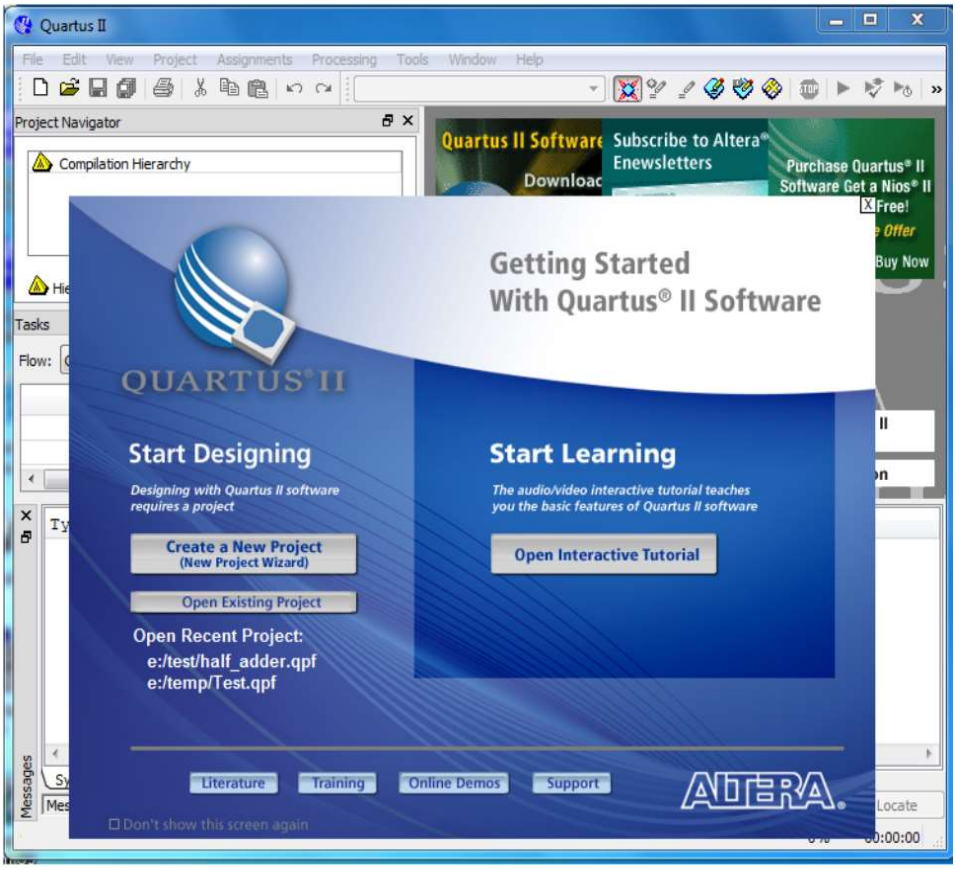
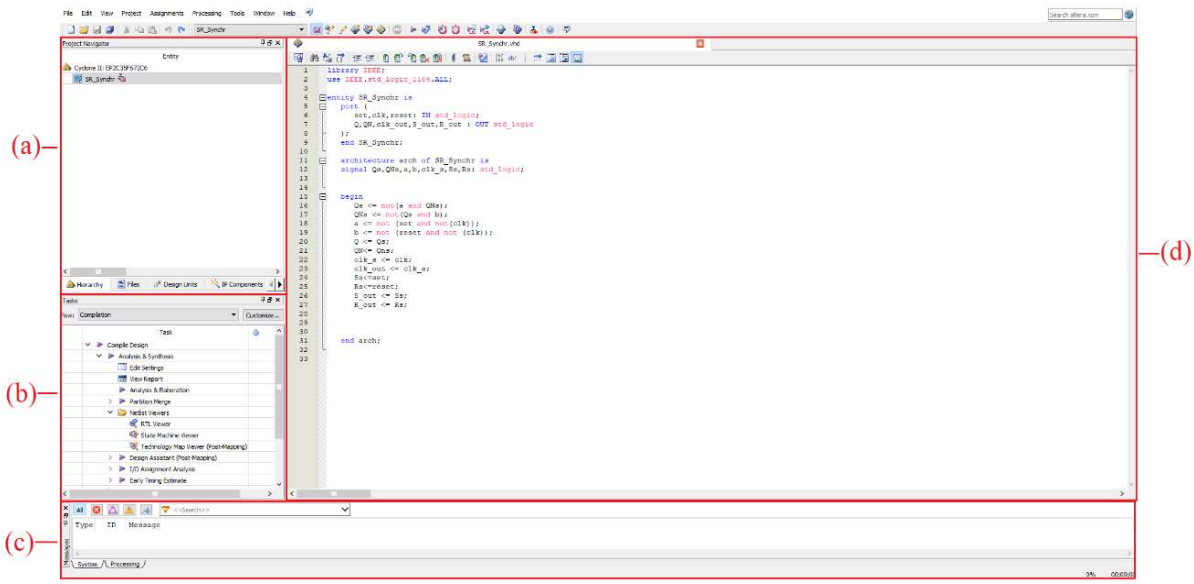
Oprogramowanie Quartus II

