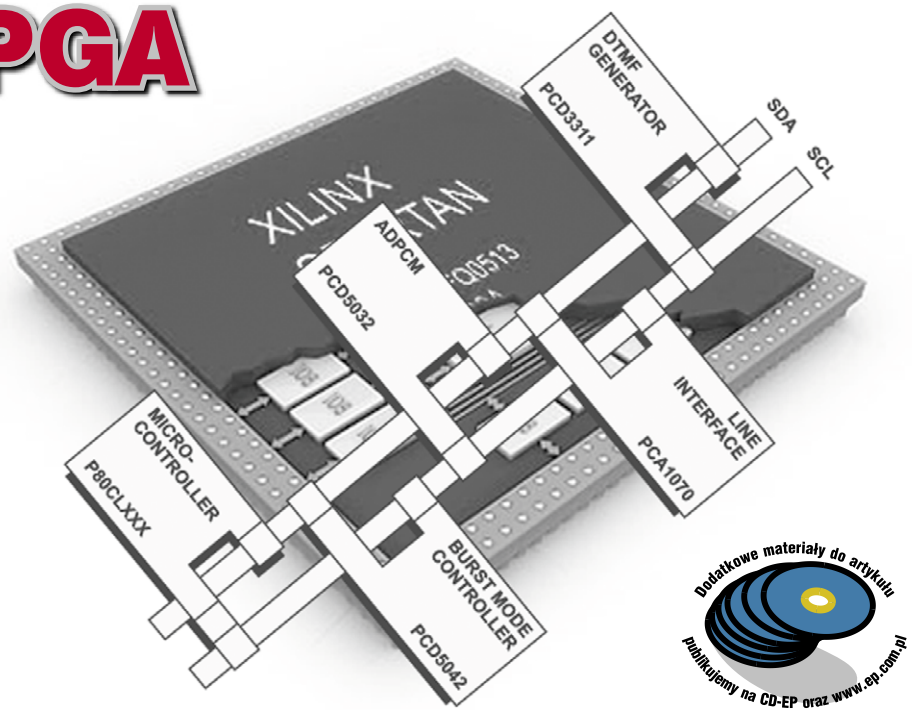
I²C w FPGA



Wiele układów peryferyjnych stosowanych w urządzeniach cyfrowych wykorzystuje do komunikacji z otoczeniem protokół I²C. Także układy FPGA, użyte jako peryferyjne dla jednostek centralnych, mogą porozumiewać się z nimi z wykorzystaniem tego protokołu. W artykule przedstawiamy przykładową implementację interfejsu I²C slave w układzie FPGA. Prezentowany „IP core" autor przygotował z wykorzystaniem języka VHDL.

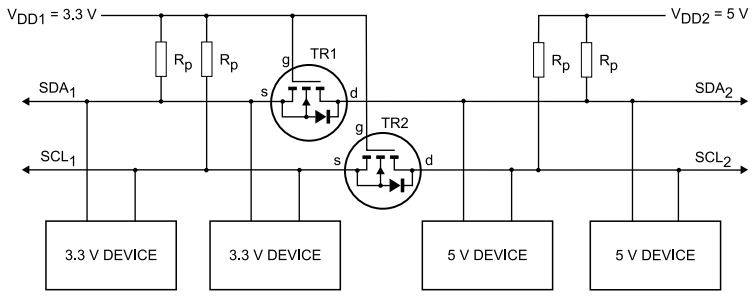
Protokół I²C opracowali inżynierowie firmy Philips, jego specyfikacja jest dostępna pod adresem: *http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf* oraz na płycie CD–EP9/2007B. Warto się z nim zapoznać, ponieważ w artykule skupimy się na pokazaniu wyłącznie implementacji interfejsu.

