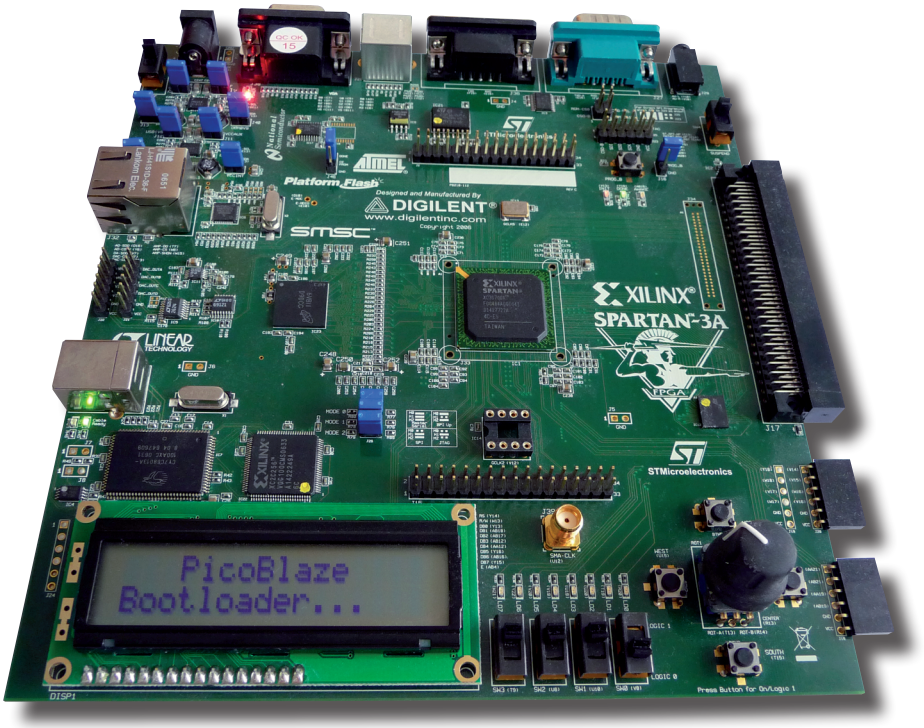
Sprzętowy bootloader dla mikrokontrolerów PicoBlaze (2)

Podczas uruchamiania i testowania oprogramowania dla mikrokontrolerów osadzonych w FPGA pewien kłopot stanowi sposób wymiany kodu programu, który zazwyczaj wymaga ponownej kompilacji całego projektu z mikrokontrolerem i rekonfiguracji układu FPGA. Kompilacja nawet stosunkowo niewielkich projektów jest procesem zajmującym relatywnie dość dużo czasu, przez co częsta wymiana oprogramowania dla osadzonego mikrokontrolera staje się niewygodna i nieefektywna. Kontynuujemy opis sposobu rozwiązania tego problemu dla mikrokontrolerów PicoBlaze.



Nadajnik i odbiornik UART z buforem FIFO

Podczas obsługi transmisji szeregowej często stosowane jest buforowanie wysyłanych i odbieranych danych. Buforowanie takie ułatwia obsługę portu UART, umożliwiając bardziej elastyczną (w sensie reżimu czasowego) komunikację z portem przez układy nadrzędne, np. mikrokontroler. Typowym buforem wykorzystywanym w takich zastosowaniach jest bufor FIFO (*First In First Out*). Bufor taki można sobie wyobrazić jako dwuportową pamięć RAM wraz z odpowiednim układem sterującym. Za pomocą jednego portu dokonuje się zapisu nadchodzących danych. Drugi port wykorzystywany jest do niezależnego odczytu tych danych, w kolejności w jakiej zostały zapisane (pierwszy zapisany bajt również odczytywany jest jako pierwszy).

Nadajnik UART z buforem FIFO wykorzystywany jest (tworzona jest jego instancja) przez wirtualny komponent sprzętowego *bootloadera*. Z kolei wirtualne komponenty nadajnika i odbiornika UART wyposażone w tego typu bufor zastosujemy dla, omawianej w dalszej części tekstu, przykładowej aplikacji z mikrokontrolerem pBlazeZH oraz sprzętowym *bootloaderem*.

W **tab. 4** przedstawiono opis interfejsu wirtualnego komponentu nadajnika UART z buforem FIFO. **Rys. 6** z kolei ilustruje przebiegi czasowe podczas zapisu danych do bufora dla tego wirtualnego komponentu. Możliwy jest zapis pojedynczego bajtu, poprzez uaktywnienie na jeden takt zegara sygnału *WR_EN*, albo zapis pakietowy – gdy sygnał *WR_EN* utrzymywany jest w stanie wysokim przez więcej niż jeden takt zegara. Dane zapisywane są do bufora podczas narastającego zbocza zegara *CLK_WR*, gdy *WR_EN* znajduje się w stanie wysokim.

