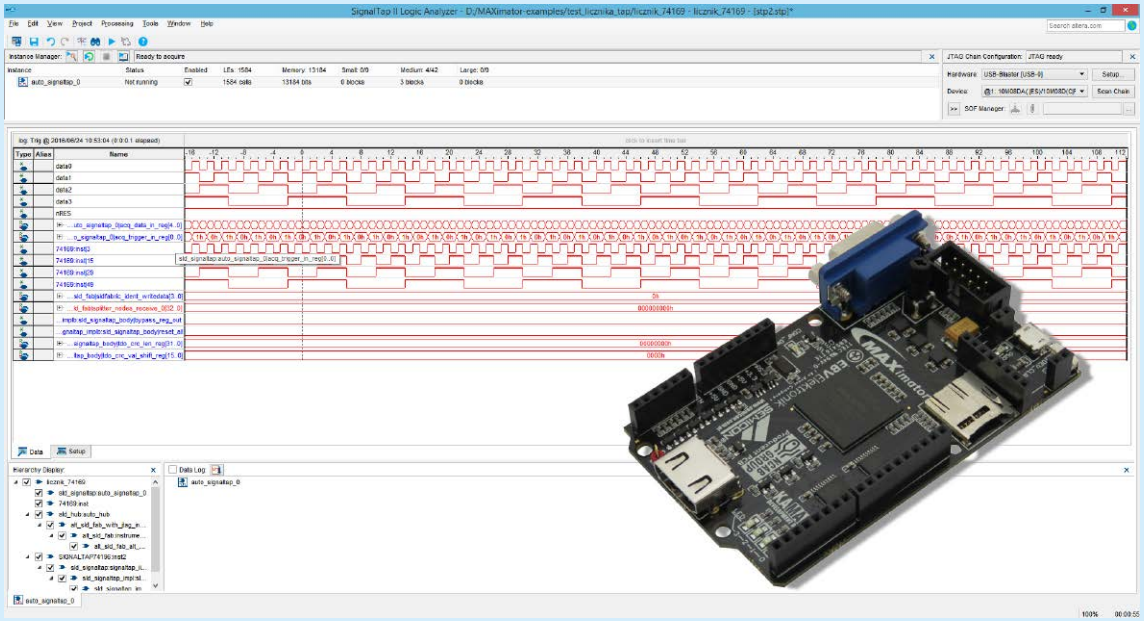


Pierwsze kroki z FPGA (4)

Szkoła MAXimatora – monitorowanie pracy projektu z użyciem debugera SignalTAP II

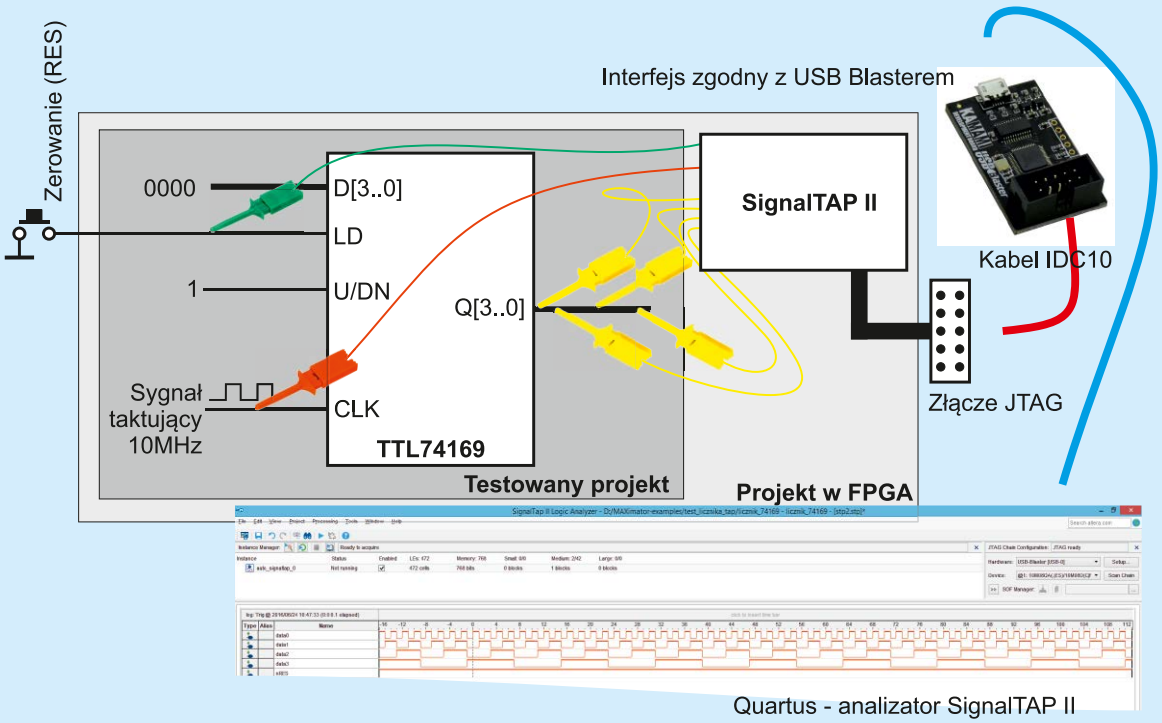
Miesiąc temu pokazaliśmy sposób weryfikacji działania projektu implementowanego w FPGA za pomocą symulatora wbudowanego w środowisko projektowe Quartus. Konstruktorzy korzystający z FPGA mają także inną możliwość weryfikacji działania implementowanego projektu, koncepcyjnie bliższą współczesnym mikrokontrolerom: dzięki bezpłatnemu IP core o nazwie SignalTAP II, który jest dystrybuowany wraz z Quartusem, można wygodnie zweryfikować działanie projektu, korzystając z interfejsu JTAG. W artykule pokażemy jak to zrobić krok-po-kroku.

