Język Verilog w przykładach (3)

Implementacja pamięci RAM w układach FPGA

Pamięć RAM jest najczęściej używana w aplikacjach akwizycji danych, chociaż nie tylko – na przykład można w niej przechowywać współczynniki filtrów lub próbki przetwarzanego sygnału. W tym odcinku kursu zajmiemy się właśnie implementacją pamięci RAM w strukturze FPGA. Ze względu na budowę pamięci te dzieli się na jednoportowe i dwuportowe. Różnią się one liczbą wejść adresowych i sposobem dostępu.

List. 15. Jednoportowa pamięć RAM z priorytetem zapisu

```
module pamiec
#(parameter dlength=18, parameter depth=64, parameter add_length=6)
(clk,en,wr,din, dout, address);

input clk,wr,en;
input [dlength-1:0] din;
output reg [dlength-1:0] dout;     //registered output
input [add_length-1:0]address;
reg [dlength-1:0] memory [depth-1:0]; //memory

always@(posedge clk)
if (en)
        if (wr) begin
                memory[address] <= din;
                dout <= din;
        end else
                dout <= memory[address];
endmodule
```

Tab. 2. Rodziny układów FPGA firmy XILINX z blokami pamięci

Rodzina:	Dostępność bloków RAM
Spartan-3 Virtex-II, II Pro, II Pro X Virtex-4	Tak (tryby: write-first, read-first, no-change)
Virtex, Virtex-E, Spartan II, IIE	Tak (tylko w trybie write-first)

Pamięć RAM może być realizowana w układach FPGA na dwa sposoby: z użyciem funkcjonalnych zasobów logicznych układu (*distributed RAM*) lub wbudowanych pamięci (*block RAM*) dostępnych w niektórych układach. Użycie gotowych bloków pamięci RAM jest łatwiejsze i efektywniejsze niż realizowanie jej w logice rozproszonej, dlatego skupimy się właśnie na tym rozwiązaniu. W **tab. 2** przedstawiono rodziny układów firmy XILINX z blokami pamięci *block RAM*.

Można opisać pamięć RAM strukturalnie korzystając z elementów bibliotecznych, ale jest to niepraktyczne, dosyć czasochłonne i mało efektywne. Przy opisie behawioralnym pamięci należy przestrzegać ściśle określonych reguł – opis musi być zgodny z możliwościami danej technologii, a niestety zazwyczaj różni się ona między układami różnych producentów. W tym artykule zastosujemy opis odpowiedni dla układów firmy XILINX.

Zaprojektujmy zatem jednoportową pamięć RAM do realizacji w układzie FPGA (**list. 15**). Będzie ona pracować w trybie z priorytetem zapisu, ponieważ właśnie ten tryb jest najbardziej rozpowszechniony (tab. 2). Tryb priorytetu określa stan wyjścia pamięci w momencie zapisu nowej danej. W celu większej uniwersalności modelu sparametryzujemy go, zarówno ze względu na długość słowa (parametr *dlength*) jak i pojemność pamięci (parametr *depth*). Należy również sprecyzować szerokość magistrali adresowej (*add_lenght*) – zależy ona oczywiście od pojemności pamięci. Dodatkowo dodano w opisie sygnał zezwolenia *en*. Charakterystycznymi cechami opisu pamięci tego rodzaju jest wyjście z dodatkowym rejestrem (*dout*) oraz to, że w czasie zapisu wyjście przyjmuje wartość nowej danej. Pamiętajmy, że pojemność pamięci w każdym układzie jest ograniczona. Przykładowo, układ Spartan 3E XC3S500E zawiera w swojej strukturze 20 bloków pamięci po 18 kb każda, co daje w sumie 360 kb dostępnej pamięci. Należy o tym pamiętać na etapie projektowania urządzenia i w razie potrzeby dołączyć do układu FPGA pamięć zewnętrzną.

