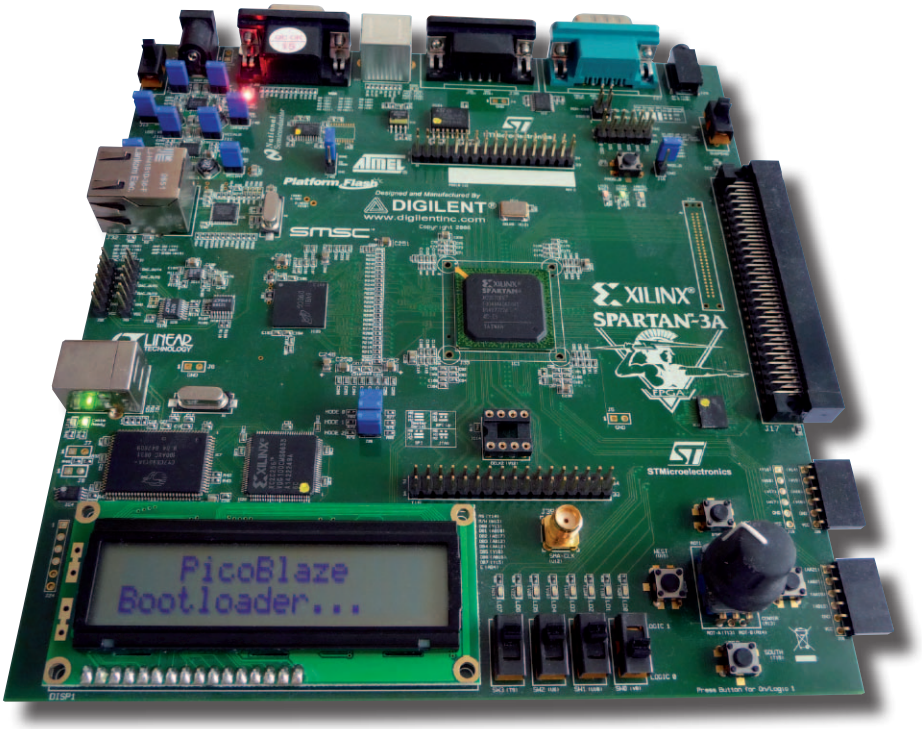
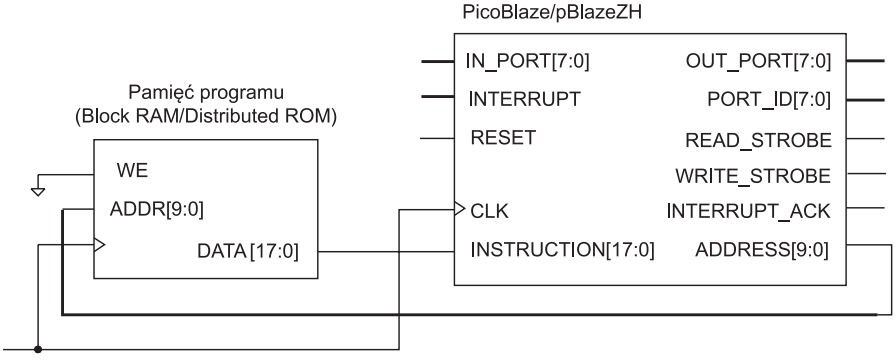
Sprzętowy bootloader dla mikrokontrolerów PicoBlaze (1)

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. W artykule opisano sposób rozwiązania tego problemu dla mikrokontrolerów PicoBlaze.

