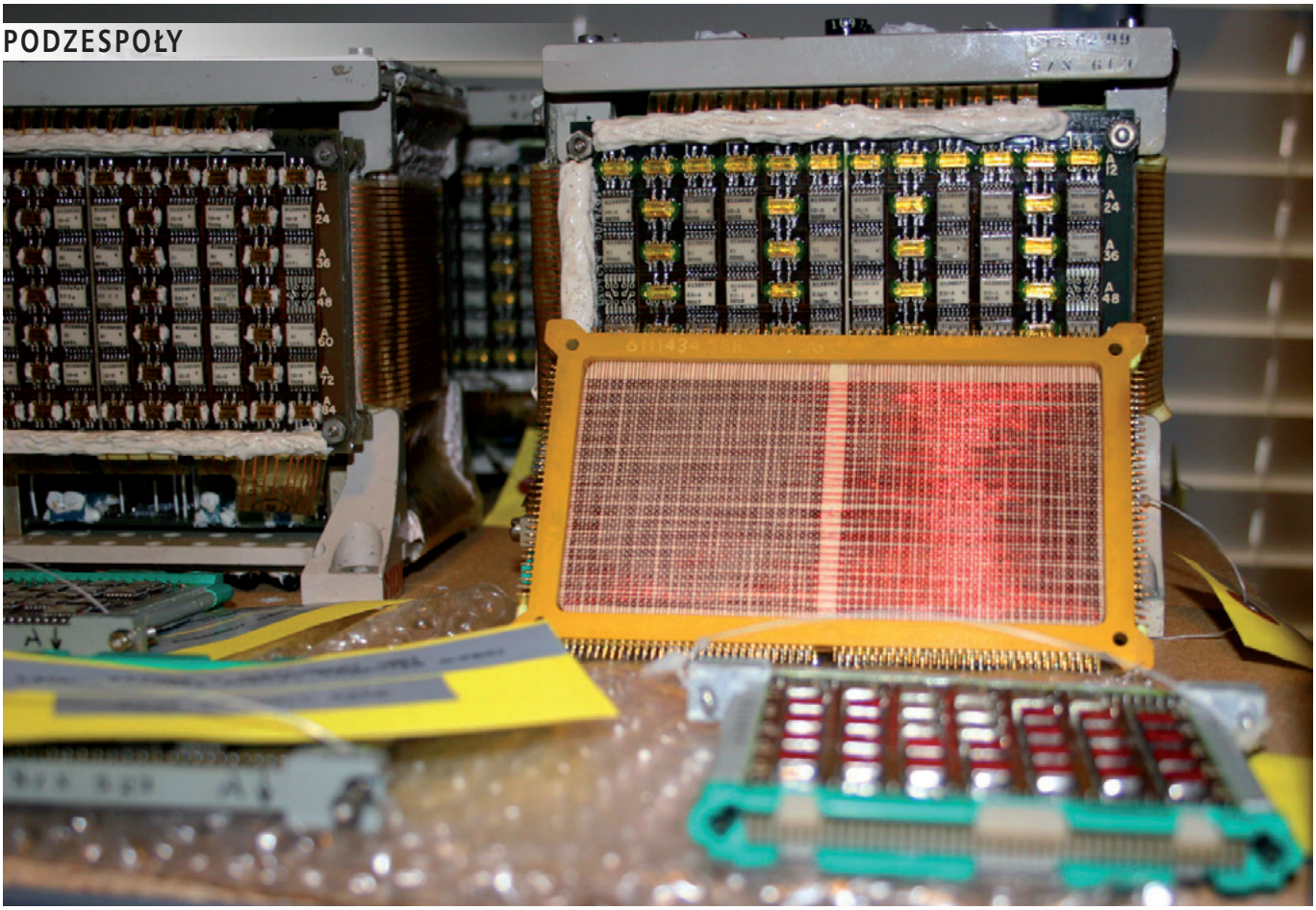


# Siła (w) pamięci on-chip

## Implementacje pamięci w układach Cyclone IV firmy Altera


*Jedną ze sztandarowych cech współczesnych układów FPGA jest możliwość implementacji w nich – dzięki wbudowanym specjalnym zasobom logicznym - bloków szybkich pamięci RAM, CAM, ROM i FIFO, których jedyną „wadą" są ich relatywnie (w stosunku do powszechnie dostępnych w urządzeniach elektronicznych „megabajtów") niewielkie pojemności. Problem ten można bez większego trudu ominąć, a w jaki sposób – pokażemy w EP za miesiąc. Teraz przybliżymy „tajniki" implementacji pamięci w wewnętrznych zasobach FPGA.*

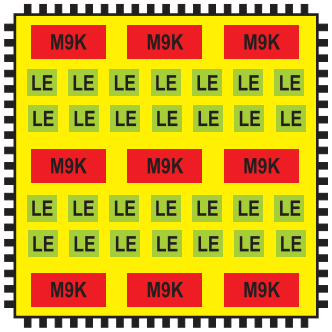
**Dodatkowe informacje:**
Testy sprzętowe niezbędne do opracowania artykułu przeprowadzono na zestawie Cyclone IV GX Transceiver Starter Kit udostępnionym przez firmę EBV Elektronik, tel. 71 342 29 44 oraz 22 640 23 55, www.ebv.com.