Idea działania IP core o nazwie SignalTAP II przypomina wkładany do wnętrza FPGA rejestrator stanów logicznych. Wyobraźmy sobie wielowejściową sondę, której wejścia są dołączone do interesujących nas sygnałów w implementowanym projekcie, wyposażoną we własną pamięć (**rysunek 1**). W pamięci tej są rejestrowane stany logiczne występujące w wybranych (poprzez dołączenie linii wejściowych modułu SignalTAP) przez konstruktora miejscach. Odczyt zgromadzonych w pamięci danych odbywa się za pomocą interfejsu *USB Blaster* za pośrednictwem interfejsu JTAG, tego samego, który jest używany do programowania i/lub konfigurowania FPGA.

Podczas realizacji projektów z wykorzystaniem SignalTAP II trzeba pamiętać, że jest to IP core implementowany w FPGA, w związku z czym pochłania część zasobów tego układu. Na **rysunku 2** pokazano porównanie wykorzystania zasobów przy implementacji samego licznika 74169 (lewa strona rysunku 2) i przy implementacji tego samego licznika z dołączonym debugerem SignalTAP. Porównanie liczb może wywołać myśl, że SignalTAP II jest bardzo zasobożerny (odnosząc do implementację do samego 74169), ale przy dostępnych zasobach układu 10M08 użytego w MAXimatorze widać, że nie „zjada" on relatywnie wielu zasobów. Trzeba wziąć pod uwagę, że pojemność użytej pamięci i liczba zajętych przez SignalTAP rejestrów będzie się zmieniać w zależności od liczby rejestrowanych kanałów i głębokości pamięci rejestrującej zmiany stanów logicznych. W prezentowanym przykładzie zadeklarowano 7 kanałów wejściowych (z czego użyto 6) i głębokość pamięci 128 słów.

Przykład zastosowania debugera SignalTAP pokażemy na przykładzie znanego nam już licznika 74196, który będzie taktowany sygnałem zegarowym 10 MHz (generator kwarcowy wbudowany na płytce MAXimator). Monitorować będziemy stany linii wyjściowych licznika, a także poziomy sygnałów na jego wejściach. Wejście LD

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **77642**, pass: **3220ppmm**

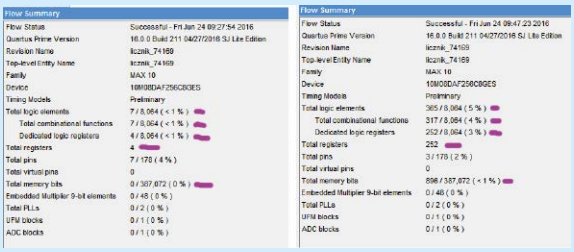


Rysunek 1. Schemat blokowy ilustrujący działanie sprzętowego debugera SignalTAP II

