"=>
            features_reg<=DATA_FROM_HOST(7 downto 0);
          when „010"=>
            sector_cnt_reg<=DATA_FROM_HOST(7 downto 0);
          when „110"=>
            devhead_reg<=DATA_FROM_HOST(7 downto 0);
         when „101"=>
           cylhigh_reg<=DATA_FROM_HOST(7 downto 0);
           cylhigh_reg_last<=cylhigh_reg;
         when „100"=>
           cyllow_reg<=DATA_FROM_HOST(7 downto 0);
           cyllow_reg_last<=cyllow_reg;
         when „011"=>
           secnum_reg<=DATA_FROM_HOST(7 downto 0);
           secnum_reg_last<=secnum_reg;
         when others=>
           NULL;
          end case;
        end if;
      end if;
    end if;
end process;
```

66 MB/s. W praktyce jest to wynikiem na tyle dobrym, że większość standardowych dysków twardych obsługujących wyższe standardy UltraDMA i tak nie jest w stanie osiągnąć wyższych transferów danych. Można więc uznać, że w większości przypadków urządzenie nie spowalnia dysku bądź spowalnia go w niewielkim stopniu.

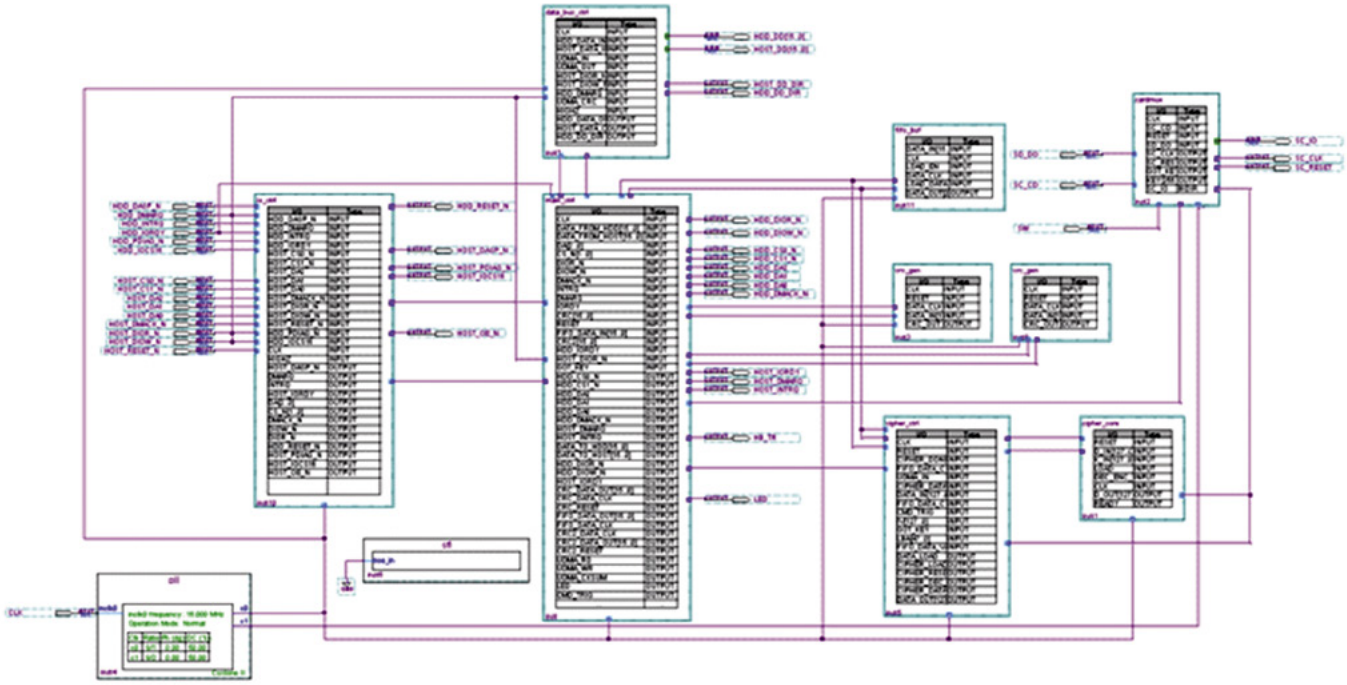
Po podłączeniu zasilania, host powinien rozpocząć procedurę detekcji urządzeń podłączonych do magistrali oraz rozpoznania ich możliwości. Jedną z pierwszych wysyłanych komend jest komenda IDENTIFY DRIVE o kodzie 0xEC. W odpowiedzi na nią, dysk twardy zwraca 512-bajtową strukturę (256 16-bitowych słów).