Jak pamiętamy, do przesyłania danych w I²C stosowane są dwie linie SCL i SDA. Łączą one dwa lub więcej urządzeń współpracujących w konfiguracji *master–slave*. Linia SCL przenosi sygnał zegarowy, generowany przez urządzenie *master*. Gdy *slave* chce na chwilę wstrzymać transmisje danych, może wymusić na niej stan niski. Podstawową częstotliwością taktowania linii SCL jest 100 kHz (prędkość standardowa, *standard mode*), stosowane są także częstotliwości do 400 kHz (*fast mode*), a nawet do 3,4 MHz (*high speed mode*).

Linia SDA służy do dwukierunkowego przesyłania danych. Rozpoczęcie przesyłania danych zawsze inicjuje *master*. *Slave* nie może zainicjować przesyłania danych. Rozpoczęcie transmisji danych (*start condition*) następuje, gdy przy aktywnej stanem wysokim linii SCL, następuje zmiana stanu linii SDA z wysokiego do niskiego. W następnej kolejności *master* wysyła 7 bitów adresu wskazującego urządzenie, z którym ma nastąpić nawiązanie komunikacji. Następnie jest wysyłany bit kierunku transferu danych. Logiczne 0 oznacza zapis, zaś logiczne 1 odczyt danych. Na końcu urządzenie odbiorcze (w tym przypadku *slave*) musi potwierdzić odebranie danych – ACK. Potwierdzenie polega na wystawieniu stanu niskiego na linii SDA, przy dziewiątym impulsie sygnału SCL.

Po prawidłowym wysłaniu bitów adresu oraz ustaleniu kierunku transmisji jest wykonywana transmisja danych. Wysyłanych jest osiem bitów danych i na dziewiątym impulsie SCL wystawiane jest ACK. W przypadku zapisu danych z *mastera* do *slave'a* sygnał ACK wystawia *slave*, zaś w przypadku odczytu danych wystawia go *master*.

Na końcu transmisji *master* wysyła sekwencję zakończenia transmisji (*stop condition*). Polega ona na tym, że przy wysokim stanie linii SCL następuje zmiana stanu na linii SDA z niskiego do wysokiego.



Rys. 1. Jedno z wielu możliwych rozwiązań konwerterów napięciowych 5<–>3,3 V

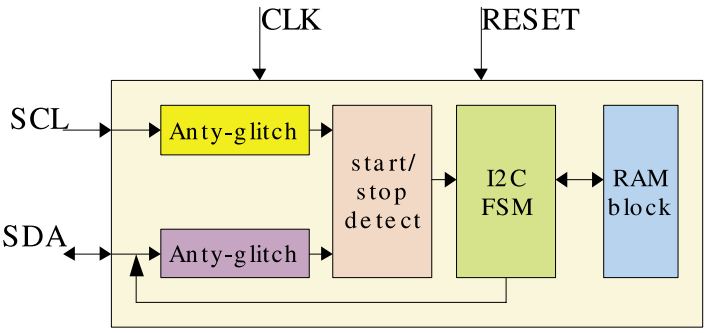
Połączenie elektryczne

Ponieważ część układów pracuje przy napięciach zasilania 5 V na obu liniach SCL i SDA, zaś układ FPGA z płytki ewaluacyjnej jest dostosowany tylko do napięć 3,3 V niezbędne jest wykonanie odpowiedniego przejścia. Wystarczające rozwiązanie jest zaproponowane na **rys. 1** (zgodnie ze specyfikacją I²C firmy NXP).

Implementacja

Projektowany interfejs ma następujące wejścia/wyjścia (**rys. 2**):

– SCL – sygnał synchronizujący I²C – wejście,
– SDA – sygnał danych I²C – wejście/wyjście,



Rys. 2. Schemat blokowy opisywanego interfejsu I²C dla FPGA

– CLK – sygnał zegara, synchronizujący pracę całego układu – wejście,
– RESET – sygnał resetujący układ – wejście.