Współczesne aplikacje układów FPGA wymagają ich współpracy z szybkimi pamięciami o pojemnościach – często – sięgających dziesiątek, a nawet setek megabajtów. Jeżeli wymagania aplikacji co do pojemności pamięci są mniejsze (do kilkunastu Mb), konstruktorzy mogą korzystać z konfigurowalnych bloków logicznych zoptymalizowanych konstrukcyjnie do implementowania w nich pamięci różnego rodzaju (RAM, ROM, FIFO, CAM, SIPO, PISO itp.). Rozwiązanie takie ma wiele zalet, spośród których warto wspomnieć przede wszystkim: dużą szybkość pracy pamięci wbudowanych w FPGA, łatwość ich integracji w projekcie, minimalizację liczby układów scalonych, ograniczenie emisji EM i poboru prądu oraz uproszczenie płytki drukowanej.

### Pamięci wbudowane w FPGA

W układach Cyclone IV (najnowsza rodzina układów FPGA z serii Cyclone produkowana przez firmę Altera) do implementacji zespołów pamięci przewidziano dwa rodzaje zasobów logicznych (**rysunek 1**):

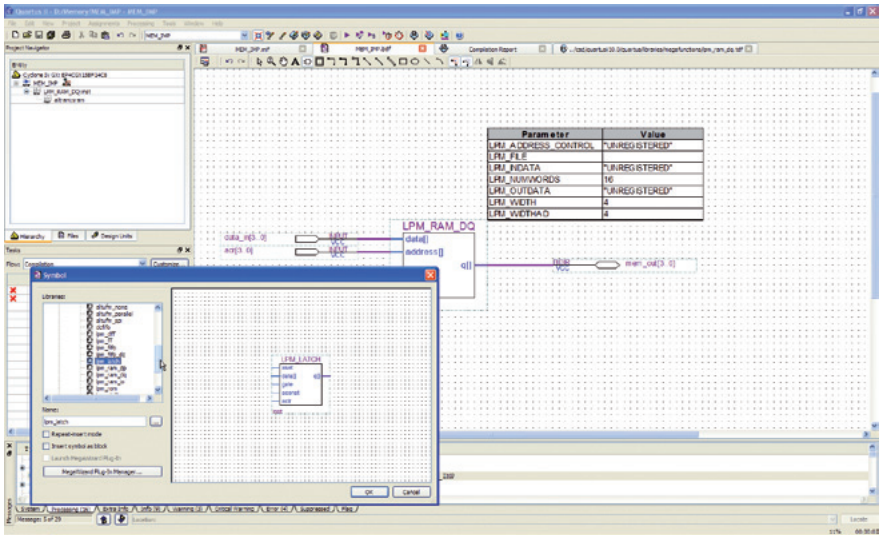
– Konfigurowalne bloki M9K o pojemności 8192 bitów danych oraz 1024 bitów parzystości, co daje łącznie 9216 komórek pamięci. Mogą być one – w zależności od potrzeb – konfigurowane jako