Listing 2. Kod wrappera ograniczającego obsługiwane przez dysk twardy tryby UDMA

```
process(CLK,RESET)
begin
  if RESET='1' then
    data_cnt<=0;
    wrap_mask<=(others=>'1');
    elsif CLK'event and CLK='1' then
    if dior_n='0' and dior_n_last(1)='1' then
      if cs_n="10" and da="000" then
        if cmd_reg=IDENTIFY_DRIVE then
          data_cnt<=data_cnt+1;
        else
          data_cnt<=0;
        end if;
        case data_cnt is
        when 83=>wrap_mask<="1111111111111101";
        when 88=>wrap_mask<="0001111100011111"; -- udma66
        when 255=>wrap_mask<=(others=>'0'); -- no checksum
        when others=>wrap_mask<=(others=>'1');
        end case;
      else
        wrap_mask<=(others=>'1');
        data_cnt<=0;
      end if;
    end if;
  end if;
end process;
```

W strukturze tej w słowie 88. umieszczono informacje o obsługiwanych trybach UltraDMA urządzenia. Jeżeli dołączony dysk twardy będzie wspierał tryb wyższy, np. UDMA 100, to host dążąc do jak największej szybkości działania wybierze ten tryb transferu danych. Jak wspomniałem wcześniej, szyfrator obsługuje maksymalnie UDMA66, toteż trzeba w jakiś sposób „oszukać" hosta i zamaskować bity słowa 88 odpowiadające za wyższe tryby. Do tego celu opisany został w języku VHDL prosty wrapper, którego kod umieszczono na **listingu 2**. Wcześniej jednak proponuję się zapoznać z **listingiem 1**. Przedstawia on opis sprzętu, który wyłapując przepływające dane, de facto tworzy kopię niektórych rejestrów dysku twardego. Dzięki temu możemy stwierdzić jaka jest aktualnie wykonywana komenda oraz jakie są jej parametry. Dodatkowo jeśli wykryta zostanie komenda zapisu bądź odczytu w trybie DMA, to ustawiane są odpowiednie rejestry dma_wr/dma_rd na podstawie których reszta opisanego sprzętu (np. blok kontrolujący magistrale danych urządzeń) może odpowiednio inaczej się zachować. Zgłębianie się w każdy szczegół standardu ATA/ATAPI jest materiałem na książkę, a nie artykuł miesięcznika, toteż opis każdego rejestru dysku twardego oraz sposobu dostępu do niego pozostawiam jako temat do przestudiowania przez dociekliwych Czytelników np. na podstawie źródeł dostępnych w Internecie.

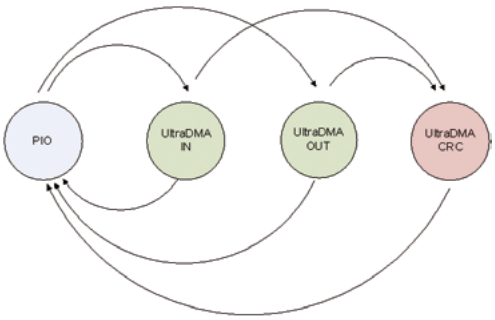
Sam kod wrappera składa się z licznika słów, zerowanego zawartością rejestru komend inną niż IDENTIFY DRIVE (0xEC). Dodatkowo przy pozytywnym porównaniu wartości licznika z numerem interesującego nas słowa (88), następuje zapis do rejestru wrap_mask odpowiedniej maski bitowej, która zeruje bity odpowiedzialne za tryby UltraDMA wyższe niż UDMA66. Każdy zapis na magistralę danych hosta jest w innej części bloku main_ctrl.vhd poddawany operacji logicznej AND z wrap_mask, dzięki czemu wyzerowane bity rejestru wrap_mask bezpośrednio wpłyną na dane



Rysunek 6. Schemat blokowy sprzętu opisanego w FPGA

które otrzyma w odpowiedzi kontroler dysku (host). Po całej operacji z punktu widzenia komputera nasz dysk twardy będzie widziany jako niewspierający trybów powyżej trybu UltraDMA 66.

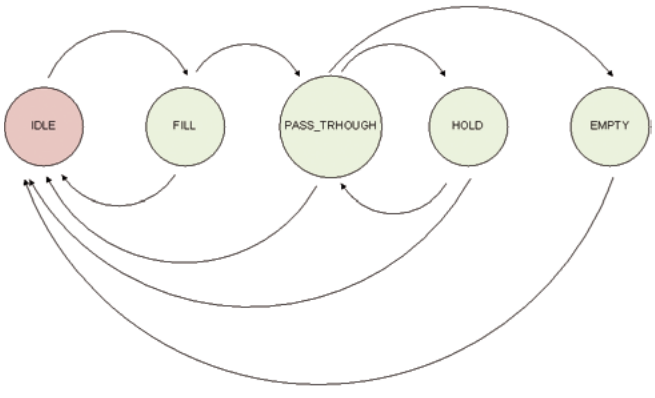
Jak wspomniano wcześniej, ze względu na niską szybkość oraz obciążenie procesora zaczęto uciekać od trybów PIO na rzecz trybów bezpośredniego dostępu do pamięci (DMA). Najbardziej spopularyzowany i najszybszy jest UltraDMA. Wprowadzenie UDMA stanowiło prawdziwą rewolucję. Dopiero od czasu ustandaryzowania trybu UDMA, odseparowano sygnały taktujące dane urządzenia nadającego od odbierającego. Dotychczas to host taktował dane zarówno przychodzące jak i wychodzące, co wobec nieprzewidywalnych opóźnień wprowadzanych przez taśmę oraz drivery linii znacznie ograniczało możliwą do uzyskania przepustowość. Standard UltraDMA zdefiniował nowe funkcje dla linii DIOR, DIOW oraz IORDY na potrzeby inicjalizacji oraz transferów danych. DIOW pełnił dodatkowo funkcję STOP, DIOR – HSTROBE oraz HDMARDY, IORDY – DDMARDY oraz DSTROBE. Dodatkową cechą charakterystyczną dla UDMA jest zatrzaskiwanie danych na obydwu zboczach sygnału taktującego. Pozwoliło to na podwojenie przepustowości, nie zwiększając przy tym częstotliwości sygnału taktującego. W celu zapewnienia niezawodności i bezpieczeństwa danych wprowadzono wymóg obliczenia 16-bitowej sumy kontrolnej. Suma ta zawsze obliczana jest przez hosta i „wystawiana" na magistralę pod koniec każdego transferu. Urządzenie (dysk twardy) porównuje wtedy otrzymaną wartość z wartością obliczoną w wewnętrznym generatorze i w przypadku wykrycia nieprawidłowości zgłasza błąd, obligujący hosta do ponownego transferu. Wszystko to komplikuje nieco sterowanie buforami trójstanowymi magistrali danych, ponieważ urządzenie szyfrujące musi dokładnie rozpoznać moment w którym mają się pojawić dane z/do hosta, czy też suma kontrolna i odpowiednio ustawić kierunek magistrali. Pomaga w tym opisana maszyna stanów, wypracowująca sygnały, które informują w jakim aktualnie momencie transferu znajdują się urządzenia. Jej graf pracy przedstawiono na **rysunku 7**.