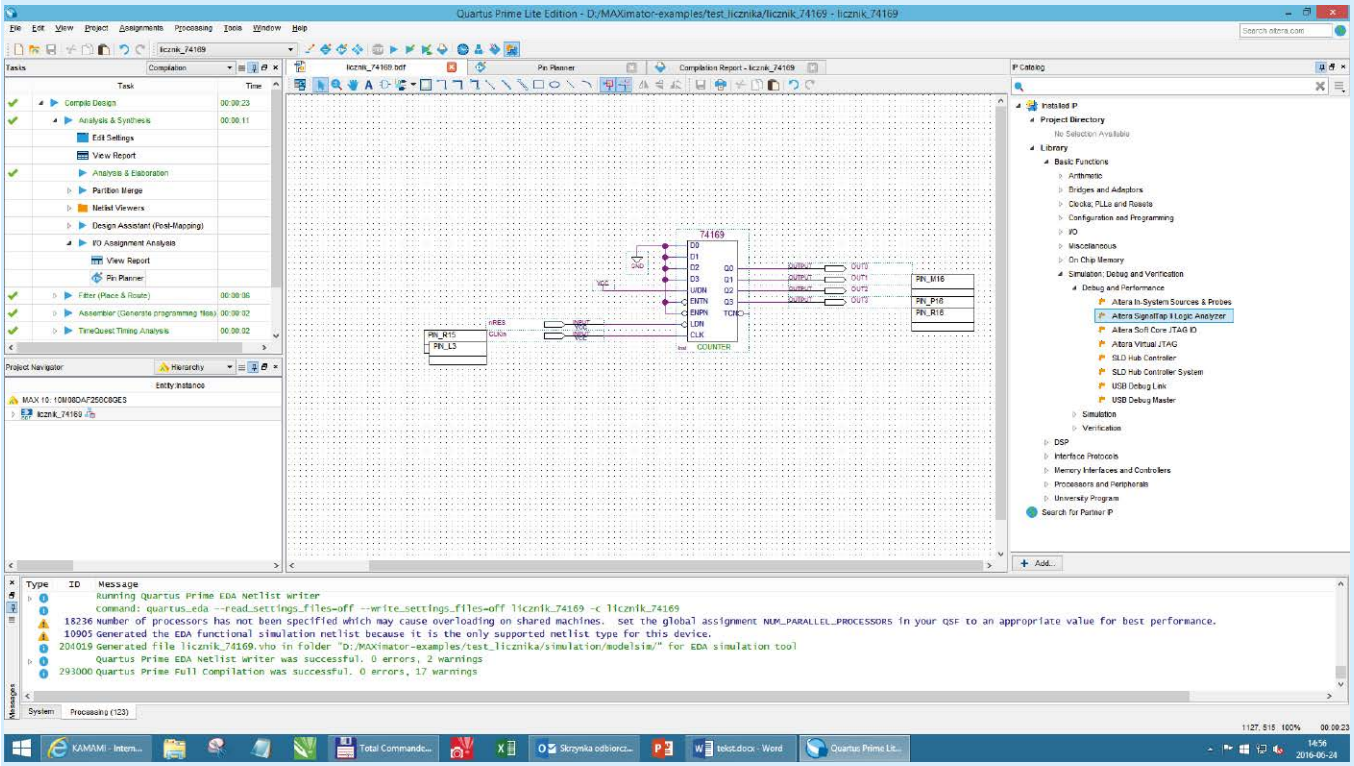
licznika spełnia w przykładzie rolę wejścia zerującego, posłuży nam ono w projekcie także do wyzwolenia rejestracji stanów linii przez SignalTAP II.

W wersjach Quartusa od 14.0 możliwe są dwie ścieżki implementacji debugera SignalTAP II, zaczniemy od prezentacji metody klasycznej, w której ręcznie dodajemy do schematu lub opisu HDL IP core debugera.

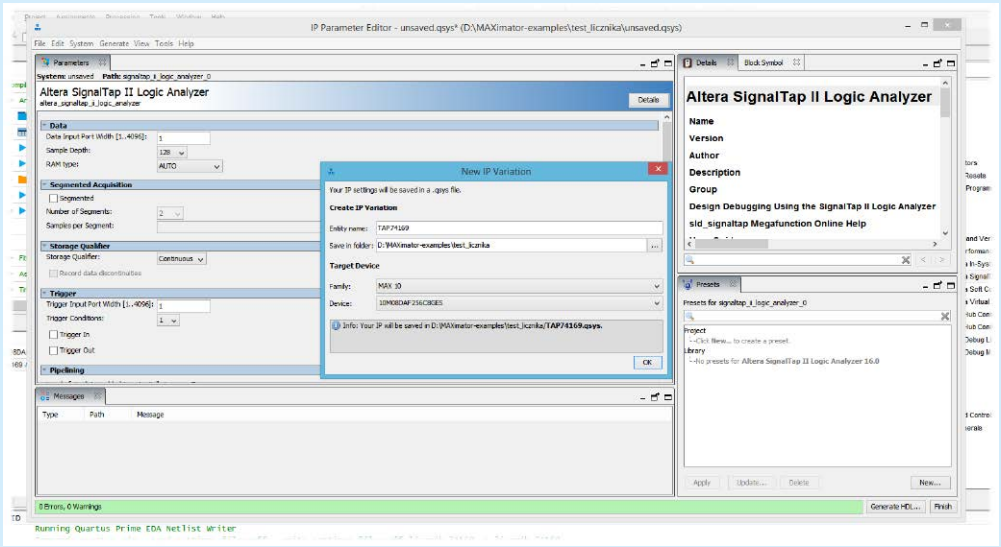
Implementację debugera SignalTAP II w projekcie implementowanym w FPGA należy przeprowadzić po jego kompilacji i przypisaniu linii sygnałowych do fizycznych wyprowadzeń układu. Zaczynamy od wygenerowania sparametryzowanego IP core