Jak rozpoznać, czy implementowana pamięć (jej opis) została prawidłowo rozpoznana przez program syntezy? Sięgnijmy do takich narzędzi narzędzi: *synthesis report* i *RTL schematic*. Jeśli wszystko jest w porządku, powinniśmy znaleźć w raporcie z syntezy fragmenty informujące o rozpoznaniu struktury pamięci i użyciu *block RAM'u* do jej implementacji (**list. 16**). Gdybyśmy jednak w tym przykładzie zrezygnowali z rejestrowego wyjścia i użyli ciągłego przypisania (*assign dout = memory[address];*), wtedy użycie *block RAM'u* byłoby niemożliwe i w raporcie znaleźlibyśmy następujący zapis informujący o zaimplementowaniu pamięci rozproszonej:

INFO:Xst:2664 – HDL ADVISOR – Unit <pamiec> : The RAM <Mram_memory> will be implemented on LUTs either because you have described an asynchronous read or because of currently unsupported block RAM features. (…)

Pamięć jest taktowana sygnałem zegarowym *clk*. Jeśli sygnał *wr* ma poziom niski, to port wyjściowy pamięci przyjmuje wartość komórek (bajtu) wskazywanych przez adres (*address*). Zwróćmy jednak uwagi, że właściwa wartość zostanie wystawiona na port dopiero w następnym cyklu zegara. Poziom

List. 16. Fragmenty raportu z syntezy przy prawidłowej implementacji pamięci z użyciem *block RAM*

```
Synthesizing Unit <pamiec>.
    Related source file is „pamiec.v”.
    Found 64x18-bit single-port RAM <Mram_memory> for signal <memory>.
    Found 18-bit register for signal <dout>.
    Summary:
        inferred   1 RAM(s).
        inferred  18 D-type flip-flop(s).
Unit <pamiec> synthesized.
(...)
INFO:Xst:2691 - Unit <pamiec> : The RAM <Mram_memory> will be implemented as
a BLOCK RAM, absorbing the following register(s): <dout>.
    ---------------------------------------------------------------------
    | ram_type           | Block                                |          |
    ---------------------------------------------------------------------
    | Port A                                                    |          |
    |     aspect ratio   | 64-word x 18-bit                     |          |
    |     mode           | write-first                          |          |
    |     clkA           | connected to signal <clk>            | rise     |
    |     enA            | connected to signal <en>             | high     |
    |     weA            | connected to signal <wr>             | high     |
    |     addrA          | connected to signal <address>        |          |
    |     diA            | connected to signal <din>            |          |
    |     doA            | connected to signal <dout>           |          |
    ---------------------------------------------------------------------
    | optimisation       | speed                                |          |
    ---------------------------------------------------------------------
```

List. 17. Inicjalizacja *block RAM'u* – sposób 1

```
integer index;
initial begin
        for (index=0; index <= depth-1 ; index=index + 1)
                memory[index] = index;
end
```

List. 18. Inicjalizacja *block RAM'u* – sposób 2

```
initial begin
        $readmemh(„wspolczynniki.dat”,memory);
end
```

List. 19. Inicjalizacja przerzutnika

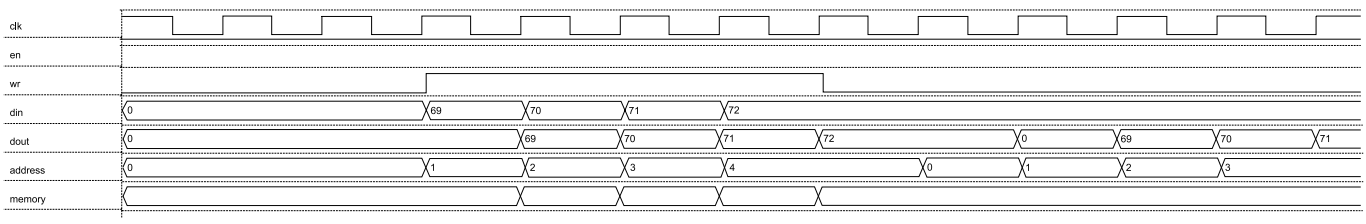
```
reg ff = 1'b1;  //stan po włączeniu
always@(posedge clk or negedge reset)
        if (!reset)
                ff <= 1'b0; //reset jawny
        else
                ff <= d;
```

wysoki na linii *wr* spowoduje, że komórki pamięci wskazywane przez adres przyjmą wartość z portu wejściowego, natomiast wyjście przyjmie wartość nowo wpisanej danej.