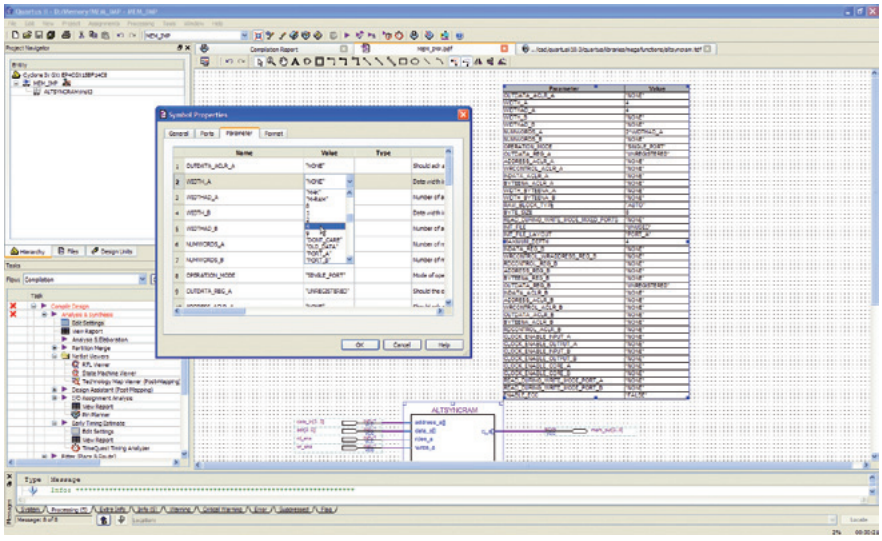

**Rysunek 1. Uproszczona budowa układów FPGA** (pominięto bloki niepotrzebne podczas implementowania bloków pamięci)

pamięci o organizacji 1×8192, 2×4096, 4×2048, 8×1024, 9×1024, 16×512, 18×512, 32×256 lub 36×256 i architekturze ROM, RAM, FIFO, także w wersji dwuportowej. Bloki M9K mogą także być skonfigurowane jako rejestry przesuwające, co pozwala stosować je jako pamięci SIPO i PISO, często stosowane w filtrach FIR (*Finite Impulse Response*). Linie I/O danych pamięci mogą być rejestrowe lub kombinacyjne, użytkownik może mieć także, w pewnym zakresie, wpływ na sprzętowy interfejs pamięci.

– Komórki logiczne *Logic Element* (LE) wyposażone w konfigurowalny przerzutnik LE/T/D/JK oraz 4-wejściową komórkę LUT (*Look-up Table*), która może spełniać rolę samoadresowalnej komórki ROM. Implementacja w LE pamięci o większych (począwszy od kilkunastu bajtów) pojemnościach nie jest racjonalna, bowiem są one (ze względu na fizyczne rozproszenie komórek w strukturze półprzewodnikowej) wolniejsze od pamięci implementowanych w wyspecjalizowanych blokach M9K, pochłaniają ponadto dużo, dość kosztownych, uniwersalnych zasobów FPGA.



**Rysunek 2. Okno edytora schematów w pakiecie Quartus II z wyświetlanym przykładowym symbolem bibliotecznym wymagającym parametryzacji**



**Rysunek 3. Ze względu na dużą liczbę parametrów, konfiguracja megafunkcji *altsyncram* wymaga sporego nakładu pracy i może sprawić początkującym trudności**

W **tabeli 1** znajduje się zestawienie najważniejszych zasobów układów Cyclone IV GX, z której to rodziny wykorzystano w aplikacjach testowych układ EP4CGX15.

Korzystanie z pamięci wbudowanych w FPGA w projektach przygotowywanych za pomocą pakietu Quartus II jest możliwe na wiele sposobów, z których – zwłaszcza w przypadku większych projektów – najwygodniejszy jest edytor schematów i udostępnione przez Alterę biblioteki z predefiniowanymi blokami pamięci. We wcześniejszych wersjach pakietów projektowych firmy Altera do implementacji bloków pamięci stosowane były parametryzowane bloki logiczne LPM (*Library Parametrized Module* – **rysunek 2**), zwane także megafunkcjami, jak np. *lpm_ram_dp* (dwuportowa pamięć RAM) lub *lpm_ram_io* (pamięć z dwukierunkowymi liniami danych), które obecnie zastępuje uniwersalna megafunkcja *altsyncram*. W pakiecie Quartus II 10.0 jest ona automatycznie podstawiana podczas syntezy logicznej w miejsce megafunkcji *lpm_* i konfigurowana w taki sposób, żeby użytkownik nie musiał samodzielnie modyfikować nastaw w kompilowanym projekcie.

## Pamięci z megafunkcji

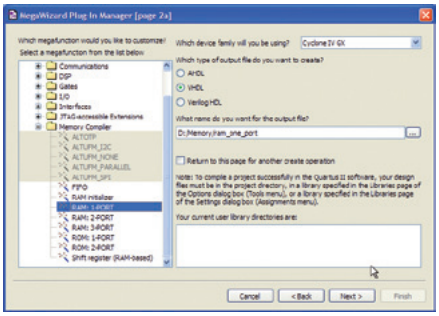
Korzystanie z megafunkcji we własnych projektach wymaga pewnej wprawy, bowiem każdorazowo wymagają one konfiguracji przez użytkownika, co w przypadku początkujących konstruktorów może sprawić nieco kłopotów ze względu na szeroką gamę możliwych opcji, co dość dobrze zilustrowano na **rysunku 3**. Na szczęście twórcy pakietu Quartus II nie zignorowali takich potrzeb

**Tabela 1. Zestawienie wybranych zasobów układów Cyclone IV GX**

| Cecha | EP4CGX15 | EP4CGX22 | EP4CGX30 | EP4CGX50 | EP4CGX75 | EP4CGX110 | EP4CGX150 |
|---|---|---|---|---|---|---|---|
| Liczba bloków LE | 14400 | 21280 | 29440 | 49888 | 73920 | 109424 | 149760 |
| Liczba bloków M9K | 60 | 84 | 120 | 278 | 462 | 610 | 720 |
| Pojemność pamięci w blokach M9K [kB] | 540 | 756 | 1,080 | 2,502 | 4,158 | 5,490 | 6,480 |

**Całkiem bezpłatnie**

Może się to wydawać niewiarygodne, ale zarówno pakiet Quartus II jak i zestaw bibliotek, których niewielki fragment przedstawiliśmy w artykule, są dostępne bezpłatnie na stronie www.altera.com. Pakiet Quartus II w wersji 10.0 opublikowaliśmy na płycie DVD-EP9/2010C, którą otrzymali bezpłatnie wszyscy prenumeratorzy EP. Obecnie na stronie Altery jest dostępna stabilniejsza wersja pakietu zintegrowana z SP1.