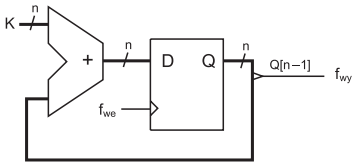


Typowy sposób dołączenia pamięci programu dla popularnego mikrokontrolera PicoBlaze, a także jego przyśpieszonej i bardziej uniwersalnej wersji pBlazeZH, przedstawia **rys. 1**. Pamięć programu może być zrealizowana albo z wykorzystaniem dedykowanych bloków pamięci RAM (*Block* RAM) dostępnych w układach FPGA, albo jako rozproszona pamięć ROM, której implementacja opiera się na wykorzystaniu bloków logicznych (tablic LUT) układu programowalnego. Zapis kodu wynikowego do pamięci programu, niezależnie od jej typu, wymaga definiowania wielu współczynników w kodzie źródłowym w języku HDL opisującym cały projekt z osadzonym mikrokontrolerem. Każda zmiana programu dla mikrokontrolera pociąga za sobą konieczność ponownej kompilacji całego projektu i przesłania konfiguracji do układu FPGA. Kompilacja projektów, zależnie od ich złożoności, może trwać dość długo a tym samym proces rozwoju oraz weryfikacji oprogramowania mikrokontrolera w systemie docelowym staje się mocno niewygodny i nieefektywny. Firma Xilinx proponuje rozwiązanie tego problemu poprzez stosowanie kilku specyficznych technik, np. tzw. częściowej rekonfiguracji FPGA poprzez interfejs JTAG czy wykorzystanie specjalnego oprogramowania, które w pliku wynikowym do konfiguracji FPGA zamienia dane dla pamięci *Block* RAM. Jednak techniki te bezpośrednio nadają się do zastosowania tylko dla układów Xilinx (mikrokontroler pBlazeZH może być zaimplementowany za pomocą układów dowolnego producenta) i mimo wszystko nie są zbyt komfortowe.

Podobny problem wygodnego załadowania kodu wynikowego programu do pamięci programu istnieje również w przypadku mikrokontrolerów wytwarzanych jako układy ASIC. Popularnym rozwiązaniem tego zagadnienia jest wykorzystanie tzw. *bootloadera* czyli programu rezydującego w wybranym obszarze pamięci typu Flash, który za pomocą zwykłego portu szeregowego zapisuje



Rys. 1. Typowy sposób dołączenia pamięci programu dla mikrokontrolerów PicoBlaze



Rys. 2. Uproszczony schemat architektury sprzętowego *bootloadera* oraz sposób jego dołączenia do mikrokontrolera PicoBlaze

przesyłane dane (kod programu) do wybranego obszaru tej pamięci. Ideę takiego *bootloadera* wykonanego w wersji sprzętowej wykorzystamy do wygodnego załadowania kodu wynikowego do pamięci programu mikrokontrolera PicoBlaze. Przy czym w odróżnieniu od programowych *bootloaderów* dla mikrokontrolerów ASIC nasz *bootloader* będzie zapisywał dane bezpośrednio do pamięci programu zrealizowanej jako pamięć RAM, z pominięciem pamięci konfiguracji układu FPGA. Po każdorazowym wyłączeniu i załączeniu napięcia zasilania całego systemu konieczne zatem będzie ponowne ładownie kodu programu dla mikrokontrolera. Z tego też powodu zastosowania sprzętowego *bootloadera* ograniczają się do fazy uruchamiania i testowania oprogramowania.

Sprzętowy *bootloader* został zaprojektowany w postaci tzw, wirtualnego komponentu i istnieje jako odpowiedni kod w języku opisu sprzętu HDL, w tym przypadku w języku Verilog.

Przy czym kod ten jest całkowicie niezależny od docelowej architektury (producenta, rodziny) układu programowalnego, w którym może być zaimplementowany. Wirtualny komponent bootloadera należy dołączyć do całego projektu z mikrokontrolerem PicoBlaze/pBlazeZH na czas uruchamiania oprogramowania dla mikrokontrolera. Gdy prace nad rozwojem oprogramowania zostaną zakończone, wówczas wirtualny komponent *bootloadera* może być usunięty z projektu, a opracowany kod programu dla mikrokontrolera należy zapisać w sposób konwencjonalny – jako zbiór odpowiednich współczynników inicjujących zawartość pamięci *Block RAM* lub *Distributed ROM*.