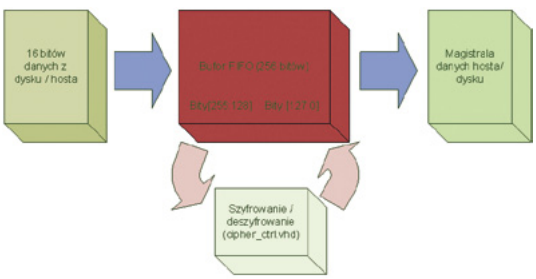
Rysunek 7. Graf pracy automatu stanów detekcji trybu transferu

Stan tegoż automatu jest również jednym z sygnałów wejściowych dla bloku data_bus_ctrl.vhd.

Wyjaśnienia mogą wymagać użyte sygnały dma_wr/dma_rd oraz udma_en. Wypracowywane są one w innej części bloku main_ctrl.vhd. Otóż pierwsze z nich przyjmują wartość 1, kiedy zostanie wykryta komenda transferu w jakimkolwiek trybie bezpośredniego dostępu do pamięci (Read DMA, Write DMA, Read DMA Ext, Write DMA Ext), natomiast druga przyjmuje stan wysoki gdy aktywna jest obsługa UltraDMA przez dysk twardy. Po zakończeniu transferu UDMA, automat zawsze wraca do stanu



Rysunek 8. Graf pracy automatu obsługującego tryb UltraDMA



Rysunek 9. Schemat blokowy idei szyfrowania danych w FIFO

Listing 3. Automat stanów kontroli trybu transmisji (kod źródłowy VHDL).

```
process(hdd_state,dma_wr,dma_rd,dmack_n_last,DMARQ,crc_cnt,udma_en,dmarq_
forced)
begin
  hdd_state_nx<=hdd_state;
  case hdd_state is
    when PIO=>
    if dmack_n_last='0' then
      if dma_wr='1' and udma_en='1' then
        hdd_state_nx<=UDMA_OUT;
      elsif dma_rd='1' and udma_en='1' then
        hdd_state_nx<=UDMA_IN;
      end if;
    end if;
    when UDMA_IN=>
    if dmack_n_last='1' then
    hdd_state_nx<=PIO;
  elsif dmarq_forced='0' then
    hdd_state_nx<=UDMA_CRC;
  end if;
  when UDMA_OUT=>
  if dmack_n_last='1' then
    hdd_state_nx<=PIO;
    elsif DMARQ='0' then
    hdd_state_nx<=UDMA_CRC;
  end if;
  when UDMA_CRC=>
    if crc_cnt=3 or DMARQ='1' then
      hdd_state_nx<=PIO;
    end if;
  end case;
end process;

process(CLK,RESET)
begin
  if RESET='1' then
    hdd_state<=PIO;
    crc_cnt<=0;
  elsif CLK'event and CLK='1' then
    hdd_state<=hdd_state_nx;
    UDMA_CKSUM<='0';
    UDMA_RD<='0';
    UDMA_WR<='0';
    case hdd_state_nx is
    when UDMA_CRC=>
      UDMA_CKSUM<='1';
      if DMACK_N='1' then
        crc_cnt<=crc_cnt+1;
      else
        crc_cnt<=0;
      end if;
      when UDMA_OUT=>
        UDMA_WR<='1';
      when UDMA_IN=>
        UDMA_RD<='1';
      when PIO=>
    end case;
  end if;
end process;

process(fifo_out_state,hdd_state,CS_N,DIOW_N,IORDY,DMARQ,fifo_out_words_
cnt,udma_en,strobe_cnt,redundant_data_cnt,redundant_data_addr)
begin
  fifo_out_state_nx<=fifo_out_state;
  if CS_N="11" then
    case fifo_out_state is
    when IDLE=>
      if hdd_state=UDMA_OUT and DIOW_N='0' then
        fifo_out_state_nx<=FILL;
      end if;
    when FILL=>
      if fifo_out_words_cnt=FIFO_SIZE then
      fifo_out_state_nx<=PASS_THROUGH;
    end if;
    when PASS_THROUGH=>
      if DIOW_N='1' and IORDY='0' then
        fifo_out_state_nx<=EMPTY;
      end if;
      if IORDY='1' and DIOW_N='0' then
        fifo_out_state_nx<=HOLD;
      end if;
      when HOLD=>
        if DIOW_N='0' and IORDY='0' and redundant_data_cnt=redundant_data_addr
then
          fifo_out_state_nx<=PASS_THROUGH;
        end if;
      when EMPTY=>
        if fifo_out_words_cnt=0 and DMARQ='0' then
        fifo_out_state_nx<=IDLE;
      end if;
    end case;
  else
    fifo_out_state_nx<=IDLE;
  end if;
end process;

process(CLK,RESET)
begin
  if RESET='1' then
    fifo_out_data_out<=(others=>'0');
    fifo_out_data_clk<='0';
    fifo_out_state<=IDLE;
    fifo_out_words_cnt<=0;
```

podstawowego – PIO. Ewentualne bezpośrednie przejście ze stanu UDMA_IN lub UDMA_OUT do stanu PIO, pomijające stan UDMA_CRC nie powinno normalnie występować (nie jest ujęte w specyfikacji ATA/ATAPI) – jest jedynie zabezpieczeniem na wypadek nieprzewidzianej sytuacji.