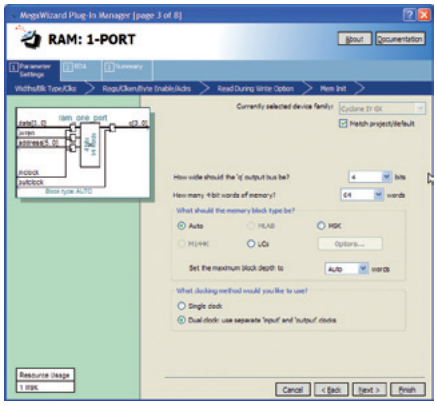


**Rysunek 4. Pierwszy krok MegaWizarda podczas parametryzowania megafunkcji *altsyncram***

i wyposażyli go w kreator elementów bibliotecznych (*MegaWizard Plug-in Manager*), który w przyjazny sposób prowadzi projektanta przez meandry konfiguracji predefiniowanych megafunkcji. Dzięki kreatorowi użytkownik może wybrać megafunkcję do parametryzacji i określić w jakim języku

**Listing 1. Przykładowy plik **.mif* z zawartością pamięci RAM ustawianą po włączeniu zasilania**

```
-- Quartus II generated Memory
Initialization File (.mif)

WIDTH=4;
DEPTH=64;

ADDRESS_RADIX=UNS;
DATA_RADIX=UNS;

CONTENT BEGIN
        0   :   12;
        1   :   15;
        2   :   9;
        [3..4]  :   0;
        5   :   9;
        6   :   2;
        [7..10]  :   0;
        11  :   11;
        12  :   0;
        13  :   8;
        14  :   3;
        15  :   15;
        16  :   4;
        [17..18]  :   0;
        19  :   11;
        20  :   0;
        21  :   7;
        22  :   4;
        23  :   15;
        24  :   4;
        [25..26]  :   0;
        27  :   11;
        28  :   0;
        29  :   6;
        30  :   5;
        31  :   15;
        32  :   4;
        [33..34]  :   0;
        35  :   11;
        36  :   0;
        37  :   5;
        38  :   6;
        39  :   15;
        40  :   4;
        [41..42]  :   0;
        43  :   11;
        44  :   0;
        45  :   4;
        46  :   7;
        47  :   15;
        48  :   3;
        [49..50]  :   0;
        51  :   11;
        52  :   0;
        53  :   3;
        54  :   8;
        55  :   15;
        56  :   3;
        [57..58]  :   0;
        59  :   11;
        60  :   0;
        61  :   2;
        62  :   9;
        63  :   15;
END;
```



**Rysunek 5. Kolejny etap parametryzacji megafunkcji *altsyncram***

HDL ma zostać utworzony jej model (**rysunek 4**). Na potrzeby testowe została wybrana megafunkcja jednoportowej, dwubramowej, synchronicznej pamięci RAM (**rysunek 5**), która będzie wyposażona w 4-bitowe bramy wejściową i wyjściową i miała głębokość 64 takich pół-bajtów. Oczywiście organizacja pamięci może być inna (we wspomnianym wcześniej zakresie), a skonfigurowane pamięci można łączyć w zespoły o dowolnej organizacji.

Jak widać na rysunku 5, użytkownik może wybrać sposób implementacji w FPGA parametryzowanej megafunkcji:

- można wymusić jej implementację w blokach M9K lub
- wymusić jej implementację w rozproszonych komórkach logicznych LC (*Logic Cells* – odpowiedniki LE w układach Cyclone IV).

W zależności od przyjętego sposobu implementacji bloku pamięci – zgodnie z intuicyjnymi oczekiwaniami – jest on umieszczany w którymś z bloków M9K (co widać na **rysunku 6**) lub w uniwersalnych komórkach logicznych (**rysunek 7**). Na rysunkach tych pokazano także szczegółowy podgląd sposobu implementacji obydwu wersji pamięci w strukturze układu Cyclone IV oraz fragment wyników kompilacji projektu, z których wyraźnie widać różnice wykorzystanych zasobów.