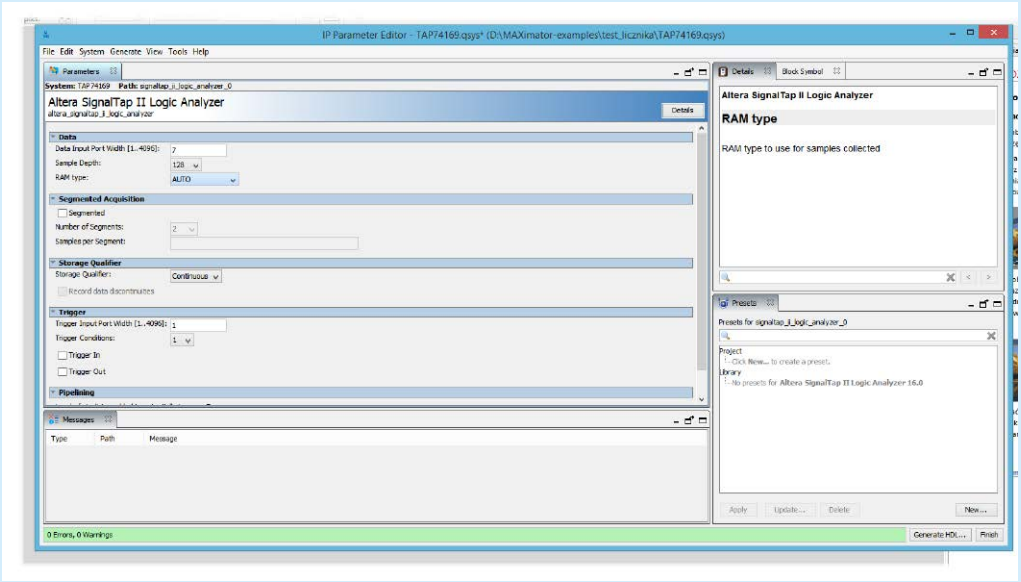
Rysunek 2. Porównanie zajętych w FPGA zasobów przy implementacji samego licznika 74169 (lewa strona) i przy implementacji tego samego licznika z dołączonym debugerem SignalTAP II (prawa strona)



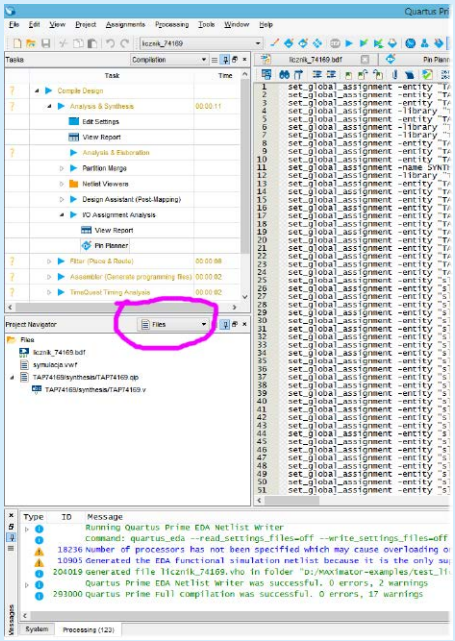
Rysunek 3. SignalTAP II jest dostępny w ramach bezpłatnych IP core w pakietach Quartus II i Quartus Prime



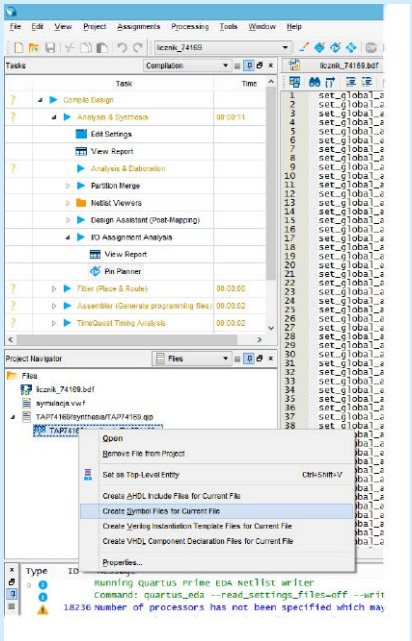
Rysunek 4. Generację IP core debugera zaczynamy od nadania nazwy naszej mutacji SignalTAP II



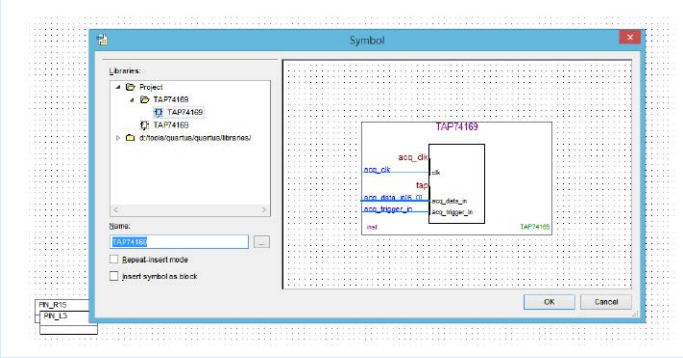
Rysunek 5. Okno parametryzacji debugera SignalTAP II



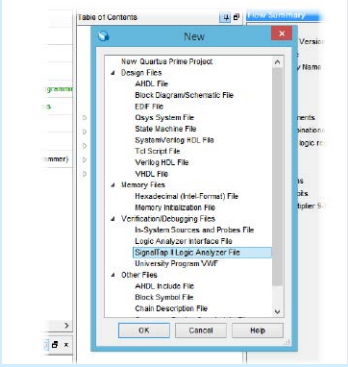
Rysunek 6. Ustawienie w menedżerze projektu pozwalające wyświetlić listę wszystkich plików w projekcie