Opis budowy interfejsu projektowanego modułu w języku VHDL wygląda następująco:

```
entity i2c_slave is
  port (
        RESET : in std_logic;
        CLK1  : in std_logic;
        SCL   : in std_logic;
        SDA   : inout std_logic);
end i2c_slave;
```

Wewnątrz modułu możemy wydzielić następujące bloki:
– *anty–glitch* – blok służący do synchronizacji sygnałów I²C wejściowych z głównym sygnałem synchronizujący,
– *dekoder start/stop* – blok dekodujący sekwencje *start* i *stop*,
– *I²C slave FSM* – automat stanów nadzorujący pracę modułu,
– *blok RAM* – blok pamięci RAM, w której będzie można przechowywać dane.

Anty–glitch

Detekcja sygnałów wejściowych I²C może odbywać się na dwa sposoby:

1. Sygnały wejściowe SCL i SDA są podawane bezpośrednio na wejścia przerzutników w układzie FPGA. Takie rozwiązanie jest jednak bardzo zawodne. Zdarza się bowiem, że na tych liniach występują zakłócenia, które mogą źle wpływać na pracę układu.
2. Sygnały wejściowe są „próbkowane" zewnętrznym sygnałem zegarowym CLK. Powszechnie można spotkać się z opisem takiego rozwiązania jako *anty–glitch*. Takie rozwiązanie ma praktycznie jedną wadę: zewnętrzny sygnał CLK musi mieć częstotliwość kilka razy większą niż sygnały próbkowane.

W Internecie można spotkać się z różnymi sposobami implementacji układu *anty–glitch*. Autor niniejszego artykułu sprawdził i przetestował jedno z najprostszych rozwiązań. Sygnał próbkowany podawany jest jako wejście danych na przerzutnik, za którym w kaskadzie są trzy kolejne przerzutniki. Wszystkie przerzutniki są taktowane zewnętrznym sygnałem zegarowym CLK. Następnie jako wyjście wyprowadzany jest sygnał będący iloczynem logicznym wszystkich wyjść z przerzutników. Na **list. 1** znajduje się opis w języku VHDL dla obu sygnałów układu I²C *slave*.

Sygnał CLK powinien mieć częstotliwość przynajmniej dziesięciokrotnie większą, niż sygnał SCL (zależy to od sposobu implementacji). Standardowo sygnał SCL ma częstotliwość 100 kHz.

Detektor warunków *start* i *stop*

Detektor obu warunków jest opisany prostym procesem (**list. 2**). Wewnętrzne sygnały *start* i *stop* są generowane na jeden impuls sygnału CLK.

Oba sygnały *start* i *stop* są wykorzystywane w głównej części obsługującej układ I²C *slave*.

Detektor zbocza sygnału SCL

Główna maszyna stanów układu I²C *slave* musi posiadać informacje o tym kiedy nastąpiła zmiana stanu linii SCL. Tranzycja ze stanu niskiego do wysokiego potrzebna jest do stwierdzenia kiedy coś zostało uaktywnione (odpowiednie sygnały na tej tranzycji są przygotowywane), np:
– wejście SDA jest gotowe do odczytu,
– na wyjściu SDA należy wystawić sekwencję ACK lub not ACK,

List. 1. Opis przerzutnikowej wersji układu anty–glitch

```
chip_reset <= RESET;
SCL_antyglitch : process(CLK, chip_reset)
begin
  if (chip_reset = ,1') then
    SCL_Q1 <= ,0'; SCL_Q2 <= ,0'; SCL_Q3 <= ,0';
    SDA_Q1 <= ,0'; SDA_Q2 <= ,0'; SDA_Q3 <= ,0';
  elsif CLK'event and CLK='1' then
    SDA_Q1 <= SDA; SDA_Q2 <= SDA_Q1; SDA_Q3 <= SDA_Q2;
    SCL_Q1 <= SCL; SCL_Q2 <= SCL_Q1; SCL_Q3 <= SCL_Q2;
  end if;
end process;
SCL_sig <= SCL_Q1 and SCL_Q2 and SCL_Q3;
SDA_sig <= SDA_Q1 and SDA_Q2 and SDA_Q3;
```