Na **rysunku 8** pokazano schemat implementowanego projektu z zastosowaną megafunkcją *mem_one_port_le*, którą utworzono

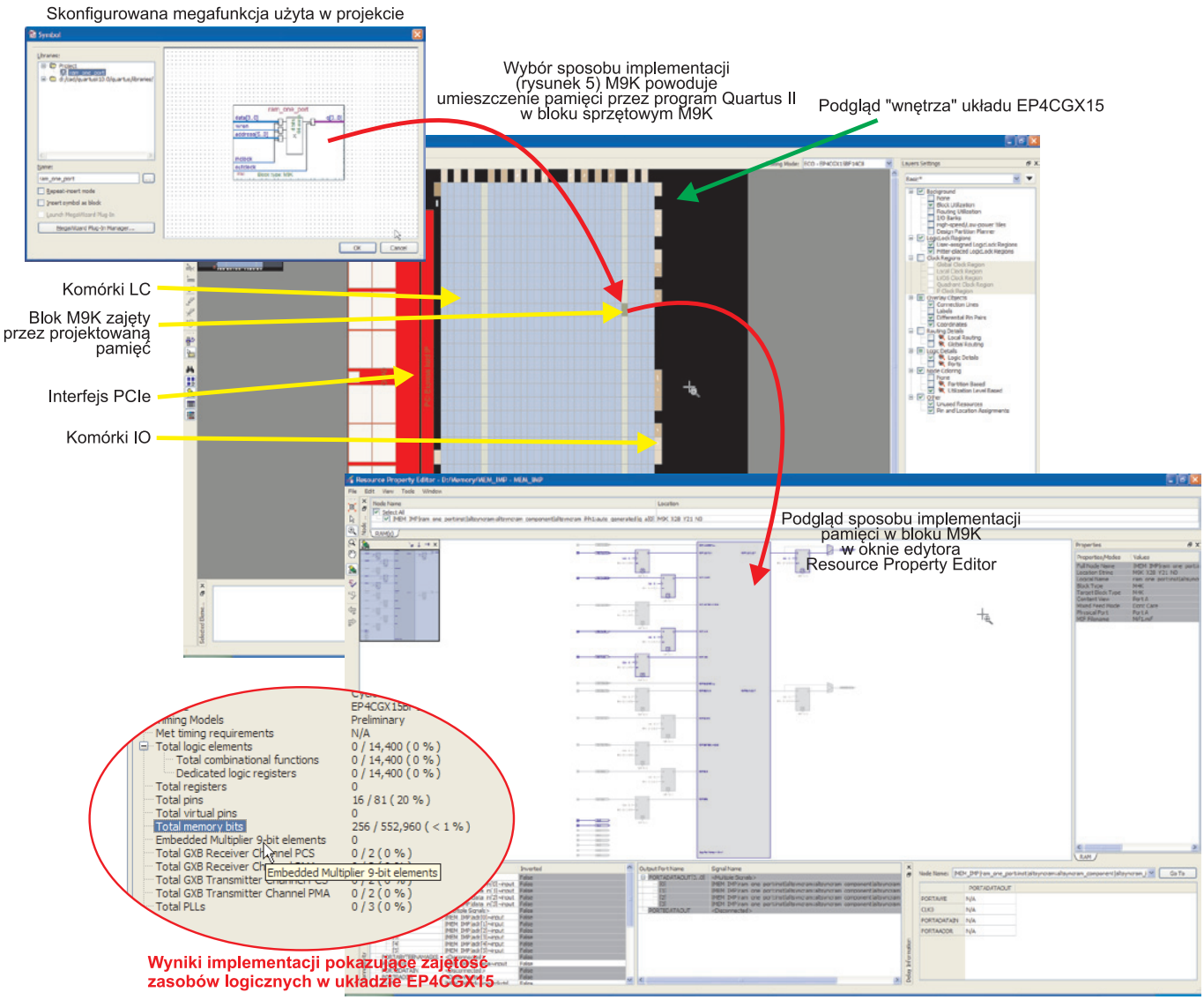
**Nie chcesz rysować, wolisz HDL?**
To nie problem! Dla wszystkich megafunkcji poddawanych parametryzacji Quartus II generuje komplety plików w językach Verilog, VHDL lub AHDL. Można z nich korzystać jak z funkcji – przykład w VHDL dla *mem_one_port_le* pokazano poniżej:

```
FUNCTION ram_one_port_le
(
        address[5..0],
        data[3..0],
        inclock,
        outclock,
        wren
)

RETURNS (
        q[3..0]
);
```

**Wstępna inicjalizacja pamięci...**
...po włączeniu nie jest możliwa w przypadku implementowania jej w uniwersalnych komórkach logicznych. Pliki **.mif* (*Memory Initialization File*) lub standardowe **.hex* będą przydatne wyłącznie w przypadku pamięci implementowanych w blokach M9K.