Ze względu na to, że obecnie praktycznie wszystkie dyski korzystają z UDMA podczas transferów, to jedynie ten tryb objęty jest szyfrowaniem/deszyfrowaniem w prezentowanym urządzeniu. Idea opiera się na przepuszczeniu danych płynących z/do hosta poprzez kolejkę FIFO na tyle głęboką, aby możliwe było szyfrowanie/deszyfrowanie danych „w locie". Sterowanie transferami UltraDMA zrealizowane jest również za pomocą maszyn stanów o grafie pracy przedstawionym na **rysunku 8** – osobnej dla transferów wychodzących oraz przychodzących. Poszczególne stany wymagają dokładniejszego wyjaśnienia.

Stan IDLE jest stanem oczekiwania na odpowiednią sekwencję stanów wejść (opisaną w specyfikacji ATA/ATAPI) sygnalizującą rozpoczęcie transferu danych w trybie UltraDMA. Gdy ta zostanie wykryta, automat przechodzi do stanu FILL. Do szyfrowania wykorzystywany jest szyfr blokowy, który wymaga określonej liczby danych (w tym przypadku 128 bitów) do wygenerowania szyfrogramu. Na początku każdego transferu 256-bitowa kolejka FIFO urzadzenia szyfrującego jest pusta. Urządzenie szyfrujące musi więc wypełnić bufor i odebrać od urządzenia nadającego pierwszych 16 słów danych, jednocześnie informując urządzenie odbierające (blokując przenoszenie sygnału taktującego), że transfer jeszcze nie został rozpoczęty. Stan PASS_THROUGH jest stanem, w którym sygnały taktujące i sterujące transferem są przenoszone wprost między urządzeniami, podmieniane są jedynie sygnały na liniach danych, na te z wyjścia kolejki FIFO (zaszyfrowane w przypadku transferu host–>dysk lub zdeszyfrowane w sytuacji przeciwnej). Za

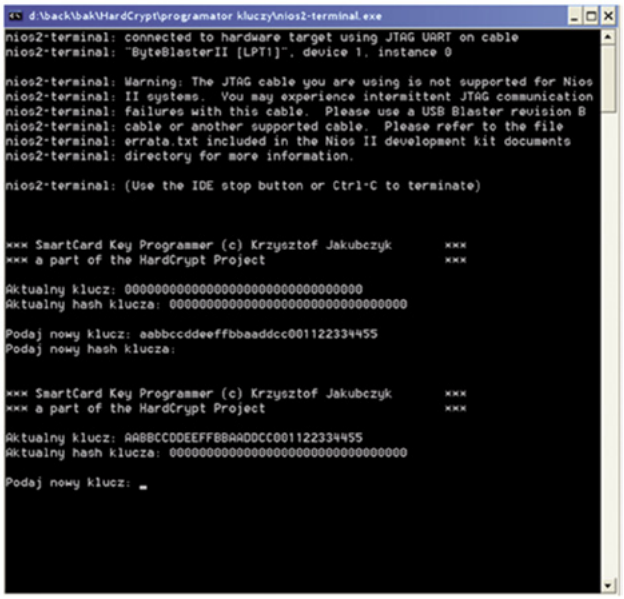
szyfrowanie/deszyfrowanie danych z bufora odpowiada blok *cipher_ctrl.vhd* który zostanie opisany w dalszej części. Specyfikacja ATA/ATAPI przewiduje dodatkowo sytuację chwilowego wstrzymania transferu przez hosta lub dysk twardy – za obsługę tego wyjątku odpowiedzialny jest stan HOLD naszej maszyny stanów. Tu pojawia się pewna komplikacja, którą przedstawię na przykładzie. Załóżmy, że jesteśmy w trakcie transferu danych z dysku twardego. Nagle z pewnych przyczyn kontroler IDE po stronie hosta potrzebuje na chwilę wstrzymać transmisję, więc generuje odpowiednią sekwencję na liniach sterujących magistralą. Pamiętamy jednocześnie, że w przypadku trybów UltraDMA, to urządzenie wysyłające dane generuje również sygnał taktujący. W takim wypadku istnieje prawdopodobieństwo, że zanim dysk twardy „zauważy" żądanie wstrzymania transferu, zdąży jeszcze wysłać kilka kolejnych słów danych. Dane te nie mogą zostać zgubione, stąd w stanie HOLD urządzenie szyfrujące zapisuje te nadmiarowe dane do wewnętrznej pamięci nazwanej *redundant_host_data*. Gdy transfer zostanie wznowiony, najpierw musimy wysłać „zaległe" dane z tej pamięci, a dopiero później szyfrator może wrócić do stanu PASS_THROUGH. Kolejnym stanem maszyny stanów jest stan EMPTY, który w przypadku zakończenia transmisji danych odpowiada za opróżnienie kolejki FIFO. Urządzenie nadające przestaje generować sygnał taktujący, ponieważ wysłało już wszystkie dane, szyfrator natomiast musi jeszcze opróżnić FIFO, generując sygnał taktujący dla tych danych. Należy zwrócić uwagę, że istnienie kolejki FIFO na drodze transmisji danych, generuje de facto niezauważalną latencję danych odpowiadającą czasowi wstępnego jej wypełnienia w stanie FILL. Po wszystkim automat wraca do stanu wyjściowego. Dla zainteresowanych kod VHDL znajduje się na **listingu 4**.

Z bloku main_ctrl.vhd przekazywana jest również informacja na temat aktualnie zapisywanego lub odczytywanego sektora. Informacja ta jest bezpośrednio związana z zastosowanym trybem szyfrowania. Ustaliliśmy wcześniej, że będziemy szyfrować z zastosowaniem symetrycznego szyfru blokowego. Co to oznacza w praktyce? Otóż gdybyśmy nie zastosowali jakiegoś trybu wiązania danych oznaczałoby to tyle, że szyfrogramy jednakowych bloków danych byłyby identyczne w całym obszarze dysku, co stanowiłoby poważną lukę w bezpieczeństwie i czyniłoby nasze urządzenie podatnym np. na ataki typu watermarking. Nie sposób omówić wszystkich drobnych funkcji jakie pełnią wszystkie bloki układu, dlatego ciekawych Czytelników zapraszam do zapoznania się z całością kodu źródłowego umieszczonego na płycie CD.