List. 2. Opis detektora warunków start i stop

```
process_StartStopDetect : process(CLK, chip_reset)
begin
  if (chip_reset = ,1') then
    start <= ,0';
    stop <= ,0';
    SDA_sig_prev <= ,0';
  elsif CLK'event and CLK='1' then
    SDA_sig_prev <= SDA_sig;
    if (SDA_sig='0') and (SDA_sig_prev='1') then
      start <= SCL_sig;
    else
      start <= ,0';
    end if;
    if (SDA_sig='1') and (SDA_sig_prev='0') then
      stop <= SCL_sig;
    else
      stop <= ,0';
    end if;
  end if;
end process;
```

List. 3. Detektor zbocza na linii SCL

```
process_SCLDetect : process(CLK, chip_reset)
begin
  if (chip_reset = ,1') then
    SCL_high <= ,0'; SCL_low <= ,0'; SCL_sig_prev <= ,0';
  elsif CLK'event and CLK='1' then
    SCL_sig_prev <= SCL_sig;
    if (SCL_sig='0') and (SCL_sig_prev='1') then SCL_low <= ,1';
    else SCL_low <= ,0'; end if;
    if (SCL_sig='1') and (SCL_sig_prev='0') then SCL_high <= ,1';
    else SCL_high <= ,0'; end if;
  end if;
end process;
```

List. 4. Automat stanów I2C slave

```
process_MainFSM : process(CLK, chip_reset, stop, start)
begin
  if (chip_reset = ,1') or (stop='1') or (start='1') then
    state <= „0000"; cnt <= 7;
  elsif CLK'event and CLK='1' then
    if (SCL_high = ,1') then
      if (state = „0000") then
        if cnt > 0 then
          cnt <= cnt - 1;
        else
          cnt <= 7;
          if (i2c_addr(7 downto 1) = device_addr(7 downto 1)) then
            state <= „1000";
          else
            state <= „0000";
          end if;
        end if;

      elsif (state = „1000") then
        if (i2c_addr = (device_addr(7 downto 1) & ,0')) then
          state <= „0111";
        elsif (i2c_addr = (device_addr(7 downto 1) & ,1')) then
          state <= „0011";
          cnt <= cnt - 1;
        else
          state <= „0000";
        end if;

      elsif (state = „0111") then
        if cnt > 0 then
          cnt <= cnt - 1;
        else
          cnt <= 7;
          state <= „1011";
        end if;

      elsif (state = „1011") then
        state <= „0010";

      elsif (state = „1110") then
        state <= „0010";

      elsif (state = „0010") then
        if cnt > 0 then
          cnt <= cnt - 1;
        else
          cnt <= 7;
          if (address = „11111111") then
            state <= „1111";
          else
            state <= „1110";
          end if;
        end if;

      elsif (state = „0011") then
        if (cnt = 7) then
          state <= „1100";
        elsif (cnt = 0) then
          cnt <= 7;
        else
          cnt <= cnt - 1;
        end if;

      elsif (state = „1100") then
        if SDA = ,0' then
          cnt <= cnt - 1;
          state <= „0011";
        elsif SDA = ,1' then
          state <= „1111";
        end if;

      end if;
    end if;
  end if;
end process;
```

– na wyjściu SDA należy wystawić dane odczytywane przez I²C *master*.

Tranzycja ze stanu wysokiego do niskiego oznacza wystawienie odpowiednich danych na wyjściu SDA, tak aby I²C *master* przy następnym zboczu narastającym sygnału SCL mógł odczytać linię SDA. Odpowiedni proces dekodujący tranzycję na linii SCL jest przedstawiono na **list. 3**.

Automat stanów I²C *slave*

Poniżej przedstawiono opis VHDL głównej maszyny stanów układu I²C *slave* (**list. 4**). W **tab. 1** przedstawiono również opis poszczególnych stanów.