Dzięki wykorzystaniu bufora FIFO możliwy jest zapis ciągu bajtów do wysłania za pomocą portu szeregowego bez oczekiwania na zakończenie transmisji poprzednich danych. Nie zachodzi też potrzeba synchronizacji odpowiednich sygnałów sterujących z sygnałem taktującym nadajnik (*CLK_TX*), gdyż wprowadzony został drugi sygnał zegarowy przeznaczony do zapisu danych do bufora (*CLK_WR*).

W **tab. 5** przedstawiono opis interfejsu kolejnego wirtualnego komponentu wyposażonego w bufor FIFO – odbiornika UART. **Rys. 7** ilustruje odpowiednie przebiegi czasowe. Gdy dane zostaną odebrane przez odbiornik UART, aktywowany jest sygnał *DATA_RDY*. Jednocześnie na wyjściu *DATA* pojawia się (2 takty zegara *CLK_RD* wcześniej niż sygnał *DATA_RDY*) pierwszy z odebranych bajtów. Potwierdzenie odczytu tego bajtu wymaga ustawienia na jeden takt zegara *CLK_RD* sygnału *RD_EN*. Podczas narastającego zbocza *CLK_RD*, gdy *RD_EN* znajduje się w stanie wysokim, następuje wyprowadzenie na wyjście *DATA* kolejnego odebranego bajtu danych, pod warunkiem że sygnał *DATA_RDY* pozostanie w stanie aktywnym (w buforze znajduje się kolejny odebrany bajt). Gdy w buforze nie ma już nowych (nieodczytanych) danych, wraz z narastającym zboczem zegara wycofywany jest sygnał *DATA_RDY*.

Projektując wirtualne komponenty nadajnika oraz odbiornika UART z buforem FIFO, nie wprowadzono żadnych mechanizmów kontroli przepełnienia bufora. Należy więc zadbać o to, by za pomocą parametru *DEPTH* dobrać odpowiednią pojemność bufora dostosowaną do wielkości przesyłanych danych. Warto również

zwrócić uwagę na to, że omawiane wirtualne komponenty mają strukturę hierarchiczną. Z wnętrza komponentu (modułu) nadajnika z buforem FIFO dokonywane jest tworzenie instancji, omawianego powyżej, modułu nadajnika UART (*SERIAL_TX*), zaś z wnętrza komponentu odbiornika z buforem FIFO tworzona jest instancja modułu odbiornika (*SERIAL_RX*).

Wirtualny komponent *bootloadera*

W **tab. 6** pokazano opis interfejsu sprzętowego *bootloadera*. Oprócz typowych portów wejścia-wyjścia, wirtualny komponent *bootloadera* ma jeszcze trzy parametry (pewne stałe, których wartość określa się w kodzie źródłowym podczas tworzenia instancji komponentu). Pierwsze dwa parametry reprezentują współczynniki o tych samych nazwach obecne w równaniu (1) definiującym częstotliwość wyjściową syntezera DDFS. Współczynniki te należy dobrać w taki sposób, aby zależnie od częstotliwości wejściowej, uzyskać (z pewną dokładnością) częstotliwość wyjściową 16 razy większą niż częstotliwość nominalna portu UART, określająca wymaganą prędkość transmisji.

Parametr *DB* definiuje z kolei liczbę bitów sprzętowego licznika, którego przepełnienie, podczas niepowodzenia transmisji, określa opóźnienie wycofania sygnału *RST_OUT* służącego do resetowania mikrokontrolera. Licznik ten jest taktowany sygnałem zegarowym *CLK* i zostaje wyzerowany za każdym razem, gdy portem szeregowym przesyłany jest poprawny znak.

Spośród portów wejścia-wyjścia wirtualnego komponentu *bootloadera* na nieco większą uwagę zasługują sygnały *TX_INT*, *RX_INT*, *ON* oraz wspomniany już *RST_OUT*. Pierwsze dwa z nich reprezentują kolejno wejście i wyjście dla zewnętrznego (w odniesieniu do *bootloadera*) portu szeregowego, który może być obecny w systemie wraz z mikrokontrolerem. *Bootloader* dysponuje własnym portem UART i korzysta z zewnętrznych linii danych *RX* oraz *TX*, ale tylko w momencie określonym wysokim stanem na wejściu *ON*. W czasie kiedy wejście *ON* znajduje się w stanie niskim, zewnętrzne sygnały linii danych *RX* i *TX* dołączane są za pomocą multipleksera do sygnałów *RX_INT* i *TX_INT*. Podczas aktywności *bootloadera* (wejście *ON* w stanie wysokim), na wyjściu *RX_INT* utrzymywany jest poziom wysoki. Narastające zbocze na wejściu *ON* powoduje dodatkowo wysłanie, za pośrednictwem portu UART, komunikatu informującego o aktywności modułu *bootloadera*. Warto tutaj dodać, że wejście *ON* zostało wyposażone w układ eliminacji mechanicznych drgań zestyków i może być dołączone bezpośrednio np. do dwupołożeniowego przełącznika, który służyłby jako element wyznaczający stan aktywności modułu *bootloadera*.