**Listing 2. Jeden z alternatywnych opisów HDL pamięci RAM o organizacji 64x8, udostępniony przez firmę Altera**

```
library ieee;
use ieee.std_logic_1164.all;

entity single_port_ram is
     port
 (
          data   : in std_logic_vector(7 downto 0);
          addr   : in natural range 0 to 63;
          we     : in std_logic := ,1';
          clk    : in std_logic;
          q      : out std_logic_vector(7 downto 0)
     );

end entity;

architecture rtl of single_port_ram is
     subtype word_t is std_logic_vector(7 downto 0);
     type memory_t is array(63 downto 0) of word_t;

     signal ram : memory_t;
     signal addr_reg : natural range 0 to 63;

begin
     process(clk)
     begin
        if(rising_edge(clk)) then
          if(we = ,1') then
                    ram(addr) <= data;
          end if;

          addr_reg <= addr;
        end if;
     end process;

     q <= ram(addr_reg);

end rtl;
```




**Rysunek 6. Ilustracja skutków implementacji wykreowanej megafunkcji w blokach M9K**

**FPGA lubią synchronizm**

Zdziwienie może budzić taktowanie bram: wejściowej i wyjściowej pamięci zastosowanej w przykładowym projekcie. Wynika ono z konieczności synchronizowania projektów implementowanych w układach FPGA, co zapobiega zaburzeniom w ich pracy wywołanych opóźnieniami w propagacji sygnałów pomiędzy poszczególnymi fragmentami projektu.

na potrzeby projektu za pomocą kreatora MegaWizard jako wariant megfunkcji *altsyncram*. Projekt wygląda identycznie w przypadku zastosowania zamiast *mem_one_port_le* megafunkcję *mem_one_port*, która jest przeznaczona do implementacji w blokach M9K, a nie w uniwersalnych komórkach logicznych jak *mem_one_port_le*.

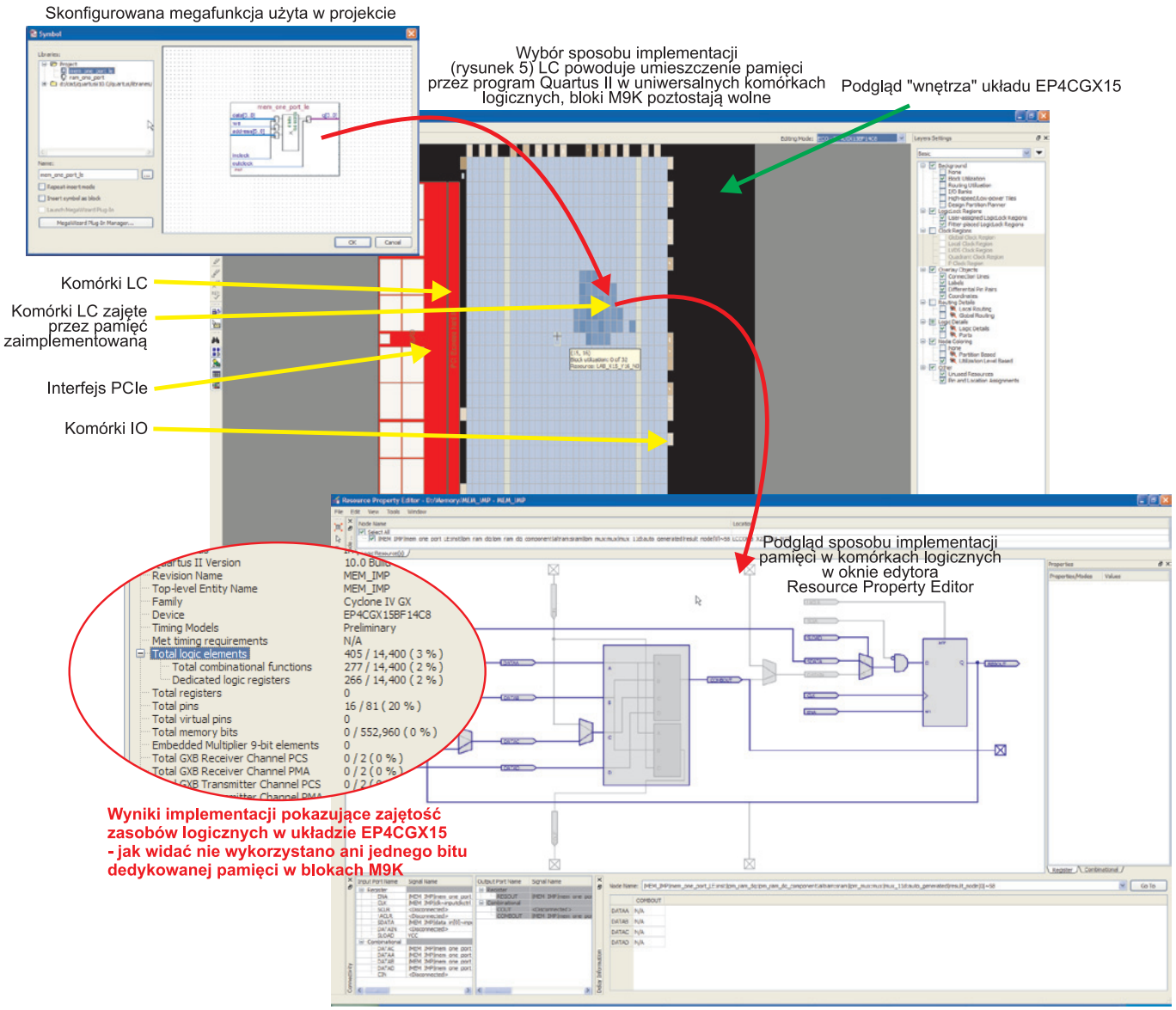
Wróćmy do prezentacji kolejnych etapów pracy kreatora-konfiguratora megafunkcji (ostatni z dotychczas omówionych etapów pokazano na rysunku 5). W oknie pokazanym na **rysunku 9** jest możliwy wybór synchronicznego lub asynchronicznego sposobu pracy linii danych wyjściowych pamięci, wybór sygnałów blokujących taktowanie od strony wejścia i/lub wyjścia, a także innych sygnałów pomocniczych. W kolejnym etapie (**rysunek 10**) projektant może skonfigurować zachowanie pamięci podczas odczytu komórki, która jest w tym samym czasie zapisywana, następnie – ale tylko w przypadku pamięci implementowanej w blokach M9K – jest ustalane zachowanie pamięci po włączeniu zasilania. Pakiet Quartus II umożliwia zapisanie w implementowanej pamięci inicjalnej zawartości (**rysunek 11**), która jest przechowywana w pliku **.mif* (*Memory Initialization File*), którego przykładową zawartość, zastosowaną w prezentowanym projekcie, pokazano na **listingu 1**. Zawartości pliku nie trzeba tworzyć ręcznie, do tego celu można wykorzystać edytor wbudowany w pakiet Quartus II, którego fragment okna edycyjnego pokazano na **rysunku 12**. Zamiast plików **.mif* do inicjalizacji zawartości pamięci można wykorzystywać także klasyczne pliki **.hex* utworzone za pomocą dowolnego programu narzędziowego.