Zmiana stanu może nastąpić jedynie po wykryciu narastającego zbocza sygnału SCL. Główny stan układu jest zakodowany symbolem "0000". Wejście do niego następuje zawsze po wykryciu warunku *startu* (lub *stopu*). W tym stanie odczytywany jest adres I²C wystawiany przez układ *master*. Po odczytaniu 8 bitów adresu, następuje porównanie odczytanego adresu z adresem naszego układu I²C. Jeśli ten adres się zgadza następuje przejście do kolejnego stanu, określonego symbolem "1000". W tym stanie następuje ponowne sprawdzenie adresu, jeśli jest on poprawny *slave* wystawia ACK. Jeśli ostatni bit jest ustawiony na 0, następnym stanem będzie stan "0111", w którym zostanie odczytanych 8 bitów adresu komórki pamięci, od której mają nastąpić kolejne operacje zapisu lub odczytu. Po odczytaniu adresu I²C *slave* wystawia ACK – stan "1011". Jeśli, odczytany adres będzie nieobsługiwany, wtedy układ przechodzi do stanu oczekiwania na warunek *startu* lub *stopu* "1111". Następnie układ *master* może bezpośrednio dokonać zapisu danych lub po wystawieniu tzw. warunku powtórzonego *startu*, może odczytać dane.

Jeśli bit ósmy adresu I²C jest ustawiony na 1 nastąpi przejście do stanu "0011" – odczyt danych. Następnie układ *slave* oczekuje, aż *master* wystawi sygnał ACK – stan "1110".

Jeśli z jakiegoś powodu układ I²C *master* nie wystawi sygnału ACK, po odczytaniu danych, wtedy układ przechodzi do stanu oczekiwania na warunek *startu* lub *stopu* "1111".

Ustawianie sygnałów kontrolnych

W **tab. 2** opisano sygnały ustawiane w procesie *process_SetFSM*, a na **list. 5** przedstawiono opis procesu ustawiającego sygnały kontrolne.

Dostęp do pamięci RAM

Układ I²C posiadający pamięć 256 bajtów może być bezpośrednio obsłużony przez zaprezentowaną implementację. Jeżeli jednak wymagane było by użycie większej pamięci wtedy należałoby rozszerzyć funkcjonalność prezentowanego układu, np. o specjalny rejestr z wyższymi bitami adresu. Na **list. 6** przedstawiono proces *process_SetRamFSM*, który ustawia trzy niezbędne sygnały umożliwiające obsługę dowolnej synchronicznej pamięci, z osobnymi sygnałami wejścia i wyjścia danych:

Tab. 1. Opis stanów automatu sterującego

Lp	Kod stanu	Opis	Kod stanu następnego
1	0000	Stan podstawowy, uruchamiany po wystąpieniu warunków start lub stop	1000
2	1000	Ustawia ACK, gdy odczytano poprawny adres I²C	0000 lub 0111 lub 0011
3	0111	Odczytuje adres dostępu	1011
4	0011	Odczytuje dane	1100
5	1011	Przygotowanie ACK, po odczycie adresu	0010
6	1110	Oczekuje ACK po wystawieniu danych	0010
7	0010	Operacja zapisu danych	1111 lub 1110
8	1111	Wystąpił błąd podczas operacji I²C, np. master nie wystawił ACK; wyjście z tego stanu nastąpi jedynie po warunku start lub stop	
9	1100	Oczekuje ACK po odczycie danych	1111 lub 0011

– *RAM_WE* – sygnał zezwalający na zapis danych do pamięci,
– *RAM_EN* – sygnał zezwalający na dostęp do pamięci (potrzebny przy odczycie i zapisie danych),
– *RAM_clk_high* – sygnał synchronizujący.