W wielu aplikacjach jest potrzebne, aby po włączeniu urządzenia w pamięci były już wpisane określone dane. W przypadku starszych układów FPGA, inicjalizacja pamięci RAM z jakimiś danymi nie jest możliwa w prosty sposób. Należy bowiem zapisać całą pamięć wykonując po kolei zwyczajne operacje dostępu do pamięci. W nowszych układach dostępny jest mechanizm pozwalający na dokonanie takiej inizjalizacji na etapie programowania FPGA – a więc z każdym włączeniem zasilania lub przeprogramowaniem. Wystarczy opisać to w kodzie źródłowym, przy czym można to zrobić dwoma sposobami: specyfikując zawartość każdej komórki (wartości poszczególnych bajtów) w kodzie źródłowym opisu pamięci lub wskazać plik zawierający żądaną zawartość pamięci. Na **list. 17** zamieszczono przykład inicjalizacji pamięci kolejnymi wartościami od 0 do 63. Należy go dodać do kodu opisu pamięci. Warto zwrócić uwagę, że jest to konstrukcja wyjątkowa – człony typu *initial... begin* są niesyntezowalne, jednak w tym przypadku program syntezy rozpoznaje, że chodzi o inicjalizację RAM'u i nie zgłasza błędu. W przypadku inicjalizowania każdego wiersza komórek (bajtu) z osobna, można to zrobić używając również w ramach bloku *initial.. begin* konstrukcji typu: *memory[0]=18'd257*;. Dużo praktyczniejsze jest jednak załadowanie tych wartości z pliku wygenerowanego np. z poziomu MATLAB'a. Na **list. 18** przedstawiono ten wariant inicjalizacji pamięci. Instrukcja *$readmemh* interpretuje zawartość wskazywanego pliku tekstowego jako wartości w formacie szesnastkowym i podstawia je do pamięci *memory*. Plik powinien zawierać kolejne wartości oddzielone od siebie np. nową linią. Liczba danych w pliku nie może przekraczać rozmiaru pamięci. Instrukcje ta oraz bliźniacza *$readmemb*, są poza wymienionym zastosowaniem przydatne przy tworzeniu *testbench'y* jako źródło sygnałów testowych, np. do symulacji przetwornika A/C. Jeśli nie wiemy czy układ FPGA ma możliwość interpretacji takiego zapisu, to można łatwo sprawdzić. Należy opisać pamięć wraz z inicjalizacją, a następnie zsyntezować projekt dla docelowego układu. Następnie otwieramy *Technology Schematic* i klikamy na blok (bloki) zsyntetyzowanej pamięci. We właściwościach tych bloków poszukujemy wartości parametrów INIT – jeśli są takie, to ta technologia jest dostępna w układzie. Inicjalizować można również przerzutniki. W układach FPGA zaraz po zaprogramowaniu, a zatem również po załączeniu zasilania, jeśli używamy pamięci konfigurującej, to generowany jest wewnętrzny sygnał resetujący (*power-on reset*). Można jednak zdefiniować swój własny, jawny sygnał resetu (np. pochodzący z przycisku). W przerzutnikach można sprecyzować dwa stany inicjalizacji: zewnętrzny reset i wewnętrzny, automatyczny reset po zaprogramowaniu. Jednak również ta technologia może nie być dostępna we wszystkich układach programowalnych. Przykładowy kod rejestru z opisanym stanem początkowym przedstawiono na **list. 19**.