Projektując wirtualny komponent *bootloadera*, zdecydowano, że będzie on interpretował jeden z plików wynikowych generowanych przez asembler KCPSM3 (standardowy program asemblera dostarczany wraz z całym pakietem zawierającym m.in. kod mikrokontrolera PicoBlaze). Wybór padł na plik tekstowy z rozszerzeniem HEX. Plik ten zawiera odwzorowanie pamięci programu mikrokontrolera: w kolejnych wierszach, odpowiadających kolejnym komórkom pamięci, zawarte są kody rozkazów zapisane w postaci 5 cyfr szesnastkowych. Z poziomu komputera PC plik HEX można przesłać do *bootloadera* za pomocą zwykłej aplikacji *HyperTerminal*. Pewną wadą takiego rozwiązania jest brak kontroli poprawności danych przesyłanych za pomocą portu szeregowego. Plik HEX generowany przez asembler KCPSM3 nie zawiera żadnych sum kontrolnych, a założenie wykorzystania aplikacji *HyperTerminal* również nie pozwala na wprowadzenie odpowiednich mechanizmów kontroli. Z kolei jako zaletę można wymienić prostą obsługę i brak konieczności tworzenia dedykowanej aplikacji zapewniającej komunikację z *bootloaderem*

Architektura sprzętowego *bootloadera*

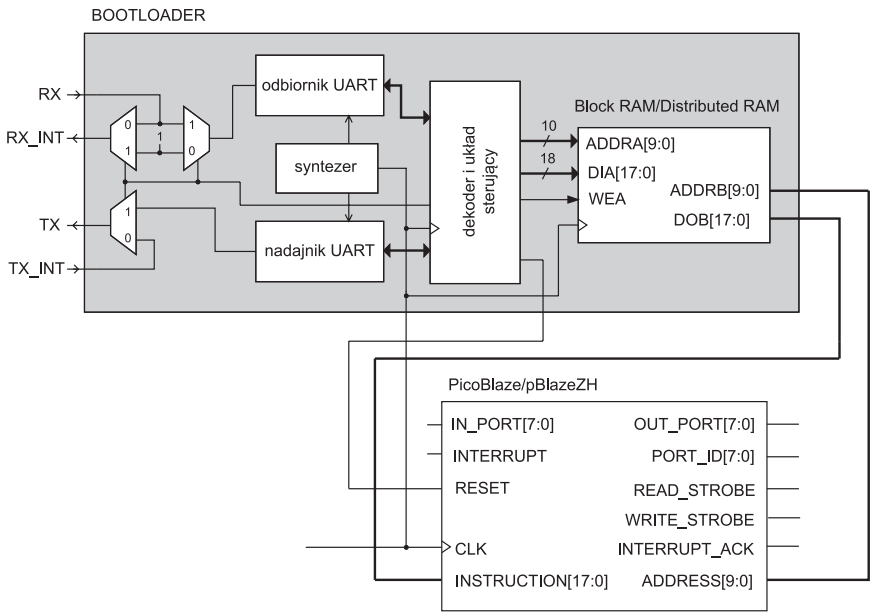
Na **rys. 2** przedstawiono uproszczony schemat architektury sprzętowego *bootloadera*. Rysunek ilustruje również sposób, w jaki *bootloader* integruje się z mikrokontrolerem PicoBlaze. Zasadniczymi blokami funkcjonalnymi *bootloadera* są: nadajnik oraz odbiornik asynchronicznej transmisji szeregowej, syntezer wytwarzający przebiegi taktujące układy transmisji szeregowej, dwuportowa pamięć RAM oraz dekoder i układ sterujący. Zespół multiplekserów widoczny na rysunku służy do przełączania linii transmisji szeregowej (RX oraz TX) pomiędzy *bootloader* a ewentualnie występujący w systemie inny port szeregowy.

Wirtualny komponent *bootloadera* w rzeczywistości ma strukturę hierarchiczną. Z poziomu modułu nadrzędnego tworzone są instancje modułów (również wirtualnych komponentów) odbiornika i nadajnika transmisji szeregowej oraz syntezera. Pozostałe elementy widoczne na rys. 2 opisane są bezpośrednio w module nadrzędnym.