Listing 3. c.d.

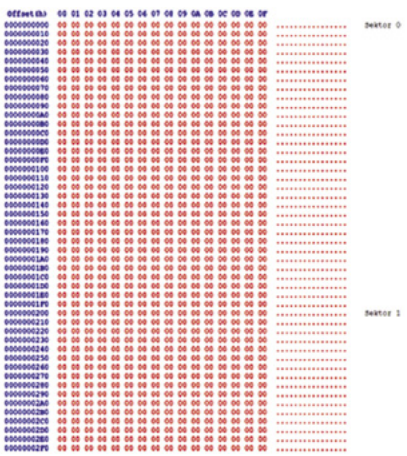
```
    strobe_cnt<=0;
    iordy_lock<='0';
    elsif CLK'event and CLK='1' then
    fifo_out_state<=fifo_out_state_nx;
    fifo_out_data_clk<='0';
    diow_n_forced<=DIOW_N;
    strobe_cnt<=0;
    iordy_lock<='0';
    redundant_data_cnt<=0;
    redundant_data_addr<=0;
    case fifo_out_state_nx is
      when IDLE=>
        dior_n_forced<=DIOR_N;
        fifo_out_words_cnt<=0;
        strobe_cnt<=0;
      when FILL=>
        dior_n_forced<='1';
        if dior_n_last(1)=not DIOR_N then
          if DIOR_N='1' then
            fifo_out_data_out<=data_from_host_rising;
          else
            fifo_out_data_out<=data_from_host_falling;
          end if;
          fifo_out_words_cnt<=fifo_out_words_cnt+1;
          fifo_out_data_clk<='1';
        end if;
      when PASS_THROUGH=>
        if dior_n_last(1)=not DIOR_N then
          dior_n_forced<=not dior_n_forced;
          if DIOR_N='1' then
            fifo_out_data_out<=data_from_host_rising;
          else
            fifo_out_data_out<=data_from_host_falling;
          end if;
          fifo_out_data_clk<='1';
        end if;
      when HOLD=>
        if DIOW_N='1' then
          dior_n_forced<='1';
        end if;
        redundant_data_cnt<=redundant_data_cnt;
        redundant_data_addr<=redundant_data_addr;
        if dior_n_last(1)=not DIOR_N and DIOW_N='0' then
          redundant_data_cnt<=redundant_data_cnt+1;
          if DIOR_N='1' then
            redundant_host_data(redundant_data_cnt)<=data_from_host_rising;
          else
            redundant_host_data(redundant_data_cnt)<=data_from_host_falling;
          end if;
        end if;
        if DIOW_N='0' and HDD_IORDY='0' then
          iordy_lock<='1';
          if strobe_cnt/=T_CYC then
            strobe_cnt<=strobe_cnt+1;
          else
            dior_n_forced<=not dior_n_forced;
            redundant_data_addr<=redundant_data_addr+1;
            fifo_out_data_out<=redundant_host_data(redundant_data_addr);
            fifo_out_data_clk<='1';
            strobe_cnt<=0;
          end if;
        end if;
      when EMPTY=>
        diow_n_forced<='0';
        if fifo_out_words_cnt/=0 then
          if strobe_cnt/=T_CYC then
            strobe_cnt<=strobe_cnt+1;
          else
            dior_n_forced<=not dior_n_forced;
            fifo_out_words_cnt<=fifo_out_words_cnt-1;
            fifo_out_data_clk<='1';
            strobe_cnt<=0;
          end if;
        end if;
    end case;
  end if;
end process;
```

cipher_ctrl.vhd. Blok ten realizuje mechanizm kontroli równoległego odczytu i zapisu kolejek FIFO (wejściowej i wyjściowej) kierując odpowiednie dane do bloków szyfrujących cipher_core.vhd i wypracowane przez nie dane z powrotem do kolejek FIFO. Zapis i odczyt do buforów musi następować w ściśle określonym czasie, dlatego zaimplementowany został licznik danych wchodzących do FIFO i na jego podstawie synchronizowany jest zapis. Schemat blokowy przedstawia rysunek 9. Opis sprzętu składa się z trzech liczników. Pierwszy zlicza kolejne 16-bitowe słowa wchodzące do kolejki FIFO. Drugi inkrementuje się wraz z każdym 128-bitowym (bo takiej długości bloki przyjmuje na swoje wejście blok szyfrujący) blokiem, który został zapisany w FIFO. Trzeci zaś jest licznikiem sektorów. Sektor dysku twardego ma 512 bajtów. Podczas transferu DMA, adresowany przez hosta jest pierwszy sektor, z którym będzie zachodzić wymiana danych. Podczas samego procesu wymiany, odczytywana bądź zapisywana jest z reguły większa liczba

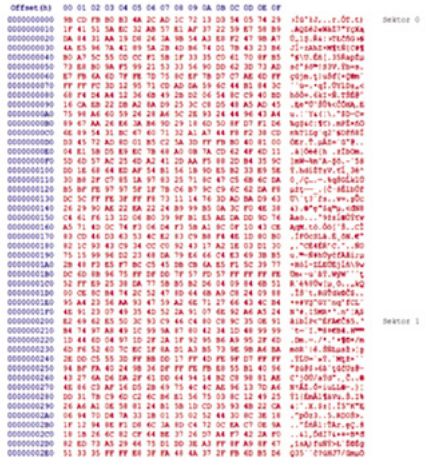