Jest to uniwersalne środowisko programistyczne które może być używane do projektowania, symulacji oraz syntezy układów cyfrowych bazujących na modułach FPGA



1. Interfejs graficzny programu

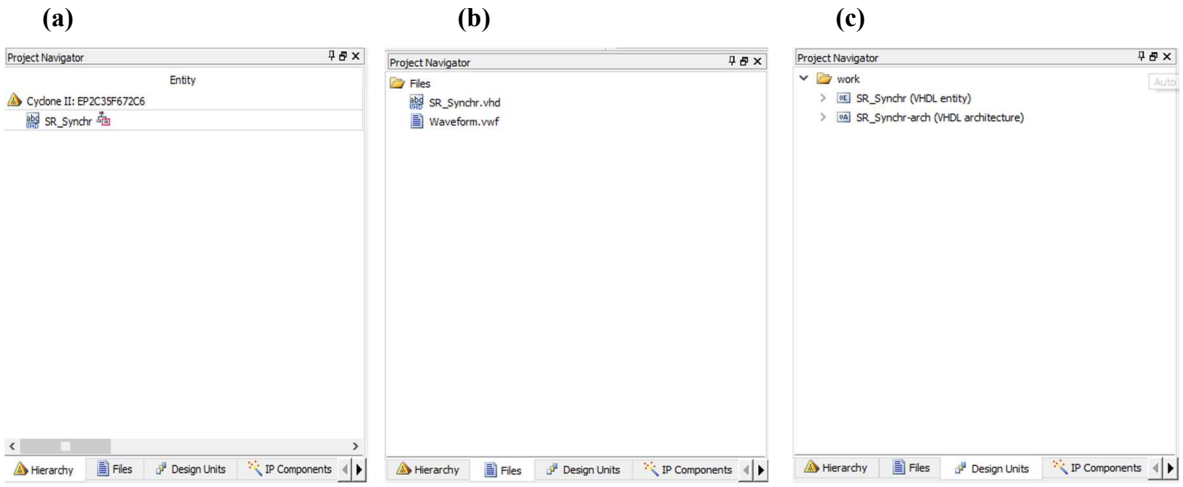
Przedstawione na **Rys. 1** okno interfejsu oprogramowania podzielić można na cztery części, które opisane zostały w dalszej części tego podrozdziału.



Rys. 1. Interfejs programu Quartus II z podziałem na okna

(a) Okno nawigacji projektu (**Project Navigator**) zawiera wszystkie najważniejsze pliki z nim związane. Okno to składa się z pięciu zakładek. Na Rys. 2, Rys. 3 i Rys. 4 przedstawiono trzy najważniejsze z nich. Zakładka hierarchii projektu (**Hierarchy**) (patrz **Rys.2 (a)**) umożliwia podgląd wszystkich tzw. „bytów" (z ang. „*entities*") znajdujących się w projekcie.

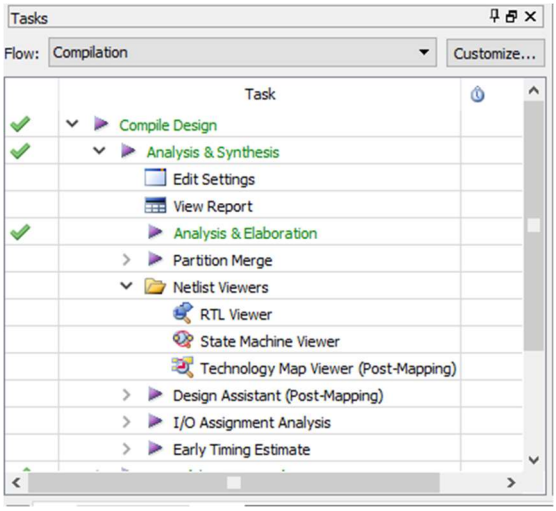
Zakładka ta jest szczególnie przydatna przy większych projektach składających się z wielu bytów, ponieważ przedstawia wtedy ona który byt jest bytem głównym spajającym cały projekt.