W dalszej części artykułu przedstawimy ogólny (bardziej od strony interfejsu) opis wirtualnych komponentów obsługi portu szeregowego, które mogą zostać wykorzystane również w innych aplikacjach. Odpowiednie kody źródłowe dostępne są w materiałach dodatkowych. Nieco bardziej szczegółowo omówimy wirtualny komponent modułu nadrzędnego *bootloadera* oraz pokażemy przykładową aplikację wykorzystującą mikrokontroler pBlazeZH oraz sprzętowy *bootloader*.

Syntezer

Istotnym składnikiem portu szeregowego jest układ odpowiedzialny za wytwarzanie odpowiedniej częstotliwości sygnału taktującego dla szeregowego nadajnika i odbiornika danych. Wytworzenie takiego sygnału jest bardzo proste, jeżeli w systemie dostępna jest częstotliwości, której wartość, podzielona przez współczynnik będący potęgą liczby 2, stanowi wymaganą wartość częstotliwości. Na przykład, jeżeli założymy, że nasz port szeregowy powinien pracować z szybkością 115200 bit/s, wówczas nominalna częstotliwość taktowania modułów odbiornika i nadajnika danych powinna wynosić 115,2 kHz. Częstotliwość taką łatwo uzyskać, dysponując sygnałem zegara z generatora kwarcowego o częstotliwości 1,8432 MHz oraz dzielnikiem o współczynniku 16 (licznikiem modulo 16). Nie zawsze



Rys. 3. Budowa syntezera DDFS

jednak sygnał o tak dogodnej częstotliwości dostępny jest w systemie.

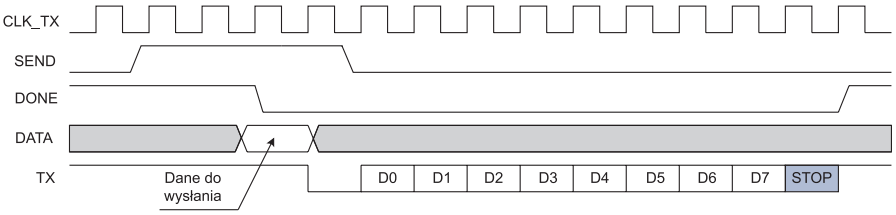
Niektóre układy programowalne FPGA dysponują wbudowanymi blokami syntezerów częstotliwości, ułatwiającymi otrzymanie przebiegów o innych częstotliwościach niż globalny sygnał zegarowy. Na przykład, układy z rodziny *Xilinx Spartan 3* mają po kilka bloków zarządzania sygnałem zegarowym DCM (*Digital Clock Manager*), które można skonfigurować jako syntezery częstotliwości. Uzyskana za pomocą takich bloków częstotliwość wyjściowa jest iloczynem częstotliwości wejściowej i pewnej liczby wymiernej podawanej jako parametr. Może być więc zarówno większa, jak i mniejsza od częstotliwości wejściowej.

W przypadku, gdy dany układ FPGA nie jest wyposażony we wbudowany blok syntezera, w celu uzyskania odpowiedniej częstotliwości taktującej układy transmisyjne portu szeregowego, można posłużyć się techniką bezpośredniej cyfrowej syntezy częstotliwości (DDFS – *Direct Digital Frequency Synthesis*). Budowa syntezera DDFS jest bardzo prosta – **rys. 3**. Syntezer taki składa się z *n*-bitowego sumatora oraz zespołu *n* przerzutników typu D. Wyjście sumatora połączone jest z wejściem danych zespołu przerzutników. Wyjście przerzutników podawane jest na jedno z wejść sumatora (sprzężenie zwrotne). Do drugiego wejścia sumatora przekazywana jest wartość pewnej stałej *K*. Wraz z każdym taktem sygnału zegarowego (częstotliwość wejściowa), wartość tej stałej dodawana jest do poprzedniego stanu pamiętanego przez zespół przerzutników. Najbardziej znaczący bit wyjścia przerzutników stanowi wyjście syntezera. Częstotliwość wyjściowa określona jest zależnością:

$$f_{wy} = \frac{f_{we} \cdot K}{2^n}$$

Tak zbudowany syntezer zapewnia wytworzenie częstotliwości wyjściowej z rozdzielczością 2^{-n}. Na przykład dla n=48 rozdzielczość ta wynosi aż $3{,}5*10^{-15}$. Wadą tego rozwiązania jest to, że częstotliwość wyjściowa musi być mniejsza niż połowa częstotliwości wejściowej. Dodatkowo wyjście obarczone jest drżeniem fazy (*jitter*) o częstotliwości sygnału wejściowego.