Rysunek 10. Widok ekranu programatora kluczy dla kart chipowych z konsoli NIOS2

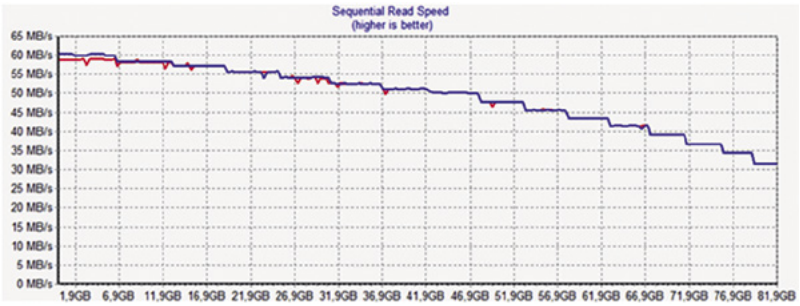
sektorów (do 65535). Jak pisałem wcześniej, żeby dane bezpiecznie zaszyfrować, trzeba kolejne szyfrowane bloki z sobą powiązać. Ponieważ blok CIPHER_CTRL realizuje tryb szyfrowania zależny od bezwzględnego numeru sektora dysku, wymagane jest sumowanie wewnętrznego licznika sektorów z adresem LBA, podanym przez hosta na początku transmisji. Po każdym naliczonym 128-bitowym bloku w FIFO, blok ten (128 najbardziej znaczących bitów kolejki) wysyłany jest do bloku szyfrującego. Tryb szyfrowania jest trybem CBC z dodatkowym zabezpieczeniem w postaci związania danych z numerem sektora dysku operacją logiczną XOR, dzięki czemu te same dane zapisane w różnych sektorach mają różne szyfrogramy.



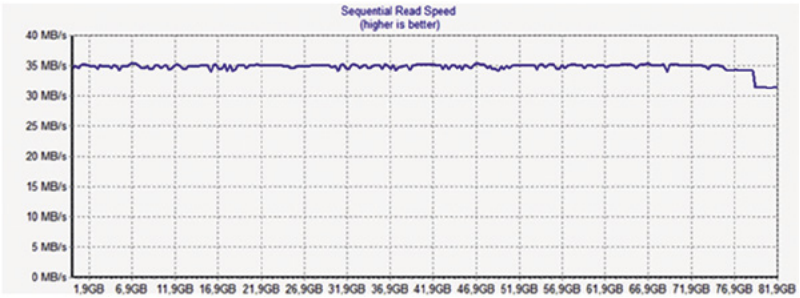
Rysunek 11. Przykładowe dane zapisane z użyciem urządzenia szyfrującego



Rysunek 12. Postać zaszyfrowana danych z rysunku 11



Rysunek 13. Test prędkości transferu dla dysku twardego Maxtor 80 GB wpiętego poprzez szyfrator (wykres czerwony) oraz wpiętego bezpośrednio (wykres niebieski)



Rysunek 14. Test prędkości transferu dysku twardego Maxtor 80 GB podłączonego poprzez przejściówkę USB⇔IDE

sd_ctrl.vhd. Jest to blok kontrolera karty pamięci SD oraz odczytu klucza szyfrującego. Składa się z trzech automatów stanów. Jeden z nich odpowiada za odpowiednie formowanie komend oraz ich parametrów i steruje drugim – niskopoziomowym automatem wysyłającym oraz odbierającym bloki danych poprzez SPI. Automat odpowiadający za formowanie komend, sterowany jest poprzez trzeci, którego kolejne stany są de facto sekwencją komend do wykonania na karcie pamięci. W praktyce realizowana jest prosta sekwencja poszukiwania pliku SECURITY.KEY w głównym katalogu systemu plików FAT16 i w przypadku jego znalezienia, zapisanie odpowiednich rejestrów oraz ustawienie sygnału GOT_KEY.

sle5528.v. Blok dopisany podczas dalszego rozwijania urządzenia (w prototypie go nie było). W międzyczasie autor docenił zalety języka Verilog w stosunku do VHDL i to w tym właśnie języku opisany został automat pozwalający na odczyt oraz zapis bajtów z popularnej karty chipowej SLE5528. Do przełączania pomiędzy kluczami z karty SD a karty chipowej służy multiplexer *cardmux.v*, który w zależności od tego jaką kartę wykryje, wystawia odpowiedni klucz oraz sygnał gotowości dla opisanego wcześniej bloku cipher_ctrl.

W urządzeniu występują również mniejsze bloki:

sfl – komponent Altery umożliwiający pośrednie programowanie pamięci konfiguracyjnej układu (EPCS) poprzez JTAG.

crc_gen – bloki generatora crc, obliczające sumę kontrolną CRC16 zgodną ze

specyfikacją ATA/ATAPI dla danych zaszyfrowanych oraz zdeszyfrowanych.

pll – komponent Altery sterujący wbudowanymi w układ blokami PLL. Generuje sygnały zegarowe taktujące poszczególne bloki.

Programowanie kluczy

Jeżeli zdecydujemy, że chcemy używać karty SD jako klucza, to wystarczy sformatować kartę stosując system plików FAT16 i zapisać w głównym katalogu plik SECURITY.KEY. Żeby ułatwić cały proces i ograniczyć stosowanie słownikowych haseł, autor napisał prosty program do generacji tego pliku. Program ten prosi użytkownika o podanie hasła, na podstawie którego generowany jest 128-bitowy skrót (*hash*) według algorytmu MD5, który stanowić będzie właściwy klucz szyfrujący. Dodatkowo generowany jest również skrót tego klucza, który posłuży do generacji wektorów inicjalizacyjnych na potrzeby trybu szyfrowania zastosowanego w urządzeniu.