Sprawdźmy jak działa zaprojektowana pamięć. Na **rys. 11** przedstawiono przebiegi czasowe sygnałów podczas symulacji obsługi pamięci. Pamięć była zainicjalizowana w sposób przedstawiony na list. 17, a testbench został przygotowany tak, by od 60 nanosekundy nastąpiły cztery wpisy do pamięci pod adresy: 1, 2, 3 i 4 wartości odpowiednio: 69,70, 71 i 72. Po zakończeniu wpisu następuje odczyt zawartości czterech kolejnych bajtów pamięci, począwszy od adresu 0. Pamięć zachowuje się zgodnie z oczekiwaniami, a zatem zgodnie z regułą *write-first*.

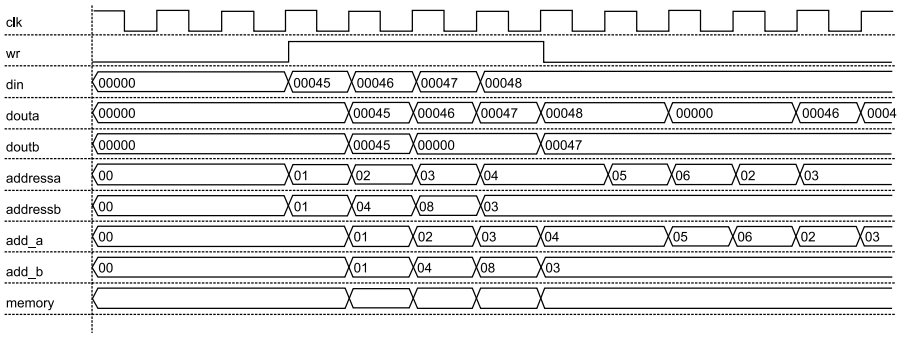
Pamięć jednoportowa ma poważne ograniczenie: możliwe jest wykonanie tylko jedną czynności w danym momencie. Czasem istnieje potrzeba jednoczesnego zapisu pamięci i jej odczytu przy użyciu różnych adresów. Pamięci dwuportowe mają właśnie taką właściwość. Z racji największej popu-



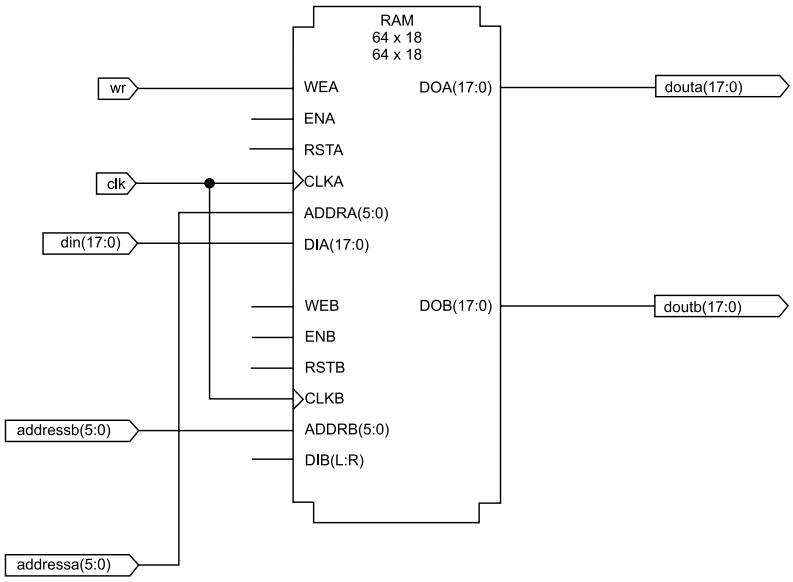
Rys. 11. Przebiegi czasowe sygnałów podczas symulacji obsługi pamięci jednoportowej