Opisane w dalszej części tekstu wirtualne komponenty nadajnika i odbiornika danych portu szeregowego wymagają dwóch różnych częstotliwości taktujących. Odbiornik powinien być taktowany częstotliwością 16 razy większą niż częstotliwość nominalna. Z drugiej strony nadajnik portu szeregowego potrzebuje sygnału taktującego o częstotliwości nominalnej. Dlatego też wirtualny komponent sytntezera winien dostarczać dwóch częstotliwości: jednej nominalnej – odpowiadającej wymaganej prędkości transmisji oraz drugiej, 16 razy większej od częstotliwości nominalnej.



Rys. 4. Przebiegi czasowe dla wirtualnego komponentu nadajnika UART

Tab. 1. Opis parametrów i portów wirtualnego komponentu syntezera

moduł: SERIAL_CLOCK

Parametr	Opis
K	Stała K według formuły (1)
N	Liczba bitów sumatora i zespołu przerzutników

Port	Kierunek	Liczba bitów	Opis
CLK	wejście	1	Sygnał taktujący – częstotliwość wejściowa
CLK_TX	wyjście	1	Sygnał taktujący odbiornik UART o częstotliwości nominalnej (CLK_RX/16)
CLK_RX	wyjście	1	Sygnał taktujący nadajnik UART o częstotliwości 16 razy większej niż częstotliwość nominalna

Tab. 2. Opis portów wirtualnego komponentu nadajnika UART

moduł: SERIAL_TX

Port	Kierunek	Liczba bitów	Opis
CLK_TX	wejście	1	Sygnał taktujący nadajnik o częstotliwości nominalnej
RST	wejście	1	Sygnał resetujący. Aktywny poziom niski
SEND	wejście	1	Sygnał aktywujący transmisję bajtu. Aktywny poziom wysoki. Musi być utrzymywany tak długo w stanie aktywnym, aż sygnał DONE zmieni stan na niski
DATA	wejście	8	Dane (bajt) do przesłania
TX	wyjście	1	Szeregowe wyjście danych nadajnika
DONE	wyjście	1	Sygnał informujący o zakończeniu transmisji bajtu danych. Aktywny poziom wysoki

W **tab. 1** przedstawiono opis interfejsu wirtualnego komponentu syntezera (kod źródłowy w języku Verilog dostępny jest w materiałach dodatkowych).

Nadajnik UART

Zadanie nadajnika portu szeregowego jest bardzo proste i polega na wygenerowaniu synchronicznie z sygnałem taktującym bitu startu, 8 bitów danych oraz bitu stopu. Opis interfejsu wirtualnego komponentu realizującego funkcję nadajnika UART przedstawia **tab. 2**.

Na **rys. 4** zilustrowano przebiegi czasowe dla wirtualnego komponentu nadajnika UART. Rozpoczęcie transmisji danych przez nadajnik inicjowane jest poprzez ustawienie w stan wysoki sygnału *SEND*. Stan wysoki powinien utrzymywać się na tym wejściu tak długo aż sygnał *DONE* zmieni swój stan na niski (co następuje po dwóch taktach zegara *CLK_TX* od momentu ustawienia w stan wysoki linii *SEND*), potwierdzając tym samym rozpoczęcie transmisji. Taki sposób inicjowania transmisji wynika z faktu, że nadajnik portu UART w docelowym systemie zazwyczaj pracuje z inną częstotliwością zegara niż pozostałe układy. Konieczna jest więc synchronizacja odpowiednich sygnałów. W kolejnym takcie zegara, po wyzerowaniu wyjścia *DONE* rozpoczyna się właściwa transmisja danych. Po wygenerowaniu ostatniego z bitów – bitu stopu wyjście DONE zostaje ustawione w stan wysoki oznaczający zakończenie transmisji danych przez nadajnik.