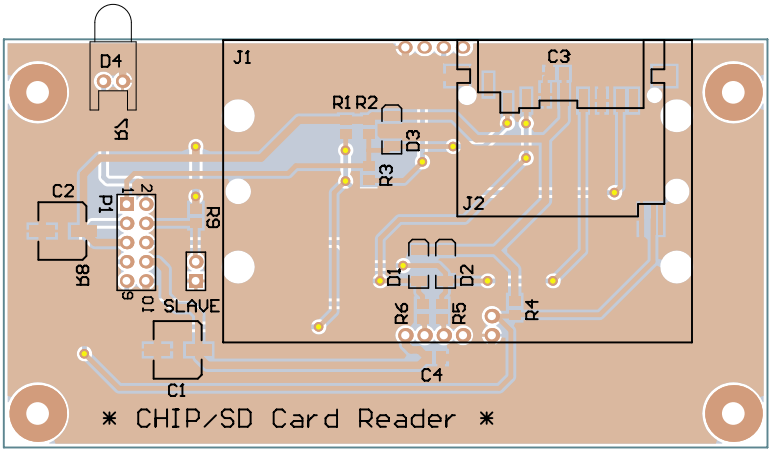
W przypadku karty chipowej mamy do wyboru dwie możliwości. Można na własną rękę programatorem zapisać pod adres 32: 256-bitowy klucz oraz sygnaturę informującą o tym, że jest to karta-klucz dla naszego urządzenia (0x68,0x63). Druga możliwość jest dla Czytelników nieposiadających specjalizowanych programatorów. Wśród materiałów do artykułu znajduje się projekt, wykorzystujący soft-core'owy procesor NIOS2 firmy Altera. Do procesora podpięty został poprzez porty GPIO, opisany wcześniej blok kontrolera karty *sle5528.v*. Napisany został również prosty program dla tego procesora obsługujący te karty. Dzięki temu, po zaprogramowaniu naszego szyfratora plikiem wynikowym tego projektu (*sle5528.sof*), nasze urządzenie zamienia się w programator kluczy. Wystarczy teraz uruchomić *nios2-terminal* (w pakiecie Altera Quartus) będąc połączonym z urządzeniem poprzez interfejs JTAG oraz wsunąć kartę chipową. Od tego momentu zostaniemy „poprowadzeni za rękę" przez proces zapisu klucza (**rysunek 10**).

Działanie praktyczne

W celu zobrazowania procesu szyfrowania, zapisano na pierwszych dwóch sektorach dysku przykładowe dane w postaci ciągu samych zer (**rysunek 11**).

Próba odczytania uprzednio zapisanych danych bez użycia urządzenia szyfrującego z właściwym kluczem przedstawiona została na **rysunku 12**. Widać, że pomimo zapisania ciągu tych samych danych, postać zaszyfrowana różni się w obu sektorach i sprawia wrażenie pseudolosowej. Cel osiągnięty.

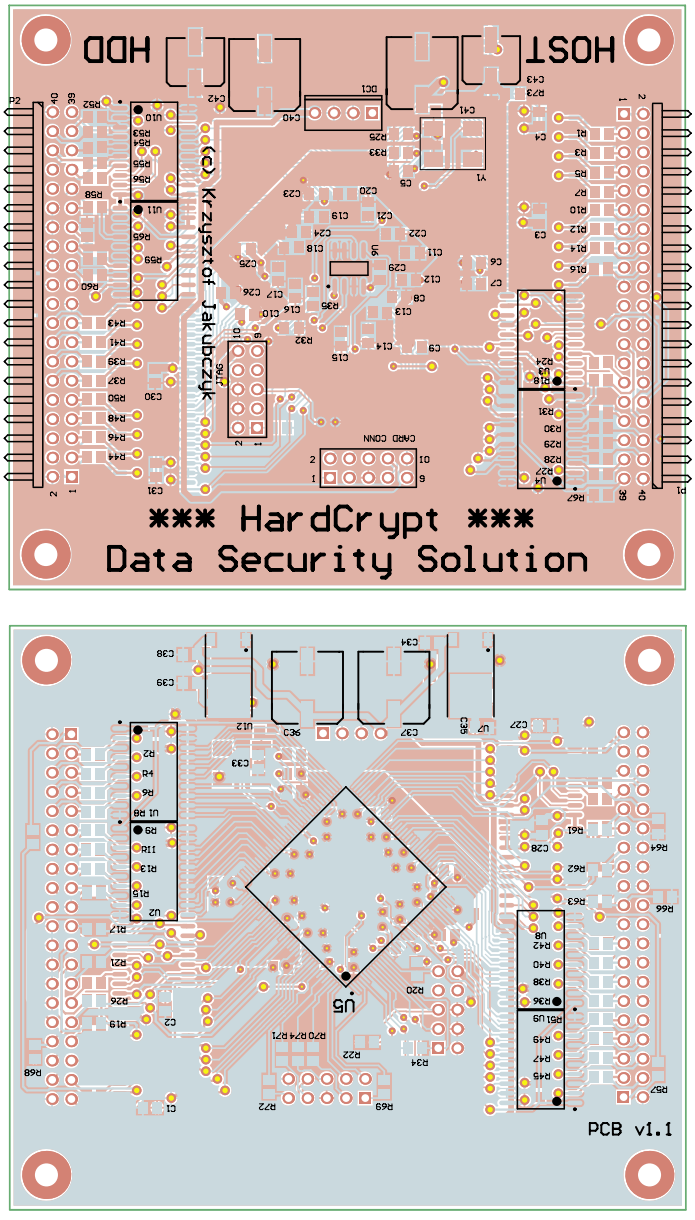
A jak wpływa nasze ograniczenie do UltraDMA 66 na transfery? W praktyce bardzo wiele dysków pomimo wspierania trybów



Rysunek 15. Schemat montażowy czytnika kart

UltraDMA 133 ma trudności z osiągnięciem prędkości transmisji większych niż 66 MB/s. Przykładem jest testowany dysk twardy o pojemności 80 GB firmy Maxtor. Obsługuje on UDMA 133, natomiast w praktycznym teście praktycznie nie widać różnicy pomiędzy szybkościami transferów osiąganymi przy wpięciu poprzez urządzenie szyfrujące, a bez niego (**rysunek 13**).