Rysunek 7. W ten sposób generujemy symbol graficzny dla wybranego pliku źródłowego

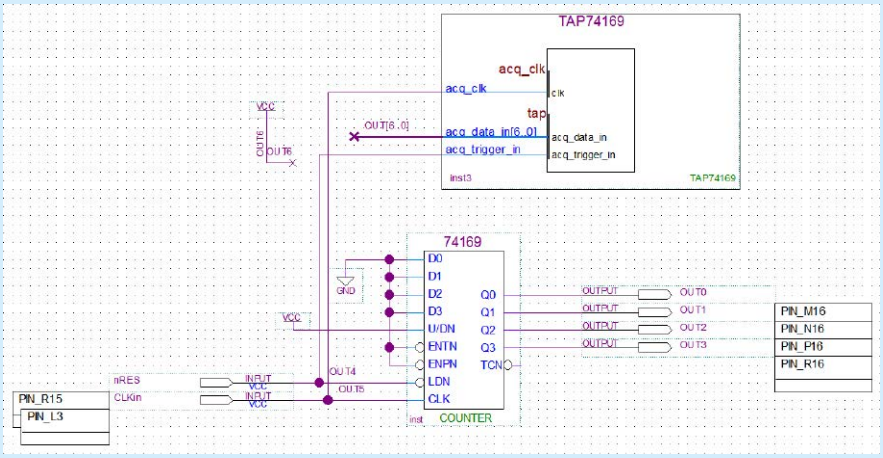


Rysunek 8. Wygenerowany symbol graficzny – użyjemy go na schemacie

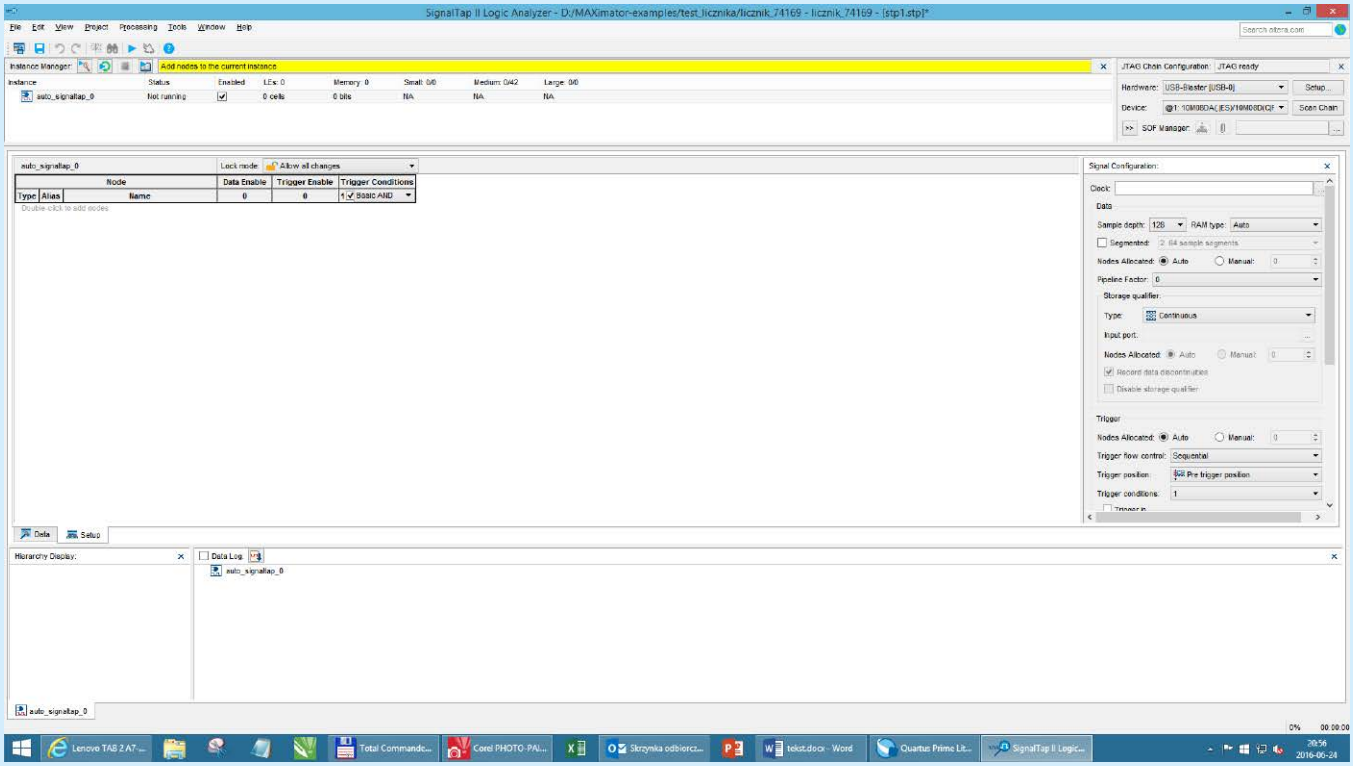


Rysunek 10. Menu, z którego wybieramy plik zawierający opis konfiguracji debugera