Synchronizacja sygnałów kontrolnych

Aby układ I²C *slave* poprawnie działał należy sygnały kontrolne ustawiać w odpowiednim momencie. Ponieważ I²C *master*, tak jak

List. 5. Proces ustawiający sygnały kontrolne

```
  process_SetFSM : process(CLK, chip_reset, stop, start)
  begin
    if (chip_reset = ,1') and (stop='0') and (start='0')
then
      address <= (others => ,0');
      i2c_addr <= (others => ,0');
      write_buf <= (others => ,0');
    elsif (chip_reset = ,1') or (stop='1') or (start='1')
then
      ACK <= ,0';  noACK <= ,0';
      SDA_out_active <= ,0';  SDA_out <= ,0';
    elsif CLK'event and CLK='1' then
      if (SCL_high = ,1') then
        if (state = „0000") then
          if cnt > 0 then
            i2c_addr(cnt) <= SDA;
          else
            i2c_addr(0) <= SDA;
            if (i2c_addr(7 downto 1) = device_addr(7 dow-
nto 1)) then
              ACK <= ,1';
            end if;
          end if;
        elsif (state = „1000") then
          ACK <= ,0';
          if (i2c_addr = (device_addr(7 downto 1) & ,1'))
then
            SDA_out <= RAM_DO(cnt);
            SDA_out_active <= ,1';
          end if;
        elsif (state = „0111") then
          if cnt > 0 then
            address(cnt) <= SDA;
          else
            address(0) <= SDA;
            ACK <= ,1';
          end if;
        elsif (state = „1011") then
          ACK <= ,0';
        elsif (state = „1110") then
          ACK <= ,0';
          address <= address + „00000001";
        elsif (state = „0010") then
          if cnt > 0 then
            write_buf(cnt) <= SDA;
          else
            write_buf(cnt) <= SDA;
            if (address = „11111111") then
              noACK <= ,1';
            else
              ACK <= ,1';
            end if;
          end if;
        elsif (state = „0011") then
          if (cnt = 7) then
            address <= address + „00000001";
            SDA_out_active <= ,0';
          elsif (cnt = 0) then
            SDA_out <= RAM_DO(cnt);
          else
            SDA_out <= RAM_DO(cnt);
          end if;
        elsif (state = „1100") then
          if SDA = ,0' then
            SDA_out <= RAM_DO(cnt);
            SDA_out_active <= ,1';
          end if;
        elsif (state = „1111") then
          noACK <= ,0';
        end if;
      end if;
    end if;
  end process;
```

List. 6. Proces ustawiający trzy niezbędne sygnały umożliwiające obsługę dowolnej synchronicznej pamięci z osobnymi sygnałami wejścia i wyjścia danych

```
  process_SetRamFSM : process(CLK, chip_reset, stop,
start)
  begin
    if (chip_reset = ,1') or (stop='1') or (start='1')
then
      RAM_WE <= ,0';
      RAM_clk_high <= ,0';
      RAM_EN <= ,0';
    elsif CLK'event and CLK='1' then
      if (SCL_high = ,1') then
        if (state = „0000") then
          RAM_EN <= ,0';
          RAM_WE <= ,0';
          RAM_clk_high <= ,0';
          if cnt = 0 then
            if ( (i2c_addr(7 downto 1) & SDA) = (devi-
ce_addr(7 downto

 1) & ,1')) then
              RAM_clk_high <= ,1';
              RAM_EN <= ,1';
            end if;
          end if;
        elsif (state = „1000") then
          RAM_clk_high <= ,0';
        elsif (state = „1011") then
          RAM_EN <= ,1';
          RAM_WE <= ,1';
          RAM_clk_high <= ,0';
        elsif (state = „1110") then
          RAM_EN <= ,1';
          RAM_WE <= ,1';
          RAM_clk_high <= ,0';
        elsif (state = „0010") then
          if cnt = 0 then
            RAM_clk_high <= ,1';
          end if;
        elsif (state = „0011") then
          if (cnt = 7) then
            RAM_clk_high <= ,1';
          end if;
        elsif (state = „1100") then
          RAM_clk_high <= ,0';
        end if;
      end if;
    end if;
  end process;
```