Rys. 2. Okna zakładek nawigacji projektu (Project Navigator)

Zakładka plików projektu (**Files**) (**Rys. 2(b)**) umożliwia podgląd plików znajdujących się aktualnie w projekcie. Zakładka jednostek projektowych (**Design Units**) (**Rys. 2(c)**) umożliwia podgląd wszystkich jednostek projektowych znajdujących się aktualnie w projekcie. Jednostki projektowe to zarówno byty, jak i architektury.

(b) Okno zadań (**Tasks**) (**Rys. 3**) zawiera wszystkie funkcje programu i wyświetla na bieżąco podczas aktualnego procesu (np. symulacji lub kompilacji) które jego części zostały wykonane poprawnie (zaznaczane są one zielonymi fajkami).



Rys. 3. Okno zadań realizowanych w procesie kompilacji projektu

(c) Okno wiadomości (**Messages**) (**Rys. 4**) zawiera wszystkie tekstowe informacje generowane przez program. Podczas kompilacji lub symulacji wyświetla informacje o błędach i ostrzeżeniach, a także inne szczegóły dotyczące danego procesu.

Rys. 4. Okno wiadomości systemowych

(d) Okno pliku (**Rys. 5**) wyświetla wszystkie pliki dołączone do projektu, jak na przykład główny plik z kodem opisującym działanie układu. W oknie tym mogą znajdować się również pliki symulacji itp...