Sygnał wyjściowy *RST_OUT* może służyć do resetowania mikrokontrolera w czasie ładowania kodu do pamięci programu. Sygnał ten znajduje się w stanie wysokim (oprócz przypadku, gdy aktywny jest sygnał *RST*) od momentu rozpoczęcia transmisji pliku z kodem programu do momentu gdy transmisja zostanie zakończona. Poprawne zakończenie transmisji, sygnalizowane odpowiednim komunikatem, następuje po przesłaniu dokładnie 1024 słów kodu programu. W przypadku przerwania transmisji przed przesłaniem 1024 kodów rozkazów, sygnał *RST_OUT* pozostaje jeszcze w stanie wysokim przez pewien czas i dopiero po jego upływie przechodzi do stanu niskiego. Czas ten określony jest przez wartość opisanego wyżej parametru *DB*.

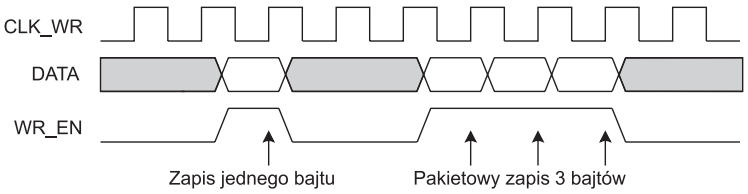
Rys. 8 przedstawia przykładowy widok okna programu *HyperTerminal* podczas komunikacji z *bootloaderem*. Ilustruje jednocześnie teksty komunikatów zwrotnych generowanych przez *bootloader*. Po przełączeniu w stan wysoki wejścia *ON* do *HyperTerminalu* wysyłany jest komunikat powitalny (pierwsza linia tekstu). Następnie wybierając z menu *Transfer*

Tab. 4. Opis parametrów i portów wirtualnego komponentu nadajnika UART z buforem FIFO

moduł: SERIAL_TX_FIFO

Parametr	Opis
DEPTH	Pojemność bufora FIFO określona zależnością 2^{DEPTH} słów 8-bitowych

Port	Kierunek	Liczba bitów	Opis
CLK_TX	wejście	1	Sygnał taktujący nadajnik (częstotliwość nominalna)
RST	wejście	1	Sygnał resetujący. Aktywny poziom niski
WR_EN	wejście	1	Sygnał zezwolenia na zapis danych z wejścia DATA do bufora FIFO. Zapis następuje podczas narastającego zbocza sygnału CLK_WR
CLK_WR	wejście	1	Sygnał zegarowy zapisu do bufora FIFO
DATA	wejście	8	Dane wejściowe
TX	wyjście	1	Szeregowe wyjście danych nadajnika



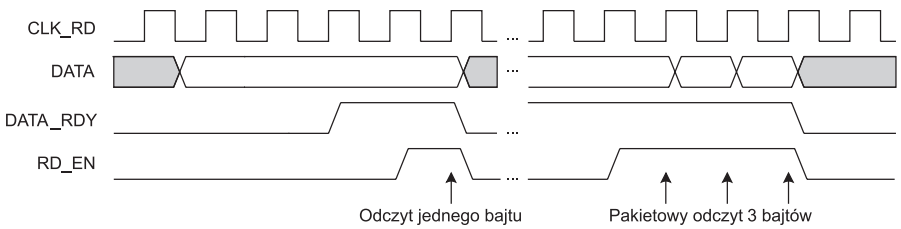
Rys. 6. Przebiegi czasowe podczas zapisu danych dla wirtualnego komponentu nadajnika UART z buforem FIFO

Tab. 5. Opis parametrów i portów wirtualnego komponentu odbiornika UART z buforem FIFO

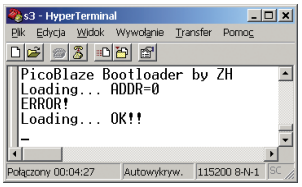
moduł: SERIAL_RX_FIFO

Parametr	Opis
DEPTH	Pojemność bufora FIFO określona zależnością 2^{DEPTH} bajtów

Port	Kierunek	Liczba bitów	Opis
CLK_RX	wejście	1	Sygnał taktujący odbiornik transmisji szeregowej (16× częstotliwość nominalna)
RST	wejście	1	Sygnał resetujący. Aktywny poziom niski
DATA_RDY	wyjście	1	Poziom wysoki na tym wyjściu oznacza, że w buforze FIFO dostępne są dane odebrane przez odbiornik transmisji szeregowej
RD_EN	wejście	1	Sygnał zezwolenia na odczyt danych z bufora FIFO. Aktywny poziom wysoki. Odczytane dane pojawiają się na wyjściu DATA podczas narastającego zbocza sygnału CLK_RD
CLK_RD			Sygnał taktujący odczytu z bufora FIFO
DATA	wyjście	8	Dane wyjściowe
RX	wejście	1	Szeregowe wejście danych odbiornika UART



Rys. 7. Przebiegi czasowe podczas odczytu danych dla wirtualnego komponentu odbiornika UART z buforem FIFO



Rys. 8. Widok okna programu HyperTerminal komunikującego się z *bootloaderem*

opcję *Wyślij plik tekstowy*, możemy przesłać do *bootloadera* kod programu zawarty w pliku tekstowym (z rozszerzeniem HEX), będącym produktem działania asemblera KCPSM3. Rozpoczęcie transmisji sygnalizowane jest komunikatem „Loading…" a jej poprawne zakończenie tekstem „OK!!".

