Zrób sobie procesor: PicoBlaze w FPGA, część 3

Budowa PicoBlaze, asembler

Kończymy wstępną część kursu, za miesiąc pokażemy pierwszy przykład programu napisanego dla PicoBlaze – prostego, 8–bitowego procesora napisanego w VHDL i Verilogu, łatwo implementowalnego w najtańszych układach FPGA z rodziny Spartan 3 firmy Xilinx.

Podłączenie pamięci programu do PicoBlaze

Do PicoBlaze jest podłączana pamięć programu o konfiguracji 1024 słów 18–bitowych (dla niektórych wcześniejszych wersji procesora jest wymagana mniejsza pamięć). Na **rys. 6** przedstawiono najprostszy sposób dołączenia pamięci – jak widać jest to pamięć synchroniczna, konieczne jest doprowadzenie do niej sygnału taktującego.

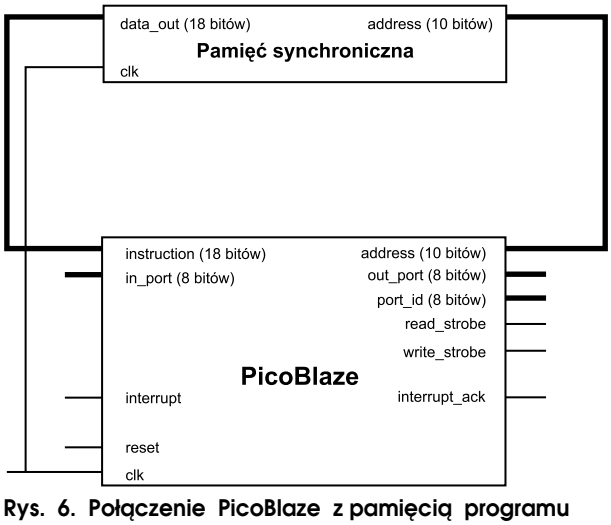
Opis plików

W trakcie procesu pobierania plików PicoBlaze ze strony producenta należy najpierw się zarejestrować (publikujemy je także na CD–EP6/2008B). Następnie należy podać, m.in., dla której rodziny układów FPGA/CPLD mikrokontroler ma być przeznaczony. Ponieważ z reguły strony internetowe ewoluują dosyć szybko, więc nie będziemy zamieszczać tutaj konkretnych informacji w jaki sposób wypełnić wszystkie formularze.

Po poprawnej rejestracji można zgrać na komputer skompresowany plik, zawierający źródła PicoBlaze, asembler, przykłady i dokumentację. Z czterech dostępnych wersji PicoBlaze, w dalszej części będziemy omawiać wersję przeznaczoną dla rodziny Spartan 3 (plik o nazwie *kcpsm3.zip*).

Po rozpakowaniu mamy do dyspozycji następujące pliki i katalogi:
readme.txt – plik, opisujący pliki dostępne w tym archiwum;
KCPSM3_manual.pdf – nota aplikacyjna, opisująca PicoBLaze;
kcpsm3.ngc – UART_manual.pdf – opis sposobu podłączenia układu kontroli transmisji szeregowej do PicoBlaze;
UART_real_time_clock.pdf – opis wykonania przykładowego projektu – zegara czasu rzeczywistego;
Assembler – katalog zawierający narzędzia do asemblera PicoBlaze;
kcpsm3.exe – asembler PicoBlaze, generuje on model pamięci programu, na podstawie opisu w asemblerze;
int_test.psm – przykładowy program w asemblerze, zawierający test mikroprocesora;
uclock.psm – program w asemblerze do przykładowego projektu – zegara czasu rzeczywistego;
ROM_form.vhd – wzór opisu implementacji pamięci programu w języku VHDL;
ROM_form.v – wzór opisu implementacji pamięci programu w języku Verilog;
clean.bat – pomocniczy program do czyszczenia raz wygenerowanych plików.

Proszę zwrócić uwagę, że programy przygotowane w języku asembler mają rozszerzenie *.psm*. Jest to rozszerzenie preferowane, a nie wymagane.
VHDL – katalog zawierający źródła PicoBlaze, w języku VHDL;
kcpsm3.vhd – opis PicoBlaze;
embedded_kcpsm3.vhd – przykładowy opis sposobu połączenia mikroprocesora PicoBlaze z pamięcią programu;
kcpsm3_int_test.vhd – implementacja PicoBlaze do celów testowych;
test_bench.vhd – opis z testem dla implementacji do celów testowych – do wykorzystania w programie symulującym język VHDL;