SR_Synchr.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3
4  entity SR_Synchr is
5     port (
6        set,clk,reset: IN std_logic;
7        Q,QN,TEST : OUT std_logic
8     );
9     end SR_Synchr;
10
11    architecture arch of SR_Synchr is
12    signal Q1,QN1,a,b: std_logic;
13    begin
14       Q1 <= not(a and QN1);
15       QN1 <= not(Q1 and b);
16       a <= not (set and not(clk));
17       b <= not (reset and not (clk));
18       Q <= Q1;
19       QN<= not(Q1);
20    end arch;
21
22
```

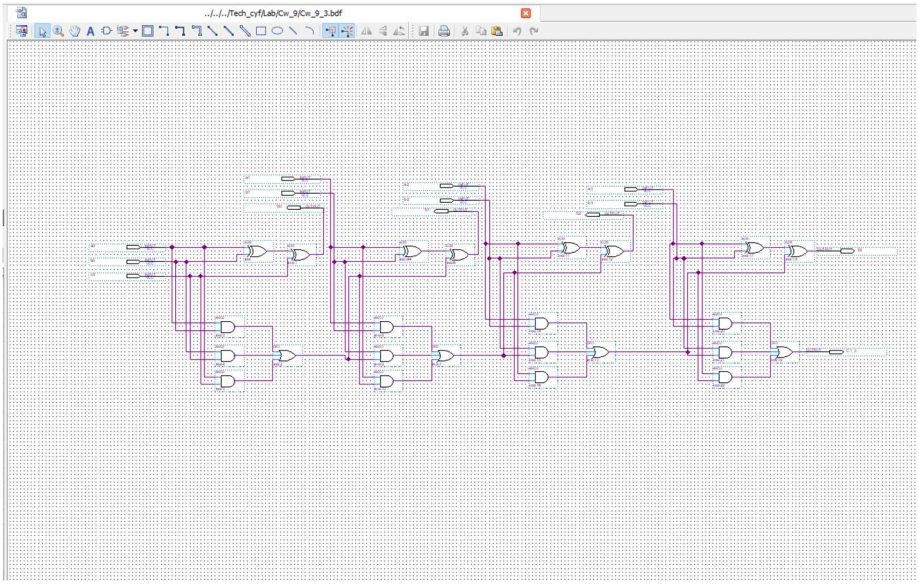
Rys. 5. Okno pliku z kodem źródłowym przykładowego projektu w języku VHDL

2. Sposoby projektowania układów

Istnieje kilka sposobów projektowania układów cyfrowych z użyciem oprogramowania Quartus II, najprostszym z nich jest projektowanie w środowisku graficznym poprzez stworzenie pliku schematu graficznego. Innym graficznym sposobem projektowania jest projektowanie maszyny stanów. Bardziej zaawansowane natomiast są języki opisu układów (VHDL, AHDL i Verilog).

- *Projektowanie graficzne*

Najbardziej prosta i intuicyjna metoda projektowania, wykorzystuje graficzne elementy które projektujący dodaje do obszaru roboczego i łączy je przewodami tworząc w ten sposób strukturę całego układu (**Rys. 6**).

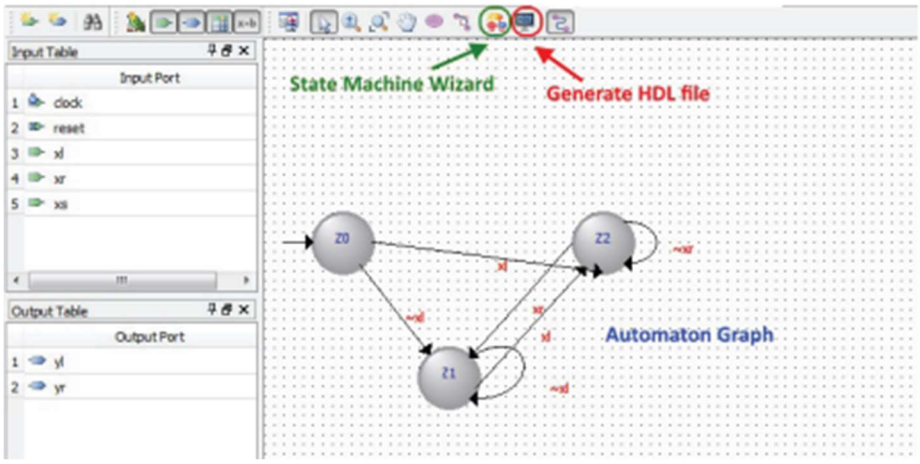


Rys. 6. Przykładowy schemat graficzny w oprogramowaniu Quartus II

Oprogramowanie Quartus zawiera dużą bibliotekę gotowych elementów, które można wykorzystać przy projektowaniu układów jak też tworzyć własne. Projektowanie w interfejsie graficznym jest jednak dosyć uciążliwe przy złożonych projektach i w tym przypadku preferowane są pliki wsadowe wykonane w jednym z dostępnych języków (VHDL, AHDL i Verilog). Pliki projektowe w tym formacie mają rozszerzenie **.bdf**.