List. 20. Pamięć dwuportowa

```
module pamiec_dualport
        #(parameter dlength=18, parameter depth=64, parameter add_length=6)
        (clk,wr,din,douta,doutb,addressa,addressb);
        input clk,wr;
        input [dlength-1:0] din;
        output [dlength-1:0] douta,doutb;
        input [add_length-1:0]addressa, addressb;

        //rejestry dla adresów:
        reg [add_length-1:0]add_a,add_b;
        reg [dlength-1:0] memory [depth-1:0]; //pamięć

//inicjalizacja RAM'u
integer index;
initial begin
        for (index=0; index <= depth-1 ; index=index + 1)
                memory[index] = {dlength{1'b0}};
        end

always@(posedge clk)
begin
if (wr)
        memory[addressa] <= din;  //wpis
add_a <= addressa;                                  //przepisanie adresów
add_b <= addressb;
end

//ustawienie wyjść
assign douta = memory[add_a];
assign doutb = memory[add_b];
endmodule
```



Rys. 12. Przebiegi czasowe sygnałów podczas symulacji obsługi pamięci dwuportowej



Rys. 13. Symbol graficzny RTL pamięci dwuportowej (XILINX)

larności zaimplementujmy dwuportową pamięć z synchronicznym odczytem. Pozwala ona na jednoczesny zapis/odczyt na jednym porcie, jak również odczyt z zupełnie innego adresu na drugim porcie. W **list. 20** zawarto opis takiej pamięci razem z jej inicjalizacją zerami. Zwróćmy uwagę na cechy charakterystyczne tego opisu: oba adresy są synchronicznie przepisywane do wewnętrznych rejestrów (*add_a* i *add_b*) i są bazą do ciągłego przypisania obu wyjść, przy czym jeśli następuje wpis (*wr*), to zapisywana jest komórka pamięci wskazywana przez wartość na wejściu adresowym (*addressa*). Podobnie jak w przypadku pamięci jednoportowej, ścisłe przestrzeganie takiego opisu gwarantuje prawidłową implementację z wykorzystaniem *block RAM'u*. W raporcie z syntezy powinniśmy znaleźć zapis podobny, jak w przypadku pamięci jednoportowej, jednak z wyszczególnieniem, że chodzi o pamięć typu *dual-port*. Na **rys. 13** przedstawiono symbol RTL opisanej pamięci. Nawet, gdy pamięć zostanie zaimplementowana w postaci rozproszonej, to widok RTL nadal będzie pokazywał strukturę typu RAM, lecz otoczoną często dodatkowymi strukturami logicznymi. Zatem podstawą kontroli poprawności opisu projektu powinien być raport z syntezy. Na rysunku widać, jak wiele sygnałów pozostało niepodłączonych. W istocie jest wiele konfiguracji, w jakich pamięć może pracować – w tym artykule przedstawione zostały jednak tylko te, naczęściej stosowane.

Spójrzmy na przebiegi czasowe sygnałów uzyskane podczas symulacji opisanej pamięci – **rys. 12**. Początkowo pamięć jest wypełniona zerami. Następnie zapisujemy pod kolejne adresy: 1, 2, 3 i 4, wartości: 45, 46, 47 i 48, jednocześnie odczytując dane przez drugi port spod adresów: 1, 4, 8, 3. Po zakończeniu zapisu dokonane są odczyty danych z różnych adresów z zastosowaniem obu portów. Na przebiegach wyraźnie widać, że oba wejściowe adresy są rejestrowe, a więc wyjścia przyjmują prawidłowe wartości po jednym cyklu zegara. Właściwości dwuportowe są również widoczne, przy czym należy zwrócić uwagę na fakt, że przy jednoczesnym zapisie i odczycie z tego samego adresu, dana wyjściowa przyjmuje wartość aktualnie wpisywaną.

Krzysztof Kasiński
Krzysztof.kasinski@o2.pl
home.agh.edu.pl/kasinski