Program *HyperTerminal* podczas swojej pracy pozwala na wytransmitowanie kodów ASCII odpowiadających naciskanym klawiszom na klawiaturze PC. Naciskanie większości klawiszy podczas transmisji może zakończyć się komunikatem zwrotnym o błędzie („ERROR!"). Wyjątek stanowią tu znaki o kodach '0'…'9' oraz 'A'…'Z', odpowiadające szesnastkowej reprezentacji liczb, a także znaki specjalne, takie jak CR, LF, BS. Szczególną rolę odgrywa jednak klawisz „Backspace", którego wciśnięcie powoduje wyzerowanie zespołu rejestrów sterujących adresem zapisu do pamięci programu (adres zapisu ustawiony na wartość początkową 0). Zdarzenie takie sygnalizowane jest komunikatem „ADDR=0". Ta opcja jest przydatna, gdy chcemy wznowić transmisję po wcześniejszym błędzie lub gdy poprzedni transfer nie został zakończony (nie przesłano wszystkich 1024 słów). Innym sposobem wyzerowania adresu jest wymuszenie poziomu niskiego na wejściu *ON* i ponowne wprowadzenie tego wejścia w stan wysoki.

Wykorzystanie aplikacji *HyperTerminal*, a także wybranego pliku generowanego przez asembler KCPSM3 nie pozwala na wprowadzenie skutecznych mechanizmów kontroli poprawności przesyłanych danych do *bootloadera*. Sytuacją, która powoduje zgłoszenie błędu przez *bootloader*, jest odebranie znaku o kodzie spoza przedziału odpowiadającego pojedynczej cyfrze reprezentowanej w kodzie heksadecymalnym. Niemniej jednak, jak wynika z doświadczeń autora, w typowym środowisku pracy przy właściwie dobranych parametrach syntezera częstotliwości, rzeczywiste błędy transmisji danych praktycznie się nie zdarzają.

Przykładowa aplikacja

Zilustrujemy teraz sposób zastosowania *bootloadera* w przykładowym projekcie z mikrokontrolerem PicoBlaze (a dokładnie wersją pBlazeZH). Wykorzystamy tutaj również opisane wcześniej wirtualne komponenty portu szeregowego. Zadaniem aplikacji będzie przepisywanie znaków transmitowanych portem szeregowym z wejścia na wyjście (odebranie znaku i ponowne jego wysłanie portem szeregowym)

Tab. 6. Opis parametrów i portów wirtualnego komponentu sprzętowego bootloadera

moduł: PBLAZE_DOWNLOADER

Parametr	Opis		
K, N	Parametry syntezera częstotliwości według zależności (1)		
DB	Liczba bitów licznika, którego przepełnienie określa opóźnienie wycofania sygnału RST_OUT przy niepowodzeniu transmisji		
Port	**Kierunek**	**Liczba bitów**	**Opis**
CLK	wejście	1	Sygnał zegarowy
RST	wejście	1	Sygnał resetujący. Aktywny poziom niski
ON	wejście	1	Poziom wysoki na tym wejściu aktywuje działanie modułu – dane przesyłane portem szeregowym są interpretowane przez bootloader. Poziom niski oznacza odłączenie portu szeregowego (sygnałów RX i TX) od bootloadera i dołączenie go do linii RX_INT oraz TX_INT
TX_INT	wejście	1	Do tego wejścia należy dołączyć sygnał TX zewnętrznego nadajnika transmisji szeregowej (jeśli taki jest obecny w systemie)
RX	wejście		Wejście danych portu szeregowego
RX_INT	wyjście	1	Do tego wyjścia należy dołączyć sygnał RX zewnętrznego odbiornika transmisji szeregowej (jeśli taki jest obecny w systemie)
TX	wyjście	1	Wyjście danych portu szeregowego
RST_OUT	wyjście	1	Poziom wysoki na tym wyjściu oznacza, że portem szeregowym transmitowane są dane lub aktywny jest sygnał RST. Może służyć do resetowania mikrokontrolera
ADDRESS	wejście	10	Szyna adresowa pamięci programu
INSTRUCTION	wyjście	18	Dane odczytane z pamięci programu o adresie ADDRESS (odczyt synchronizowany narastającym zboczem zegara CLK)

oraz wyświetlanie ich na ekranie wyświetlacza LCD, a także sterowanie diodami LED w odpowiedzi na wybrane kody ASCII odebranych znaków. Na **rys. 9** przedstawiono uproszczony schemat blokowy systemu realizującego wyżej określone zadania.