Po zakończeniu konfiguracji (parametryzacji) megafunkcji system projektowy generuje pliki niezbędne do stosowania sparametryzowanych megafunkcji w projektach i samoczynnie dodaje je do zestawu elementów bibliotecznych (symboli) dostępnych dla projektanta (**rysunek 13**). Można z nich korzystać także podczas przygotowywania innych projektów.

W podobny – do przedstawionego – sposób konfigurowania megafunkcji można zastosować w przypadku implementacji bloków pamięciowych o innych cechach: liczbie portów, sposobie synchronizacji, zasadzie działania (cechach funkcjonalnych) itp. Opcje dostępne na poszczególnych etapach konfigurowania mogą być inne niż pokazane, ale generalna zasada jest zawsze taka sama.

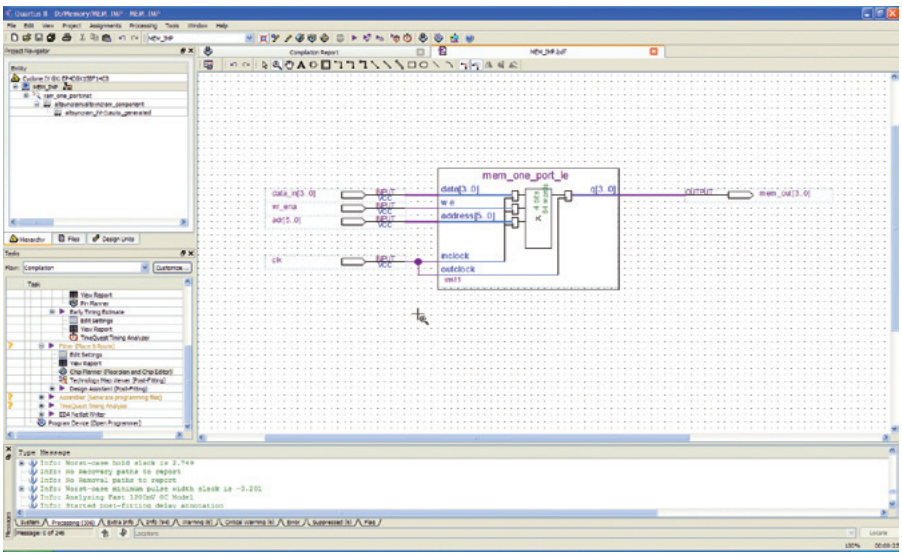
## Inne możliwości

Przedstawione w artykule sposoby utworzenia pamięci w układach FPGA firmy Altera nie są jedynymi możliwymi – projektant może samodzielnie opisać (np. za pomocą któregoś z języków HDL) dowolną pamięć i zaimplementować ją w FPGA wykorzystując pakiet Quartus II, podobnie jak dzieje się to z innymi blokami sprzętowymi implementowanymi