Rys. 6. Połączenie PicoBlaze z pamięcią programu

UART9_readme.txt – opis przygotowania układu do komunikacji szeregowej z opcją sprawdzania bitu parzystości;
bbfifo_16x8.vhd – FIFO, 16 słów 8–bitowych, opis wykorzystywany przy implementacji układu do komunikacji szeregowej;
bbfifo_16x9.vhd – FIFO, 16 słów 9–bitowych, opis wykorzystywany przy implementacji układu do komunikacji szeregowej;
kcuart_rx.vhd – 8–bitowa implementacja odbiornika do komunikacji szeregowej;
kcuart_tx.vhd – 8–bitowa implementacja nadajnika do komunikacji szeregowej;
kcuart9_rx.vhd – 9–bitowa implementacja odbiornika do komunikacji szeregowej;
kcuart9_tx.vhd – 9–bitowa implementacja nadajnika do komunikacji szeregowej;
uart_rx.vhd – odbiornik do komunikacji szeregowej z dołączoną kolejką FIFO;
uart_tx.vhd – nadajnik do komunikacji szeregowej z dołączoną kolejką FIFO;
uart9_rx.vhd – odbiornik do komunikacji szeregowej z dołączoną kolejką FIFO i z opcją sprawdzania bitu parzystości;
uart9_tx.vhd – nadajnik do komunikacji szeregowej z dołączoną kolejką FIFO i z opcją sprawdzania bitu parzystości;
uart_clock.vhd – główny plik, realizujący przykład kontrolera czasu rzeczywistego;
Verilog – katalog zawierający źródła PicoBlaze w języku Verilog. Poza główną implementacją PicoBlaze, jeszcze tylko część plików zaimplementowanych w języku VHDL została przepisana w języku Verilog. Pliki te mają taką samą nazwę (poza rozszerzeniem) i takie same funkcje.
JTAG_loader – katalog zawiera niezbędne liki i narzędzia do połączenia PicoBlaze z portem JTAG w układzie reprogramowalnym;
DATA2MEM_assistance – katalog zawiera narzędzia niezbędne do podmiany samej pamięci pragramu PicoBlaze w pliku konfiguracyjnym w układzie PLD. Takie rozwiązanie jest użyteczne do skrócenia czasu potrzebnego od momentu wykonania zmian w programie do jej przetestowania w prawdziwym układzie.

Asembler kcpsm3.exe

Program *kcpsm3.exe* pozwala na podstawie opisu programu w asemblerze wygenerować element pamięci opisany językiem opisu sprzętu (tylko VHDL i Verilog), zawierający pamięć zainicjowaną wartościami odpowiadającymi programowi PicoBlaze.

Program jest przeznaczony do uruchamiania na platformie Windows, jako narzędzie DOS–owe. Wszystkie pliki wejściowe programu muszą spełniać DOS–owy rygor nazewnictwa: do 8 liter nazwy, kropka i do 3 liter rozszerzenia. Przed uruchomieniem programu trzeba przygotować 4 pliki wejściowe:

– *<program>.[psm]* – Kod programu, napisany w asemblerze. Rozszerzenie *psm* jest domyślne i nie trzeba go podawać w momencie uruchomienia narzędzia.
– *ROM_form.vhd* i *ROM_form.v* – wejściowa forma opisu modułu pamięci, na podstawie której zostanie wygenerowany docelowy opis. Pliki te można zmieniać, w zależności od potrzeb zastosowania.
– *ROM_form.coe* – parametry generowanego modułu pamięci, wykorzystywany przez program *Core Generator*.

Do celów pierwszego projektu utwórzmy plik o nazwie *prog_1.psm*, zawierający program w asemblerze z poprzedniego podrozdziału. Do jednego katalogu skopiujmy ten plik i narzędzie *kcpsm3.exe* wraz z trzema plikami *ROM_form*. Po wydaniu następującej komendy utworzone zostaną moduły pamięci 1024 słów 18–bitowych, zawierających kod programu PicoBlaze.

Program generuje dosyć dużo informacji i przekierowuje je na standardowe wyjście. Aby bez problemu sprawdzić te informacje, można przekierować standardowe wyjście do pliku:

kcpsm3 prog_1.psm 1>prog_1.txt

Poniżej opisane są pliki generowane przez program *kcpsm3.exe*.

<program>.vhd, <program>.v

Główne pliki wyjściowe zawierające implementację pamięci ROM, wraz z pamięcią kodu PicoBlaze. Forma tych plików jest uzależniona od formy plików *ROM_form*. W miejsca, gdzie w pliku wejściowym znajdują się opisy {*INIT_xx*} zostaną wstawione odpowiednie dane, będące wartościami pamięci ROM, z jakimi będzie zainicjalizowany odpowiedni element pamięci w układzie FPGA. Natomiast wartość {*name*} zostanie zastąpiona nazwą pliku z programem w asemblerze. Wygenerowane pliki są implementacją, odpowiednio w języku VHDL i Verilog. Mogą one być dowolnie zmieniane na potrzeby danego projektu.