Zacienione bloki na schemacie z rys. 9 reprezentują moduły (wirtualne komponenty), dla których wyszczególniono wszystkie końcówki portów wejściowych oraz wyjściowych. Dla pozostałych bloków zaznaczono tylko te porty, które mają znaczenie z punktu widzenia komunikacji z mikrokontrolerem. Bramki AND, oznaczone cyfrą u dołu, do wejścia których dołączona jest magistrala portu *OUT_PORT* mikrokontrolera, to symbolicznie zaznaczone dekodery adresowe (układy kombinacyjne). Wyjście takiego dekodera znajduje się w stanie wysokim wówczas, gdy na 8-bitowym wejściu występuje wartość oznaczona cyfrą u dołu symbolu bramki. Ogólnie, sposób dołączenia do mikrokontrolera układów wejściowych a szczególnie wyjściowych z kombinacyjnym dekoderem, przerzutnikiem i bramką AND uwzględniającą sygnał strobujący (*WRITE_STROBE*) jest sposobem zalecanym przez *Xilinx*, zaczerpniętym z dokumentacji mikrokontrolera PicoBlaze. Możemy tu zauważyć, że w istocie układy wyjściowe pełnią rolę potokowego dekodera adresowego. Obydwa mikrokontrolery PicoBlaze i pBlazeZH rozkazy wejścia-wyjścia wykonują w ciągu 2 taktów zegara (pozostałe rozkazy pBlazeZH realizuje jednotaktowo, PicoBlaze potrzebuje tu dwóch taktów). Podczas pierwszego taktu zegara następuje dekodowanie adresu z wyjścia *PORT_ID* i ustawienie odpowiedniego przerzutnika (pierwszy stopień potoku), a w drugim takcie zegara realizowany jest właściwy zapis danych z portu *OUT_PORT* uwzględniający sygnał *WRITE_STROBE*. Dzięki zastosowaniu potoku poprawia się wydajność całego systemu (można zastosować większą częstotliwość taktującą), zwłaszcza w sytuacji, gdy do portu *PORT_ID* dołączonych jest wiele układów powodujących obciążenie tej magistrali.

Jak widać na rys. 9, w przykładowym systemie oprócz mikrokontrolera pBlazeZH i sprzętowego *bootloadera* wykorzystano również, opisany wcześniej, nadajnik i odbiornik transmisji szeregowej z buforem FIFO. Na pewną uwagę zasługuje sposób dołączenia wyświetlacza LCD. Przewidziano tutaj możliwość odczytu flagi zajętości wyświetlacza (najstarszego bitu magistrali danych), stąd też na schemacie zaznaczono zespół buforów trójstanowych sterowanych linią R/W wyznaczającą kierunek transferu danych pomiędzy wyświetlaczem i mikrokontrolerem.

Na **list. 1** pokazano kod modułu, który opisuje strukturę systemu z rys. 9. Nazwy portów wejścia-wyjścia modułu odnoszą się bezpośrednio do nazw stosowanych przez producenta dla zestawu uruchomieniowego UG330 (*Digilent*) z układem FPGA *Xilinx*. Kod jest jednak uniwersalny i można go wykorzystać dla układów programowalnych dowolnego producenta.

Opis w języku Verilog jest opisem mieszanym: strukturalno-behawioralnym. Cześć strukturalna odnosi się przede wszystkim do utworzenia instancji kolejnych modułów. W części behawioralnej został opisany pierwszy stopień potoku dekodera adresowego dla portów wejścia-wyjścia mikrokontrolera (linia (4)), porty wyjściowe sterujące diodami LED