Urządzenie szyfrujące zostało też przetestowane przy użyciu przejściówki USB⇔IDE. Wykresy w tym przypadku ide-



Rysunek 16. Schemat montażowy szyfratora

http://forum.ep.com.pl

alnie się pokrywały ze względu na widoczne ograniczenie portu USB do około 35 MB/s (**rysunek 14**).

Montaż i uruchomienie układu

Schemat montażowy szyfratora pokazano na **rysunku 15**, natomiast czytnika kart na **rysunku 16**. Montaż rozpoczynamy standardowo od elementów SMD, kończąc na przewlekanych. Przed montażem złącz P1 i P2 należy usunąć (przez wyciągnięcie) piny o numerze 20, ponieważ na taśmie dysku twardego pin ten jest zaślepiony (ma to na celu utrudnienie odwrotnego włożenia). W miejscu układu U5 (FPGA) można zastosować EP2C8 lub EP2C5, w zależności czy chcemy dokonywać dalszych modyfikacji w kodzie. Jeżeli nie chcemy niczego zmieniać, to wystarczy układ o mniejszych zasobach – EP2C5. To samo dotyczy układu U6 – konfiguratora FPGA. Można zastosować albo wersję tańszą EPCS1 (w przypadku użycia EP2C5), albo droższą EPCS4 (dla EP2C8).

Po zmontowaniu i sprawdzeniu urządzenia można zabrać się za uruchamianie. W materiałach do artykułu znajduje się kompletny, syntezowalny projekt (wraz z kodami źródłowymi) wykonany w środowisku Altera Quartus 9. W celu zaprogramowania układu, podłączamy do złącza DC1, czteropinową wtyczkę typu molex z zasilacza komputera (taka sama jak do zasilania stacji dyskietek 3.5"). Można urządzenie również zasilić z zewnętrznego zasilacza 5 V. Do programowania używamy złącza JTAG oraz interfejsu zgodnego z Altera ByteBlaster II. Autor wykorzystał tego typu interfejs, zbudowany w oparciu o schematy dostępne w Internecie, podłączany poprzez port równoległy LPT. Programując układ wprost plikiem wynikowym .sof, skonfigurujemy FPGA jednorazowo – tzn. do czasu zaniku napięcia zasilania. FPGA powinno być konfigurowane automatycznie po starcie, w tym celu należy zaprogramować układ EPCS. Aby dostać się do pamięci konfiguracyjnej poprzez JTAG, zastosowany został specjalny komponent firmy Altera, o którym pisałem wcześniej. Wymaga to jednak dodatkowej czynności – przekonwertowania pliku wynikowego .sof, do pliku .jic (JTAG Indirect Configuration File). W tym celu z menu środowiska Quartus wybieramy „File–>Convert Programming Files". W oknie, które otworzy się należy wybrać odpowiedni układ konfiguracyjny w zależności od wybranych układów U5 i U6. Po wszystkim klikamy przycisk „Generate".

Po wygenerowaniu pliku .jic można przejść do programowania. Z menu środowiska Altera Quartus należy wybrać „Tools–>Programmer". Za pomocą przycisku „Hardware Setup" wybieramy właściwy typ posiadanego programatora. Jako plik źródłowy do programowania używamy wygenerowanego wcześniej pliku z rozszerzeniem .jic i klikamy „start". Po kilkunastu sekundach układ jest gotowy do pracy.

Pozostaje nam sprawdzenie działania. Złącze CARD_CONN głównego modułu szyfrującego łączymy za pomocą taśmy AWG10 ze złączem P1 modułu czytnika kart. Obwód drukowany modułu czytnika kart jest przewidziany do montażu w typowej ramce 5,25", przeznaczonej do mocowania w niej stacji dyskietek 3,5". Ramkę taką można nabyć np. na popularnym internetowym serwisie aukcyjnym. Do złącza P1 szyfratora podpinamy 40 bądź 80-żyłową taśmę od kontrolera IDE z płyty głównej, natomiast do złącza P2 poprzez kolejną taśmę podłączamy szyfrowany dysk twardy. Po włączeniu zasilania, układ sygnalizuje swoje działanie za pomocą diody świecącej umieszczonej w module czytnika kart. Jeżeli wsuniemy kartę chipową lub SD zawierającą klucz szyfrujący, dioda zapali się, co oznaczać będzie że układ wykorzystuje pobrane klucze do szyfrowania/deszyfrowania danych. Kiedy dioda jest wygaszona, oznacza to stan w którymy układ nie odnalazł żadnych kluczy szyfrujących i przepuszcza dane wprost (bez szyfrowania). Dodatkowo każda aktywność dysku twardego (przesył danych) objawia się miganiem diody. W razie dodatkowych i zasadnych pytań, autor chętnie odpowie na nie drogą mailową.

Artykuł powstał jako rozwinięcie pracy magisterskiej napisanej pod kierunkiem dr inż. Marcina Kucharczyka. Autor jest absolwentem kierunku Elektronika i Telekomunikacja, wydziału Automatyki Elektroniki i Informatyki Politechniki Śląskiej w Gliwicach.

Krzysztof Jakubczyk
krzysztof.jakubczyk@gmail.com