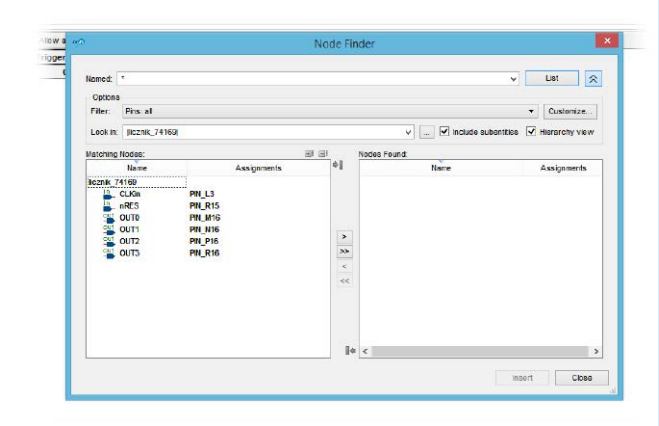
debuggera SignalTAP II, co wymaga wybrania w katalogu dostępnych IP core opcji *Library>Basic Functions>Simulation, Debug and Verification>Debug and Performance>Altera SignalTap II Logic Analyzer* (**rysunek 3**). Spowoduje to uruchomienie generatora i parametryzatora IP core'ów (**rysunek 4**), w którego pierwszym oknie podajemy nazwę generowanego modułu (w przykładzie jest to *TAP74169*). Parametryzacja debugera w naszym przykładzie ogranicza się do podania liczby potrzebnych wejść rejestrujących dane, w przykładzie ich liczbę nadmiarowo ustalono na 7 (**rysunek 5**). Do naszych celów wystarcza domyślna pojemność pamięci rejestrującej dane, która wynosi 128 próbek, ale można – w zależności od potrzeb – zwiększyć lub zmniejszyć jej pojemność (w sekcji *Data* okna pokazanego na rysunku 5). Nie będziemy teraz korzystać z pozostałych, bardziej zaawansowanych możliwości



Rysunek 9. Przykładowy schemat z zastosowanym symbolem debugera



Rysunek 11. Zakładka konfiguracyjna SignalTAP II



Rysunek 12. Lista sygnałów monitorowanych w FPGA, które będą wyświetlane w postaci przebiegów

parametryzacji, wrócimy do ich prezentacji przy okazji kolejnych odcinków kursu.

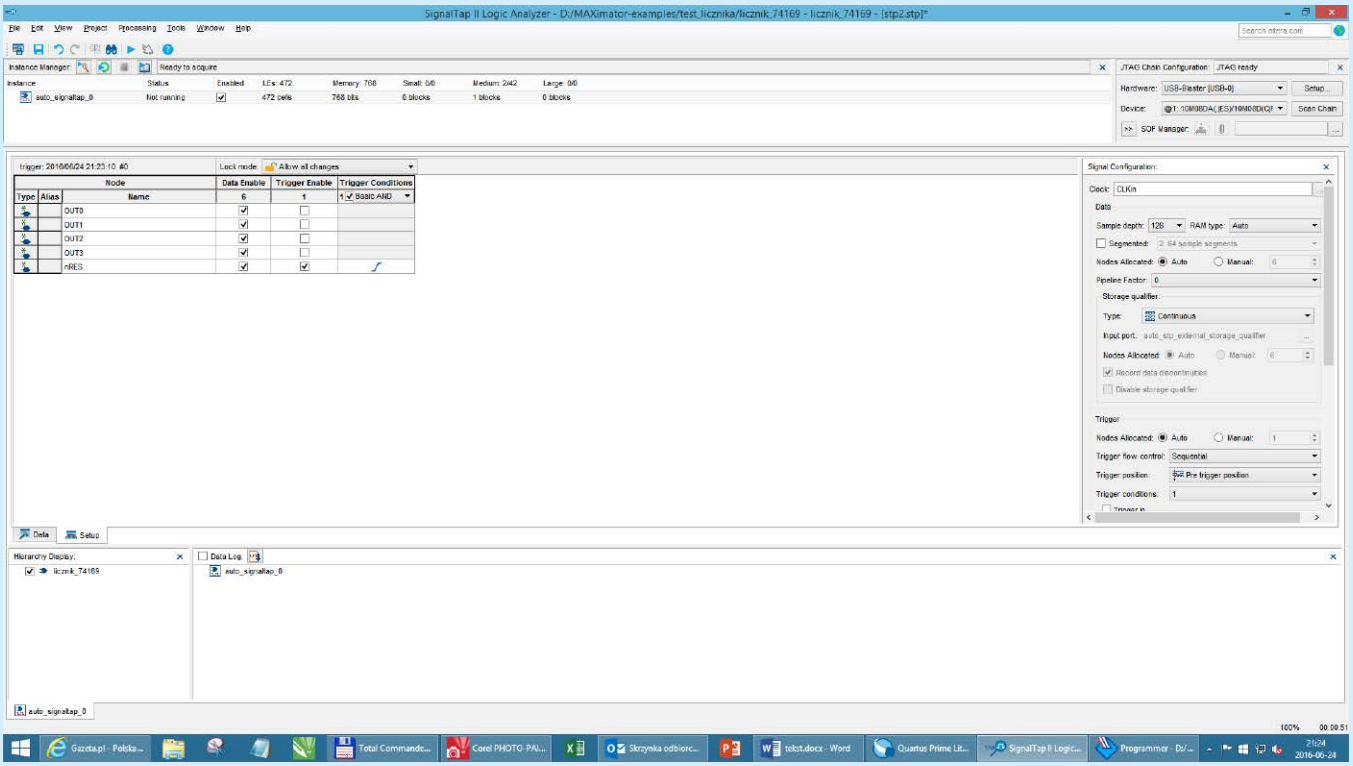
Po ustaleniu potrzebnej liczby wejść generujemy (wciskając *Generate HDL…*) pliki źródłowe debugera w wybranym języku HDL (Verilog lub VHDL). Późniejsze ewentualne modyfikacje parametrów modułu można wygodnie zmieniać otwierając plik *TAP74169.qsys* i po wprowadzeniu zmian ponownie generując pliki HDL. W tym momencie mamy wygenerowany opis HDL debugera, ale nie został on jeszcze dodany do projektu. Proces ten musimy wykonać ręcznie, co wymaga otwarcia pliku *TAP74169.qip* (znajduje on się katalogu *sciezka_generacji_plikow-debugera\synthesis*) i następnie dodania go do projektu (*Project>Add Current File to Project*). Po wykonaniu tych czynności w nawigatorze projektu (*Project Navigator* – **rysunek 6**) wyświetlamy wszystkie pliki wchodzące w skład projektu i rozwijamy gałąź *TAP74169/sythesis/* (rysunek 6), w której znajduje się plik HDL z opisem debugera (w przykładzie jest to plik w Verilogu). Klikamy w nazwę pliku prawym klawiszem myszki i wybieramy opcję *Create Symbol Files for Current File* (**rysunek 7**), co powoduje dodanie graficznego symbolu debugera do biblioteki projektu.