List. 1. Kod modułu w języku Verilog implementujący strukturę podaną na rys. 9

```
module b_test(input CLK_50MHZ,
              input BTN_SOUTH,BTN_WEST,BTN_EAST,BTN_NORTH,
              output LCD_E,LCD_RS,LCD_RW,
              inout [7:0] LCD_DB,
              input [3:0] SW,
              output [7:0] LED,
              input RS232_DCE_RXD,
              output RS232_DCE_TXD);

wire [7:0] LCD_BUS,OUT_PORT,PORT_ID;
wire WRITE_STROBE,READ_STROBE,RX_INT,TX_INT,brst;
reg en0,en1,en2,en3;
wire [17:0] INSTRUCTION;
wire [9:0] ADDRESS;
wire [7:0] IN_PORT;
wire CLK_RX,CLK_TX,INT_ACK,reset;
reg [7:0] port0,port1,port2,port_in_reg;
wire [7:0] rx_data,mux_out;

DEBOUNCER d1
 (.clk(CLK_50MHZ),.PB({BTN_SOUTH,BTN_WEST,BTN_EAST,BTN_NORTH}),
  .BUTTONS({sw1,sw2,sw3,sw4}));

SERIAL_CLOCK #(.K(2416),.N(16)) c1
 (.CLK(CLK_50MHZ),.CLK_RX(CLK_RX),.CLK_TX(CLK_TX));

SERIAL_RX_FIFO rx1
 (.CLK_RX(CLK_RX),.RST(~reset),.RX(RX_INT),.CLK_RD(CLK_50MHZ),
  .RD_EN(en1&READ_STROBE),.DATA(rx_data),.DATA_RDY(rx_rdy));

SERIAL_TX_FIFO tx1
 (.CLK_TX(CLK_TX),.RST(~reset),.WR_EN(en3&WRITE_STROBE),
 .CLK_WR(CLK_50MHZ),.TX(TX_INT),.DATA(OUT_PORT));

pBlazeZH pb1
 (.IN_PORT(port_in_reg),.INTERRUPT(rx_rdy),.RESET(brst),
  .CLK(CLK_50MHZ),.INSTRUCTION(INSTRUCTION),.OUT_PORT(OUT_PORT),
  .PORT_ID(PORT_ID),.READ_STROBE(READ_STROBE),
  .WRITE_STROBE(WRITE_STROBE),.INTERRUPT_ACK(INT_ACK),.ADDRESS(ADDRESS));

BOOTLOADER #(.K(2416),.N(16)) bloader               //(1)
 (.CLK(CLK_50MHZ),.ON(SW[0]),.RST(~reset),
  .ADDRESS(ADDRESS),.INSTRUCTION(INSTRUCTION),
  .TX_INT(TX_INT),.RX(RS232_DCE_RXD),
  .RX_INT(RX_INT),.TX(RS232_DCE_TXD),.RST_OUT(brst));

assign reset=sw1;
assign LED=port1;
assign LCD_BUS=port0;
assign LCD_RW=port2[0];
assign LCD_RS=port2[1];
assign LCD_E=port2[2];
assign LCD_DB=LCD_RW?8'hzz:LCD_BUS;     //(2)
assign BUSY_FLAG=LCD_DB[7];             //(3)

always @(posedge CLK_50MHZ)             //(4)
if(reset) begin
 en0<=0; en1<=0; en2<=0; en3<=0;
end else
begin
 en0<=PORT_ID==0;
 en1<=PORT_ID==1;
 en2<=PORT_ID==2;
 en3<=PORT_ID==3;
end

always @(posedge CLK_50MHZ)           //(5)
if (reset)
begin port0<=0; port1<=0; port2<=0; end
else
begin
 if(en0&WRITE_STROBE) port0<=OUT_PORT;
 if(en1&WRITE_STROBE) port1<=OUT_PORT;
 if(en2&WRITE_STROBE) port2<=OUT_PORT;
end

assign mux_out=PORT_ID[0]?rx_data:{7'b0000000,BUSY_FLAG}; //(6)
always @(posedge CLK_50MHZ) port_in_reg<=mux_out;         //(7)

endmodule
--
```

i wyświetlaczem LCD (linia (5)) oraz multiplekser wraz z przerzutnikiem dla portów wejściowych (linie (6) i (7)).

Zwróćmy jeszcze uwagę na sposób utworzenia instancji *bootloadera*. Odpowiedni zapis znajduje się w linii (1). Istotne jest tutaj wprowadzenie właściwych parametrów modułu wyznaczających prędkość transmisji danych za pomocą portu szeregowego. Dla przebiegu zegarowego o częstotliwości 50 MHz, dostępnego w zestawie UG330, przyjęto wartości parametrów K=2416 i N=16, co pozwala na uzyskanie częstotliwości wyjściowej syntezera równej 1,8432617 MHz. Szesnastokrotna wartość częstotliwości nominalnej odpowiadająca prędkości transmisji 115200 bit/s powinna wynosić 1,8432 MHz. Oznacza to, że dla podanych wartości parametrów uzyskuje się wystarczającą dokładność częstotliwości taktującej układy portu szeregowego.

W kontekście schematu blokowego z rys. 9 dodajmy jeszcze, że sygnał wejściowy *ON* modułu *bootloadera* dołączony jest do portu *SW[0]* modułu nadrzędnego z list. 1. Globalny sygnał resetujący cały system obsługiwany jest przez port *BTN_SOUTH*. W przypadku zestawu UG330 do portów *SW[0]...SW[3]* podłączone są cztery dwupołożeniowe (bistabilne) przełączniki, zaś do portów *BTN_SOUTH* itd. – przełączniki typu *switch*. Sygnały z tych ostatnich przełączników dołączone są w module nadrzędnym do jedynego nieopisanego wcześniej modułu eliminacji mechanicznych drgań zestyków (moduł *DEBOUNCER*). Pełne kody źródłowe wszystkich modułów dostępne są w materiałach dodatkowych.

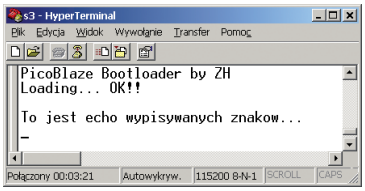
Dodajmy jeszcze, że sygnał *ON* modułu *bootloadera* nie wymaga eliminowania oscylacji pochodzących z mechanicznych przełączników. Odpowiedni obwód jest już zintegrowany w module. Warto również zwrócić uwagę na sposób opisu bufora trójstanowego dla magistrali danych wyświetlacza LCD i sposób odczytu flagi zajętości wyświetlacza. Odpowiedni kod zawarty jest w liniach (2) oraz (3).