- *Maszyna stanów*

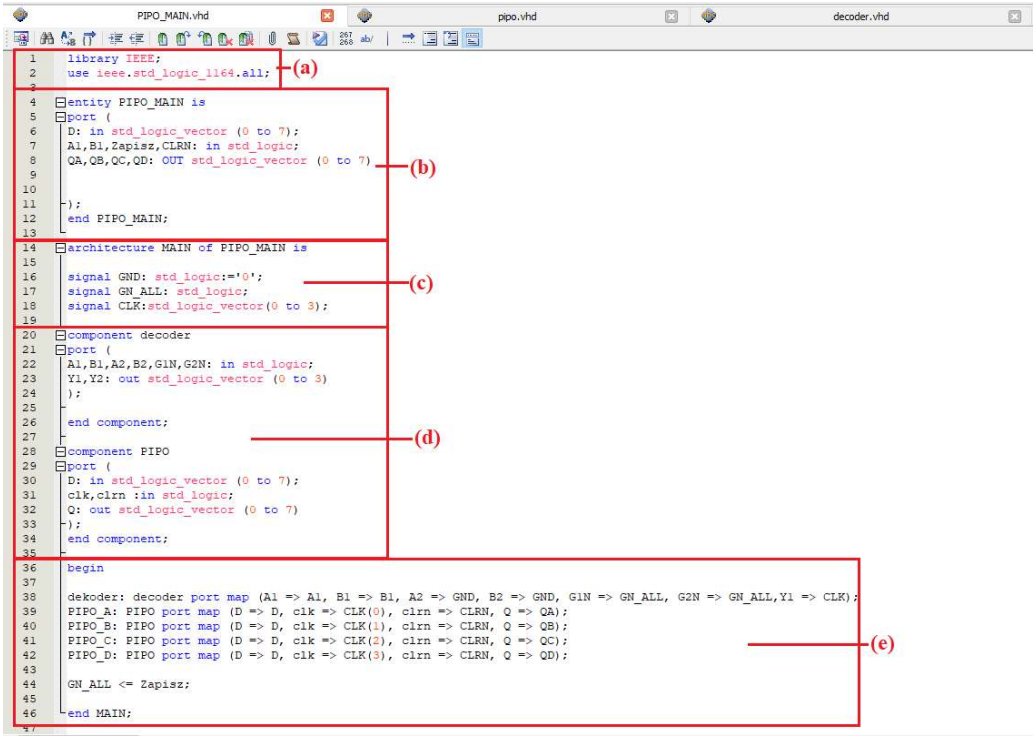
Maszyna stanów (**Rys. 7**) pozwala na projektowanie układu na zasadzie grafu, program następnie na podstawie opisu przejść stanów tworzy plik w języku opisu układu (na przykład VHDL). Pliki maszyny stanów posiadają rozszerzenie **.smf.**



Rys. 7. Przykładowy graf maszyny stanów w oprogramowaniu Quartus II

- *VHDL*

Język ten został wykorzystany w celach projektowych z niniejszej pracy. Jest to język opisu układu pozwalający na projektowanie układów w sposób przepływowy, behawioralny, lub strukturalny. Sposób strukturalny to bardziej zaawansowany sposób projektowania układów, którego zaletą jest łatwiejsze przedstawienie i stworzenie dużych i złożonych projektów. Programowanie strukturalne pozwala na wykorzystanie układu cyfrowego opisanego w oddzielnym pliku. Układ taki można wykorzystać dowolną liczbę razy w jednym projekcie opisując jedynie sposób podłączenia wejść i wyjść, oraz sygnałów do odpowiednich bloków. Jest to niewątpliwie zaleta tej metody, ponieważ możemy wielokrotnie odwołać się do układu który już zaprojektowaliśmy i nie musimy ponownie opisywać jego działania w kodzie programu. Wadą języka VHDL jest natomiast konieczność poznania jego składni i wielu zasad związanych z projektowaniem przy jego użyciu. Pliki w tym formacie mają rozszerzenie .**vhd.** Przykładowa realizacja projektu w sposób strukturalny została przedstawiona na **Rys. 8.**