Żeby położyć symbol debugera na planszy schematu postępujemy tak jak w przypadku innych symboli – dwukrotnie klikamy w puste miejsce na planszy, co spowoduje wyświetlenie okna *Symbol* (**rysunek 8**), w którym dostępne są dwa symbole TAP74169. Jest to normalne zjawisko, wynikające z pewnych niekonsekwentnych rozwiązań w pakiecie Quartus, przy czym są one nieszkodliwe.

Wybieramy jeden z dostępnych symboli i opisujemy sposób dołączenia linii wejściowych oraz linii wyzwalającej do testowanej części projektu. Odbywa się to w taki sam sposób, jak w przypadku rysowania schematu rozwiązania implementowanego w ramach projektu.

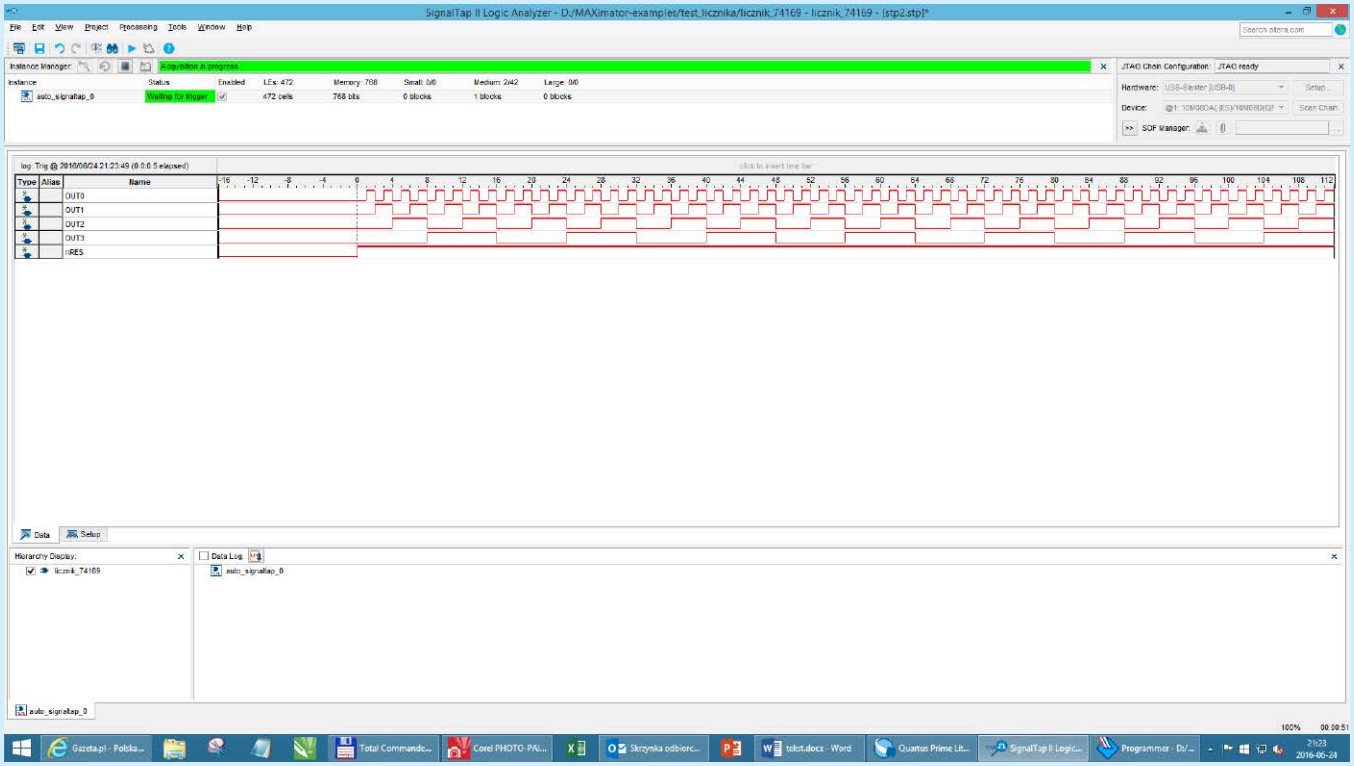
Na **rysunku 9** pokazano schemat połączeń układu testowego, w którym jedno z wejść debugera dołączono na stałe do logicznej „1”. Po wykonaniu wszystkich połączeń kompilujemy projekt, żeby wychwycić ewentualne błędy uniemożliwiające jego poprawną implementację. Jeżeli wszystko jest w porządku, możemy przejść do kolejnego kroku – przygotowania pliku debugera. W tym celu wybieramy w menu *File>New* i w wyświetlonym oknie, w sekcji *Verification/Debugging Files* (**rysunek 10**), wybieramy *SignalTap II Logic Analyzer File*. Okno analizatora pokazano na **rysunku 11**.