<program>.coe

Plik, również z opisem inicjalizacji pamięci ROM, powstaje na podstawie kodu programu w asemblerze. Jest to format wykorzystywany przez program *Core Generator*.

<program>.hex, <program>.dec

Odpowiednio: szesnastkowy i dziesiętny format pamięci programu. Każda linijka tego pliku jest kolejną linią programu dla PicoBlaze. Pliki te mogą być zastosowane m.in. w różnych narzędziach do symulacji.

<program>.fmt

Sformatowany wejściowy plik z kodem programu w asemblerze.

<program>.log, constant.txt, labels.txt, pass[1–5].dat

Pierwszy plik jest raportem, w jaki sposób kod asemblera został przetłumaczony na kod maszynowy. Następny zawiera spis wszystkich stałych, zaś ostatni wszystkich nazw, które określają miejsca w pamięci programu. Pliki te wraz z pięcioma plikami *passx.dat* pozwalają na łatwiejsze odnajdywanie błędów w asemblerze.

Narzędzie *kcpsm3.exe* zatrzymuje swoje działanie natychmiast po wykryciu pierwszego błędu.

Składnia programu asemblerowego

Program asemblerowy, który jest obsługiwany przez narzędzie *kcpsm3.exe* musi być odpowiednio sformatowany. Dyrektywy i nazwy instrukcji nie są zależne od wielkości liter. Dodatkowo, aby umożliwić przenoszenie kodu pomiędzy różnymi systemami operacyjnymi i ich wersjami językowymi powinno unikać się stosowania nazw, zawierających litery spoza podstawowego zestawu znaków ASCII.

Rejestry i stałe

PicoBlaze zawiera 16 rejestrów ogólnego przeznaczenia: sX, gdzie X jest numerem rejestru w zapisie heksadecymalnym (od 0 do 9 i dalej od A do F). W instrukcjach, które wykorzystują adresowanie względne,

za pomocą rejestru, są one dodatkowo brane w „okrągłe” nawiasy.

Stałe są reprezentowane tylko w zapisie heksadecymalnym i tylko o wartościach od 00 do FF. Należy zawsze podać dwie pozycje, np. wartość dziesiętną 8, należy zapisać jako 08.

Nazwy adresów

W programie asemblerowym z reguły nie podaje się wartości adresów w pamięci programu. Wszystkie miejsca takie jak skoki i procedury, opisuje się nazwami. Nazwę umieszcza się przed instrukcją i poprzedza się ją dwukropkiem, np:

nazwa :

W trakcie wykonywania program *kcpsm3.exe* przypisuje nazwom konkretne adresy w pamięci programu.

Komentarze

Z reguły komentarz w asemblerze występuje za średnikiem. Znak ten może zostać umieszczony w dowolnym miejscu. Cały tekst od średnika do końca linii jest traktowany jako komentarz. W niniejszym artykule komentarz będzie wyróżniany kursywą. Dodatkowo, w programach asemblerowych, warto komentować każdą instrukcję, gdyż przyczynia się to do łatwiejszego zrozumienia kodu.

Umieszczanie kodu pod specyficznym adresem

Dyrektywa *ADDRESS*, służy do umieszczenia kodu pod specyficznym adresem pamięci programu. Dyrektywa ta jest użyteczna przy opisywaniu wektora obsługi przerwań. Zapis dyrektywy w dowolnym miejscu programu, np.:

ADDRESS 3FF

spowoduje, że od tego miejsca jakakolwiek instrukcja asemblerowa zostanie umieszczona w pamięci programu pod adresem 0x3FF (dziesiętnie 1023).

Zmiana nazwy rejestrów

Ponieważ w niektórych sytuacjach warto wykorzystać niektóre rejestry do zadań specjalnych, można zmienić ich nazwę za pomocą dyrektywy *NAMEREG*, np.:

NAMEREG s9, moja_nazwa

Od miejsca umieszczenia tej dyrektywy w kodzie nie będzie już dostępna nazwa rejestru s9, tylko ‘*moja_nazwa*’. W asemblerze można dowolnie używać zmiany nazwy tego samego rejestru. Zmiana nazwy rejestru pozwala uniknąć błędów wykorzystania danego rejestru do zastosowań innych niż w danym programie zostało to założone.

Definicje stałych

Stałe pozwalają ujednolicić kod programu. Dodatkowo łatwiej jest zmienić wartość stałej w jednym miejscu, niż we wszystkich miejscach jej użycia. Za pomocą stałej możemy nadać nazwy wybranym wartościom liczbowym, np.:

CONSTANT moja_stala, kk

Po takiej deklaracji w programie asemblerowym można używać nazwy ‘*moja_stala*’, wszędzie tam, gdzie potrzeba podać wartość heksadecymalną *kk*.

Marcin Nowakowski