```
1  library IEEE;
2  use ieee.std_logic_1164.all;
3
4  entity PIPO_MAIN is
5  port (
6   D: in std_logic_vector (0 to 7);
7   A1,B1,Zapisz,CLRN: in std_logic;
8   QA,QB,QC,QD: OUT std_logic_vector (0 to 7)
9
10
11 );
12 end PIPO_MAIN;
13
14 architecture MAIN of PIPO_MAIN is
15
16  signal GND: std_logic:='0';
17  signal GN_ALL: std_logic;
18  signal CLK:std_logic_vector(0 to 3);
19
20 component decoder
21 port (
22  A1,B1,A2,B2,G1N,G2N: in std_logic;
23  Y1,Y2: out std_logic_vector (0 to 3)
24  );
25
26  end component;
27
28 component PIPO
29 port (
30  D: in std_logic_vector (0 to 7);
31  clk,clrn :in std_logic;
32  Q: out std_logic_vector (0 to 7)
33 );
34  end component;
35
36  begin
37
38  dekoder: decoder port map (A1 => A1, B1 => B1, A2 => GND, B2 => GND, G1N => GN_ALL, G2N => GN_ALL,Y1 => CLK);
39  PIPO_A: PIPO port map (D => D, clk => CLK(0), clrn => CLRN, Q => QA);
40  PIPO_B: PIPO port map (D => D, clk => CLK(1), clrn => CLRN, Q => QB);
41  PIPO_C: PIPO port map (D => D, clk => CLK(2), clrn => CLRN, Q => QC);
42  PIPO_D: PIPO port map (D => D, clk => CLK(3), clrn => CLRN, Q => QD);
43
44  GN_ALL <= Zapisz;
45
46 end MAIN;
```

Rys. 8. Przykładowy kod w języku VHDL napisany w sposób strukturalny

(a) Deklaracja bibliotek i inicjalizacja środowiska. Ten fragment kodu znajduje się na początku każdego projektu wykonanego w języku VHDL. W pierwszej linii kodu znajduje się komenda deklaracji biblioteki **`library`**, która w tym przypadku użyta została do zadeklarowania biblioteki `IEEE`, która stanowi standardową bibliotekę wykorzystywaną w każdym projekcie VHDL. W drugiej linii kodu zastosowano komendę użycia **`use`**, którą wykorzystano do aktywacji pakietu `std_logic_1164` zawierającego wszystkie podstawowe funkcje i typy zmiennych potrzebne do projektowania w VHDL.

(b) Definicja bytu głównej jednostki projektowej. Po komendzie **`entity`** wprowadzamy nazwę bytu (`PIPO_MAIN`). Nazwa bytu odnosi się do opisywanego układu którym w tym wypadku ma być rejestr równoległo-równoległy (`P`arallel `I`nput `P`arallel `O`utput). Następnie przy użyciu komendy **`port`** definiujemy wszystkie wejścia i wyjścia projektowanego bytu. Definicję bytu kończymy nazwą bytu poprzedzoną komendą **`end`**.

(c) Definicja architektury rozpoczynająca opis logiki. Komenda **`architecture`** w 14 linijce kodu pozwala na stworzenie architektury o nazwie `MAIN`, która odpowiada za logikę bytu `PIPO_MAIN` zdefiniowanego w poprzednim podpunkcie. W architekturze zdefiniowano także sygnały przy użyciu komendy **`signal`**. Sygnały to dodatkowe, pomocnicze zmienne umożliwiające stworzenie połączeń między głównymi zmiennymi.

(d) Deklaracja komponentów przy użyciu komendy **`component`** w 20 i 28 linijce. Komponenty to układy, do których można odnieść się dowolną liczbę razy w układzie, tworzone są na podstawie oddzielnych plików w języku VHDL, w których opisana jest ich logika (są to oddzielne byty). W tym przypadku są to pliki widoczne w górnej listwie **Rys. 8.** (**pipo.vhd**, oraz **decoder.vhd**). Deklaracja komponentu zawiera nazwę bytu na podstawie którego komponent jest tworzony, oraz deklarację wejść i wyjść przy użyciu komendy **`port`**. Szczegółowy opis zadeklarowanych tu bytów ilustruje **Rys. 9.**

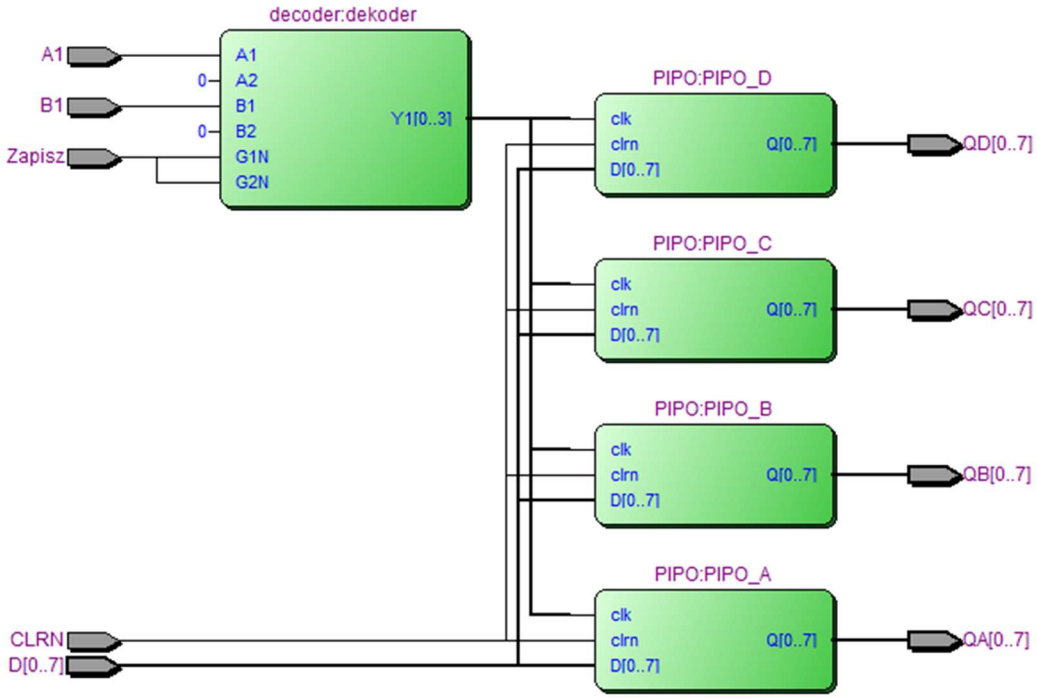
```
1   library IEEE;
2   use ieee.std_logic_1164.all;
3
4   entity decoder is
5   port (
6   A1,B1,A2,B2,G1N,G2N: in std_logic;
7   Y1,Y2: out std_logic_vector (0 to 3)
8   );
9
10  end decoder;
11  architecture decoder of decoder is
12
13  begin
14
15  process (A1,B1,G1N)
16  begin
17
18  if G1N = '1' then
19  Y1 <= (others =>'1');
20  elsif A1 ='0' and B1 ='0' then
21  Y1 <= (0 => '0', others => '1');
22  elsif A1 ='1' and B1 ='0' then
23  Y1 <= (1 => '0', others => '1');
24  elsif A1 ='0' and B1 ='1' then
25  Y1 <= (2 => '0', others => '1');
26  elsif A1 ='1' and B1 ='1' then
27  Y1 <= (3 => '0', others => '1');
28  end if;
29  end process;
30
31  process (A2,B2,G2N)
32  begin
33
34  if G2N = '1' then
35  Y2 <= (others =>'1');
36  elsif A2 ='0' and B2 ='0' then
37  Y2 <= (0 => '0', others => '1');
38  elsif A2 ='1' and B2 ='0' then
39  Y2 <= (1 => '0', others => '1');
40  elsif A2 ='0' and B2 ='1' then
41  Y2 <= (2 => '0', others => '1');
42  elsif A2 ='1' and B2 ='1' then
43  Y2 <= (3 => '0', others => '1');
44  end if;
45  end process;
46
47  end decoder;
```