Na kolejnym listingu (**list. 2**) przedstawiono fragment programu sterującego mikrokontrolerem. Jest to kod procedury obsługi przerwania, który realizuje założone zadania przykładowej aplikacji. Zwróćmy uwagę, że dzięki odpowiedniej strukturze części sprzętowej systemu, obsługa programowa portu szeregowego jest banalnie prosta. Odebranie znaku przesyłanego łączem szeregowym powoduje zgłoszenie przerwania. Odczyt tego znaku może być zrealizowany przez pojedynczą instrukcję asemblera. Podobnie wysłanie znaku do portu szeregowego również może być wykonane za pomocą pojedynczej instrukcji.

Uruchomienie całej aplikacji jest procesem dwuetapowym. Polega na skompilowaniu (synteza i optymalizacja, trasowanie i wyznaczanie połączeń) całego projektu z mikrokontrolerem i *bootloaderem*, przesłaniu konfiguracji do wybranego układu programowalnego oraz skompilowaniu w asemblerze KCPSM3 kodu programu sterującego i przesłaniu do *bootloadera* kodu wynikowego za pomocą portu szeregowego i np. aplikacji *HyperTerminal*. Natychmiast po zakończeniu transmisji portem szeregowym mikrokontroler rozpoczyna realizację programu. Wymuszenie poziomu niskiego na wejściu *ON bootloadera* (zmiana położenia przełącznika dołączonego do portu *SW[0]* zestawu UG330) umożliwia mikrokontrolerowi obsługę danych przesyłanych portem szeregowym. Wpisywanie z klawiatury kolejnych znaków w programie *HyperTerminal* skutkuje pojawianiem się tych samych znaków w oknie tej aplikacji (echo odsyłane przez mikrokontroler) oraz na wyświetlaczu LCD zestawu. Przekroczenie 16 znaków w danej linii wyświetlacza powoduje przejście do kolejnej linii (na zmianę drugiej, pierwszej, drugiej itp.), wykasowanie tekstu w tej linii i rozpoczęcie wyświetlania znaków. Przejście do kolejnej linii na wyświetlaczu LCD realizowane jest również w przypadku odebrania kodu ASCII CR (powrót karetki – 0Dh). Szczególne znaczenie mają kody ASCII 31h...38h (klawi-

List. 2. Fragment programu w asemblerze KCPSM3 sterującego pracą mikrokontrolera

```
UART_RX: ; podprogram obsługi przerwania z portu UART
   INPUT   SB,SERIAL_RX ; pobierz znak z portu UART
   OUTPUT  SB,SERIAL_TX ; przepisz na wyjście (wyślij do UART)
   COMPARE SB,0D        ; sprawdź, czy znak to CR?
   JUMP    NZ,UCNT_NC
   LOAD    SA,0A        ; dodaj LF
   OUTPUT  SA,SERIAL_TX
   LOAD    SA,00        ; i wyjdź z obsługi przerwania
   STORE   SA,1F
   RETURNI ENABLE
UCNT_NC:
   FETCH   SA,1F
   COMPARE SA,00    ; sprawdź nr kolumny wyświetlanego znaku
   JUMP    NZ,UCNT1 ; jeśli kolumna 0 - wyczyść cały wiersz
   FETCH   SC,1E
   COMPARE SC,80    ; sprawdź numer wiersza
   JUMP    Z,UCNT_N
   LOAD    S5,80    ; ustaw 1. wiersz
   JUMP    UCNT_C
UCNT_N:
   LOAD    S5,C0    ; ustaw 2. wiersz
UCNT_C:
   STORE   S5,1E
   CALL    LCD_WRITE_CMD
   LOAD    SC,00
   LOAD    S5,20
UCNTLP:             ; wypełnianie spacjami całego wiersza
   CALL    LCD_WRITE_CHR
   ADD     SC,01
   COMPARE SC,10
   JUMP    NZ,UCNTLP
UCNT1:
   FETCH   S5,1E
   ADD     S5,SA
   CALL    LCD_WRITE_CMD ; ustaw odpowiednią kolumnę
   ADD     SA,01    ; inkrementuj licznik znaków
   COMPARE SA,10    ; czy wyświetlono już 16 znaków?
   JUMP    NZ,UCNT2
   LOAD    SA,00    ; jeśli tak, zeruj licznik znaków
UCNT2:
   STORE   SA,1F    ; zapisz wartość licznika do pamięci notatnikowej
   LOAD    S5,SB
   CALL    LCD_WRITE_CHR ; wyświetl znak na ekranie LCD
   COMPARE SB,31    ; sprawdź przedział, czy znak jest ,1'..'8'?
   JUMP    C,URET   ; znak jest < ,1' - skok
   COMPARE SB,39
   JUMP    NC,URET  ; znak jest >= ,9' - skok
   LOAD    SE,31    ; znak należy do przedziału ASCII ,1'..'8'
   LOAD    SF,01
UCNT3:
   COMPARE SE,SB    ; ustalenie kodu ASCII znaku i odpowiadającego
   JUMP    Z,UCNT4  ; mu bitu w rejestrze SF
   COMPARE SE,38
   JUMP    Z,URET
   ADD     SE,01
   SL0     SF
   JUMP    UCNT3
UCNT4:
   FETCH   SA,20    ; odczyt z pamięci notatnikowej stanu LED
   XOR     SA,SF    ; negacja odpowiedniego bitu
   STORE   SA,20    ; zapis do pamięci notatnikowej
   OUTPUT  SA,LED   ; aktualizacja stanu portu - diod LED
URET:
   RETURNI ENABLE   ; powrót z przerwania od układu UART
--
```