Konfigurację tej części projektu zaczynamy od ustalenia listy monitorowanych sygnałów, co wymaga dwukrotnego kliknięcia w zakładce *Setup*, w wyniku czego wyświetli się okno *Node Finder* pokazane na **rysunku 12**. Ponieważ wejścia debugera SignalTAP II dołączyliśmy do wyjść i niektórych wejść monitorowanego licznika, w filtrze sygnałów (*Options>Filter*) wybieramy opcję *Pins:all* i następnie klikamy *List*. Wynik tej operacji pokazano na rysunku 12. Wybieramy z listy interesujące nas sygnały (w przykładzie OUT3…OUT0 oraz nRES, przy



Rysunek 13. Zalecana do celów naszego kursu konfiguracja monitorowanych sygnałów

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **77642**, pass: **3220ppmm**



Rysunek 14. Wynik działania debugera – wyświetlone sygnały zostały pobrane we wnętrzu FPGA za pomocą SignalTAP II

którym aktywujemy opcję *Trigger Enable* – **rysunek 13**) i klikamy *Insert*.

Po wykonaniu tych czynności ponownie kompilujemy cały projekt, do czego zachęca komunikat widoczny w belce *Instance Manager*, w górnej części okna pokazanego na rysunku 13. Po rekompilacji programujemy lub konfigurujemy docelowy układ FPGA, co jest oczywiste – musimy umieścić testowany projekt w FPGA, żeby móc go fizycznie monitorować.

Alternatywnym, wygodniejszym sposobem implementacji w projekcie umieszczanym w FPGA debugera SignalTAP, jest dodanie do projektu pliku **.stp* (jak na rysunku 10) i wykonanie kolejnych czynności, łącznie z wybieraniem monitorowanych sygnałów z poziomu okna SignalTap II Logic Analyzer (jak na **rysunku 14**). W nowych wersjach Quartusa uruchomienie tego okna i zdefiniowanie listy sygnałów powoduje automatyczną implementację debugera w projekcie.

We wrześniowym wydaniu EP przedstawimy zaawansowane możliwości debugera SignalTAP II.

Piotr Zbysiński, EP

Z przyjemnością publikujemy listę 14 osób, które otrzymały nagrody w konkursie dla fanów FPGA, który ogłosiliśmy w EP5/2016. Łącznie otrzymaliśmy aż 54 zgłoszenia!

Nagrody zostały wysłane pocztą przez sklep KAMAMI.pl. We wrześniowym wydaniu EP kolejny konkurs!

Lista osób nagrodzonych:

1. Aleksandra Gaszyńska
2. Jacek Dębniak
3. Jakub Jakubowski
4. Jarosław Krukowski
5. Karol Kulig
6. Konrad Miciński
7. Wojciech Kozikowski
8. Marcin Skóra
9. Mariusz Dziębowski
10. Piotr Zamorski
11. Przemysław Szymański
12. Wojciech Stodulny
13. Piotr Chodorowski
14. Wojciech Przybycień

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 77642, pass: 3220ppmm