```
1   library IEEE;
2   use ieee.std_logic_1164.all;
3
4   entity PIPO is
5   port (
6   D: in std_logic_vector (0 to 7);
7   clk,clrn :in std_logic;
8   Q: out std_logic_vector (0 to 7)
9   );
10  end PIPO;
11
12  architecture PIPO of PIPO is
13
14
15  begin
16
17  process (clk,clrn)
18  begin
19
20  if clrn ='0' then
21  Q <= (others => '0');
22  elsif (clk'event and clk ='1') then
23  Q <= D;
24
25  end if;
26  end process;
27  end PIPO;
28
```

Rys. **9**. Kod źródłowy opisujący byty dekoder oraz rejestr zawarte odpowiednio w plikach decoder.vhd i pipo.vhd

(e) Właściwa część architektury zawierająca logikę układu, część tę rozpoczyna komenda **`begin`**. W części tej odniesiono się do zdefiniowanych w poprzedniej części komponentów przy użyciu komendy **`port map`**. W 39 linii kodu stworzono instancję `PIPO_A` na bazie komponentu `PIPO` zdefiniowanego w 28 linii kodu. W nawiasie opisane są połączenia wejść i wyjść układu. W analogiczny sposób zdefiniowano instancję `PIPO_B`, PIPO_C, `PIPO_D`, oraz instancję `dekoder` na bazie komponentu `decoder`.

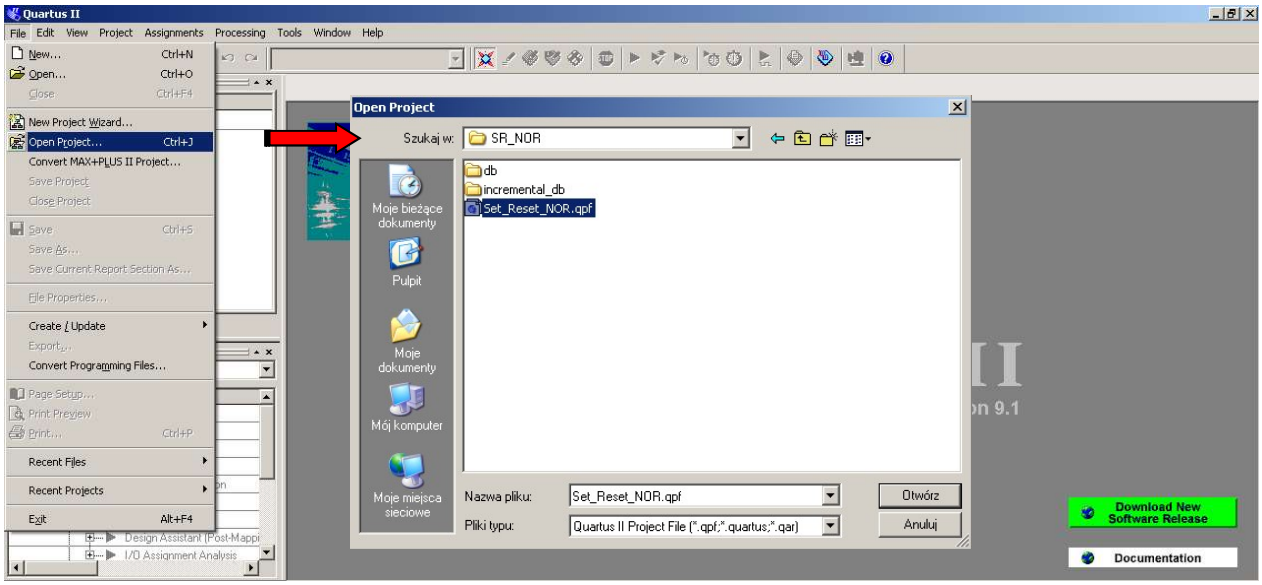
Podsumowując prezentowany projekt to układ do badania rejestru `PIPO`, który zawiera multiplekser, umożliwiający wybór wysterowywanego w danej chwili rejestru `PIPO`. W układzie znajdują się także cztery rejestry `PIPO`. Układ multipleksera zawiera w swojej strukturze dwa oddzielne multipleksery, jednak