List. 7. Proces ustawiający aktywne sygnały na opadającym zboczu sygnału SCL

```
  process_ACK_proc : process(CLK, reset_fsm)
  begin
    if (reset_fsm = ,1') then
      ACK_low <= ,0';
      noACK_low <= ,0';
      SDA_out_active_low <= ,0';
      SDA_out_low <= ,0';
      RAM_CLK <= ,0';
    elsif CLK'event and CLK='1' then
      if (SCL_low = ,1') then
        ACK_low <= ACK;
        noACK_low <= noACK;
        SDA_out_active_low <= SDA_out_active;
        SDA_out_low <= SDA_out;
        RAM_CLK <= RAM_clk_high;
      end if;
    end if;
  end process;
```

I²C *slave*, próbkuje linię SDA na narastającym zboczu linii SCL, należy zapewnić, aby dane wystawiane na liniach SDA były poprawne przed wystąpieniem narastającego zbocza sygnału SCL. Najlepiej uczynić to na opadającym zboczu sygnału SCL. Proces opisany na **list. 7** ustawia aktywne sygnały na opadającym zboczu sygnału SCL.

W tym momencie można już dokonać odpowiednich przypisań do sygnału wyjściowego SDA, za co odpowiada poniższy opis:

```
  SDA <= ,0' when (ACK_low = ,1')
else
        ,1' when (noACK_low = ,1')
else
        SDA_out_low when (SDA_out_
active_low = ,1') else ,Z';
```

Następnie należy odpowiednio obsłużyć synchroniczną pamięć RAM, np. pokazaną na **list. 8**.

Warto zauważyć, że przykład został wykonany dla pamięci jednoportowej. Jeśli wymagane byłoby użycie pamięci przez inną część układu FPGA, należałoby wykorzystać pamięć dwuportową.

Marcin Nowakowski

Tab. 2. Opis sygnałów ustawianych w procesie process_SetFSM

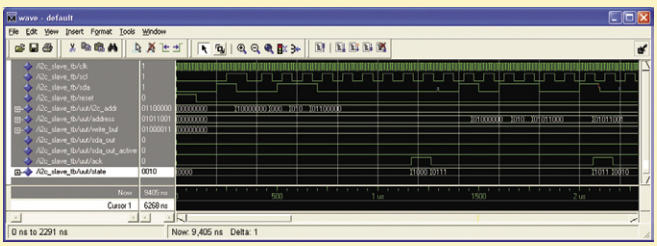
Nazwa sygnału	Opis
i2c_addr	Adres I²C odczytany z magistrali
address	Adres (8 bitów), z którego mają zostać odczytane dane, lub pod który dane mają zostać zapisane. Po każdej operacji odczytu lub zapisu danych, address jest automatycznie zwiększany o 1.
write_buf	Ośmiobitowy bufor na dane do zapisania
ACK	Sygnał określający, czy I²C slave ma wystawić potwierdzenie
noACK	Sygnał określający, czy I²C slave ma wystawić brak potwierdzenia
SDA_out_active	Sygnał określający, czy I²C slave ma przejąć kontrolę linii SDA
SDA_out	Sygnał określający jaka aktualnie wartość powinna zostać wystawiona na linii SDA przez I²C slave

List. 8. Opis synchronicznejpamięci RAM budowanej na bazie elementu bibliotecznego RAMB16_S9 (WebPack ISE)