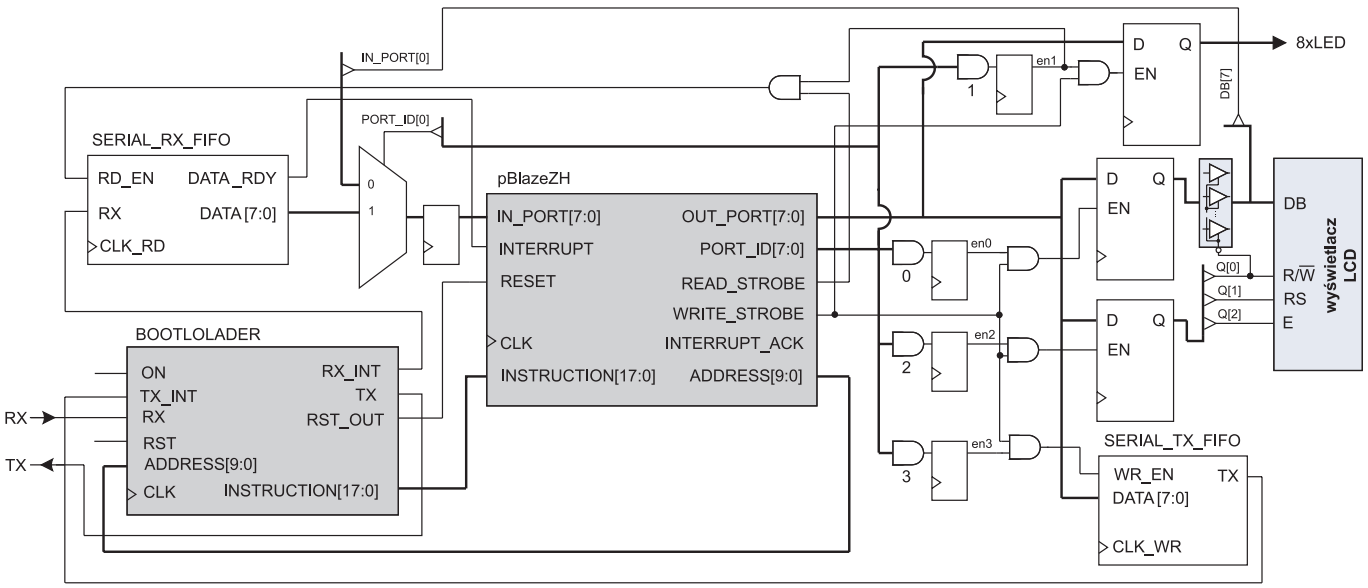
Rys. 10. Widok okna programu *HyperTerminal* komunikującego się z *bootloaderem* i mikrokontrolerem

sze '1'...'8'). Wystąpienie jednego z tych kodów powoduje na zmianę zapalenie lub zgaszenie odpowiedniej diody LED. **Rys. 10** przedstawia przykładowy widok okna programu *HyperTerminal* podczas komunikacji z omawianym systemem.

Podsumowanie

Przedstawiony tutaj sprzętowy *bootloader* znacznie ułatwia proces uruchamiania oprogramowania przeznaczonego dla osadzonych mikrokontrolerów PicoBlaze. Skraca czas i redukuje liczbę czynności niezbędnych do wymiany kodu programu. W prezentowanej wersji opisu implementacja wirtualnego komponentu *bootloadera* w układach FPGA *Xilinx* z rodziny *Spartan 3* wymaga 188 bloków logicznych *Slice*. Jest to wynik bardzo zbliżony do wymagań na zasoby logiczne mikrokontrolera pBlazeZH. Maksymalna częstotliwość taktowania *bootloadera*, zaimplementowanego w tych samych układach, może sięgać aż 136 MHz (pBlazeZH może być taktowany maksymalnie zegarem o częstotliwości 90 MHz). Dużą zaletą wirtualnego komponentu *bootloadera*, podobnie jak mikrokontrolera pBlazeZH, jest jego niezależność od docelowej architektury układu programowalnego. Może być on implementowany w układach dowolnego producenta, a nie tylko w układach *Xilinx*.

Zbigniew Hajduk
zhajduk@prz-rzeszow.pl



Rys. 9. Schemat blokowy przykładowego systemu z mikrokontrolerem pBlazeZH