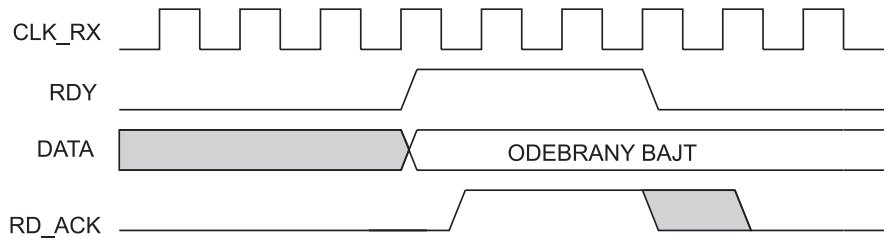
Odbiornik UART

Funkcja odbiornika asynchronicznej transmisji szeregowej jest nieco bardziej złożona niż nadajnika. Odbiornik powinien monitorować stan wejścia danych i w momencie kiedy pojawi się opadające zbocze tego sygnału (przesyłany jest bit startu), zsynchronizować swoje układy czasowe, tak aby możliwe stało się poprawne odmierzanie kolejnych interwałów czasowych odpowiadających transmisji jednego bitu z zadaną szybkością.

W zaproponowanym tu rozwiązaniu przyjęliśmy, że odbiornik będzie taktowany częstotliwością 16 razy większą niż częstotliwość odpowiadająca wybranej szybkości transmisji. Dzięki temu stan danego bitu może być próbkowany kilkakrotnie w przeciągu odpowiadającego mu interwału czasowego. W przypadku omawianego tu wirtualnego komponentu nadajnika UART stan jednego bitu próbkowany jest trzykrotnie. Osta-

Tab. 3. Opis portów wirtualnego komponentu nadajnika UART

moduł: SERIAL_TX			
Port	**Kierunek**	**Liczba bitów**	**Opis**
CLK_RX	wejście	1	Sygnał taktujący odbiornik
RST	wejście	1	Sygnał resetujący. Aktywny poziom niski
RD_ACK	wejście	1	Sygnał potwierdzający odczyt danych z modułu nadajnika. Aktywny poziom wysoki. Uaktywnienie tego wejścia powoduje wyzerowanie wyjścia RDY
RX			Szeregowe wejście danych odbiornika UART
DATA	wyjście	8	Odebrane dane
RDY	wyjście	1	Sygnał informujący o odebraniu nowego bajtu danych. Aktywny poziom wysoki



Rys. 5. Przebiegi czasowe dla wirtualnego komponentu odbiornika UART

teczną wartość odebranego bitu przyjmuje się taką, jaka wystąpiła przynajmniej dwa razy podczas wspomnianego próbkowania.

W **tab. 3** przedstawiono opis interfejsu wirtualnego komponentu nadajnika UART, a na **rys. 5** odpowiednie przebiegi czasowe. Odebranie bajtu danych przez odbiornik sygnalizowane jest ustawieniem w stan wysoki wyjścia *RDY*. Po odczytaniu tego bajtu należy ustawić w stan wysoki sygnał potwierdzający odczyt *RD_ACK*. Po dwóch taktach zegara *CLK_RX* od momentu ustawienia wejścia *RD_ACK* wycofywany jest sygnał *RDY*. W odpowiedzi na ten fakt powinien być wycofany również sygnał *RD_ACK*. Taki sposób komunikacji z odbiornikiem UART, podobnie jak w przypadku nadajnika, wynika z konieczności synchronizowania odpowiednich sygnałów odbiornika z sygnałami sterującymi modułu nadrzędnego taktowanego inną częstotliwością niż odbiornik.

Zbigniew Hajduk
zhajduk@prz-rzeszow.pl