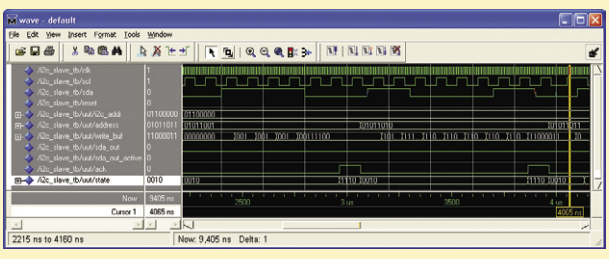
```
   -- RAMB16_S9: Virtex-II/II-Pro, Spartan-3/3E 2k x 8 + 1 Parity bit Single-
Port RAM
   -- Xilinx  HDL Language Template version 7.1i
   RAMB16_S9_inst : RAMB16_S9
   port map (
      DO => RAM_DO,       -- 8-bit Data Output
--       DOP => RAM_DOP,     -- 1-bit parity Output
      ADDR => RAM_ADDR,   -- 11-bit Address Input
      CLK => RAM_CLK,     -- Clock
      DI => RAM_DI,       -- 8-bit Data Input
      DIP => „0", --RAM_DIP,     -- 1-bit parity Input
      EN => RAM_EN,       -- RAM Enable Input
      SSR => RAM_SSR,     -- Synchronous Set/Reset Input
      WE => RAM_WE        -- Write Enable Input
   );

   -- End of RAMB16_S9_inst instantiation
   RAM_ADDR <= „000" & address;
   RAM_DI <= write_buf;
   RAM_SSR <= ,0';
```

Wyniki symulacji interfejsu I²C opisanego w artykule



Rys. 1.



Rys. 2.

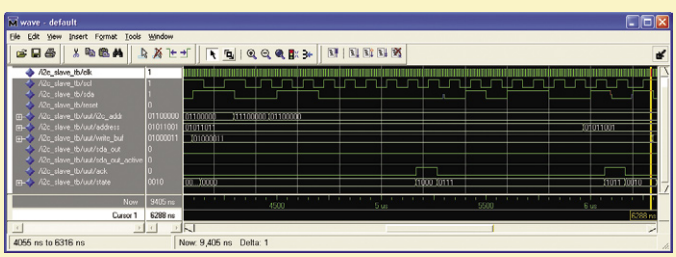
Na poniższych rysunkach przedstawiono wyniki symulacji układu I²C *slave*. Symulację przeprowadzono z wykorzystaniem pakietu ModelSim XE III w wersji 6.0a. Na wszystkich wykresach przebiegów przedstawiono zewnętrzne sygnały modułu I²C *slave* i kilka sygnałów wewnętrznych, w tym m.in. sygnały *state* i *address*.

Na **rys. 1** widać jak na magistrali I²C jest wystawiany adres 0x60 i następnie zapis adresu, komórki pamięci o adresie 0x59.

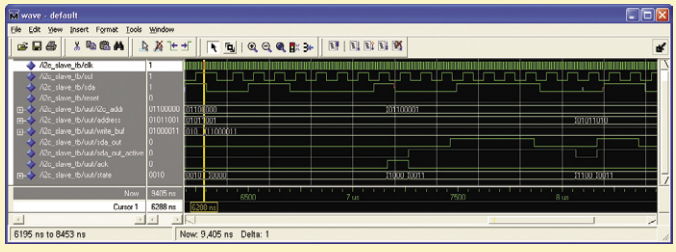
Na **rys. 2** przedstawiono przebieg zapisu dwóch wartości. Najpierw do pamięci pod wcześniej zaprogramowany adres 0x59, wpisywana jest wartość 0x3c. Następnie, co widać na przebiegu wewnętrzna wartość sygnału *address* jest zwiększana o jeden. I na końcu zapisywana jest kolejna wartość 0xc3 do pamięci, pod adresem 0x5a.

Na **rys. 3** przedstawiono początkowy przebieg procedury odczytu danych. Najpierw programowany jest adres I²C 0x60. Następnie programowany jest adres, spod którego nastąpi odczyt 0x59.

Na **rys. 4** przedstawiono procedurę odczytu. Najpierw programowany jest adres I²C 0x60, z ustawionym odpowiednio bitem odczytu. I później odczytywana jest komórka pamięci, spod poprzednio zaprogramowanego adresu 0x59. Można zaobserwować na przebiegach, że odczytano poprawnie wartość 0x3c i dodatkowo *address* został zwiększony o jeden, żeby umożliwić odczyt następnej komórki pamięci.



Rys. 3.



Rys. 4.