**Rysunek 7. Ilustracja skutków implementacji wykreowanej megafunkcji w uniwersalnych komórkach logicznych**

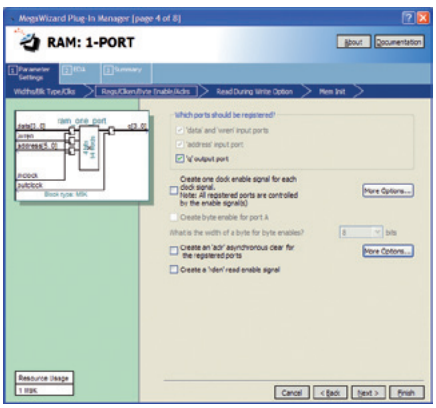
w układach cyfrowych. Przykładowe opisy różnych typów pamięci przygotowali i umieścili na firmowej stronie internetowej (w dziale *Support–>Design Examples–>On-Chip Memory*) inżynierowie firmy Altera. Jeden z udostępnionych opisów (pamięć RAM o organizacji 64×8) pokazano na **listingu 2**.



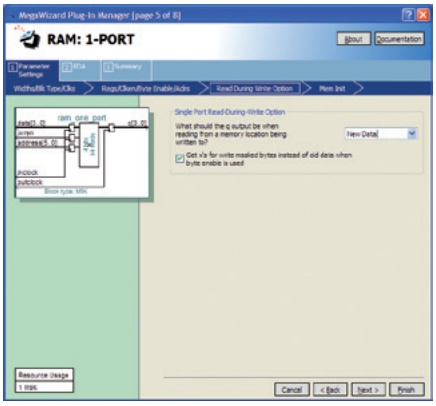

**Rysunek 8. Schemat projektu implementowanego w FPGA z megafunkcją *mem_one_port_le***

## Na zakończenie

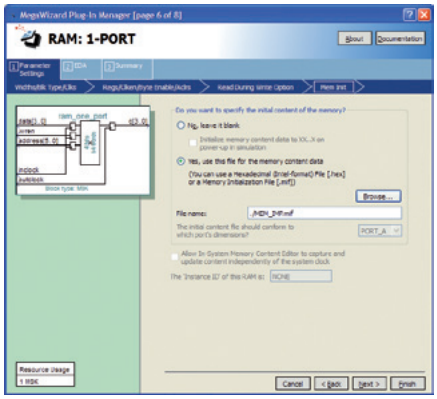
Zasoby lokalnych pamięci dostępnych w układach FPGA wystarczają w przypadku




**Rysunek 9. Na tym etapie parametryzacji megafunkcji *altsyncram* projektant może skonfigurować zachowanie wyjść pamięci podczas zapisu odczytywanej komórki**




Rysunek 10. Kolejny etap parametryzacji megafunkcji *altsyncram*

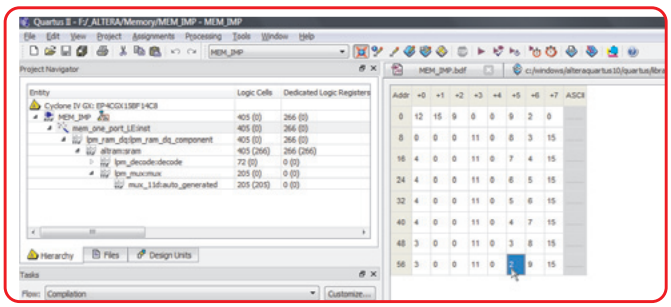



Rysunek 11. W przypadku implementowania pamięci w blokach M9K można określić stany jej komórek po włączeniu zasilania

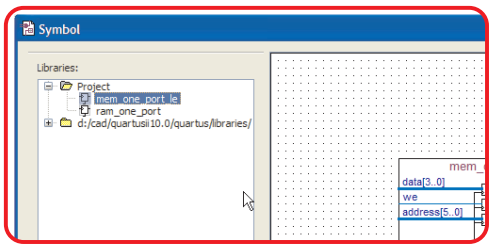
większości aplikacji cyfrowych, ich powiększenie będzie na pewno niezbędne w przypadku implementacji rdzenia mikrokontrolera lub mikroprocesora (jak choćby alterowskiego NIOS II). Twórcy „współczesnych" rodzin FPGA uwzględnili potrzeby tego typu i wyposażyli w odpowiednie zasoby zarówno same układy jak i programy narzędziowe. O tym jak dołączyć do FPGA pamięć SDRAM lub jeszcze nowocześniejszą napiszemy za miesiąc.

Czytelników zainteresowanych tą tematyką zachęcam do kontaktu – nie tylko w przypadku problemów.

**Piotr Zbysiński, EP**
**piotr.zbysinski@ep.com.pl**




Rysunek 12. Fragment okna edytora plików **.mif*




Rysunek 13. Utworzone (poprzez konfigrację megafunkcji *altsyncram*) elementy biblioteczne są dodawane do bibliotek elementów dostępnych w projekcie