w celach pomiarowych użyty został tylko jeden z nich, wejścia `A2` i `B2` zostały zatem wyzerowane. Wejścia `A1` i `B1` służą do wyboru, na które z wyjść podany zostanie impuls. Każde wyjście multipleksera podłączone jest na wejście zegarowe jednego z rejestrów, co pozwala na zapisanie wartości podanych na wejścia D0 – D7 na wybrany rejestr. Układ został zaprojektowany w sposób strukturalny. Składa się z trzech jednostek projektowych zapisanych w trzech plikach VHDL. Plik **PIPO_MAIN.vhd** to główna jednostka projektowa spajająca dwie pozostałe jednostki w jeden układ. Plik **decoder.vhd** to jednostka projektowa dekodera opisująca go w sposób behawioralny. Plik **pipo.vhd** to jednostka projektowa rejestru PIPO opisująca go w sposób behawioralny, czyli opisu zachowania się układu bez wnikania w jego szczegóły implementacyjne na poziomie fizycznym. Sposób podłączenia poszczególnych instancji projektu można wygenerować posługując się narzędziem **RTL viewer** wchodzącym w skład narzędzi **Netlist Viever** (patrz **Rys. 3**). Efekt użycia tego narzędzia ilustruje **Rys. 10**.



Rys. 10. Schemat logiczny projektu

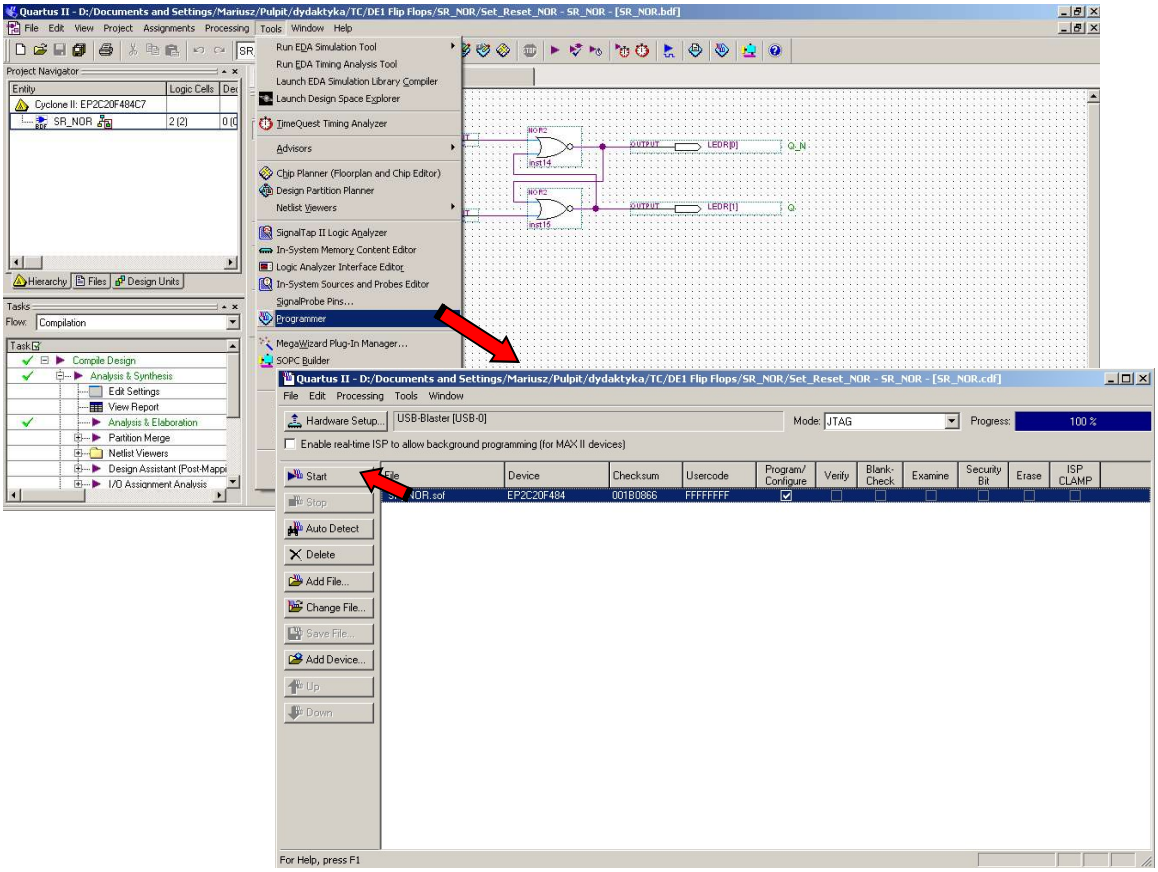
3. Uruchamianie gotowych projektów

Aby uruchomić gotowy projekt z menu wybierz polecenie **File|Open** Project a następnie wskaż plik projektu, który chcesz otworzyć tak jak pokazuje to **Rys. 11.**



Rys. 11. Otwieranie istniejącego projektu

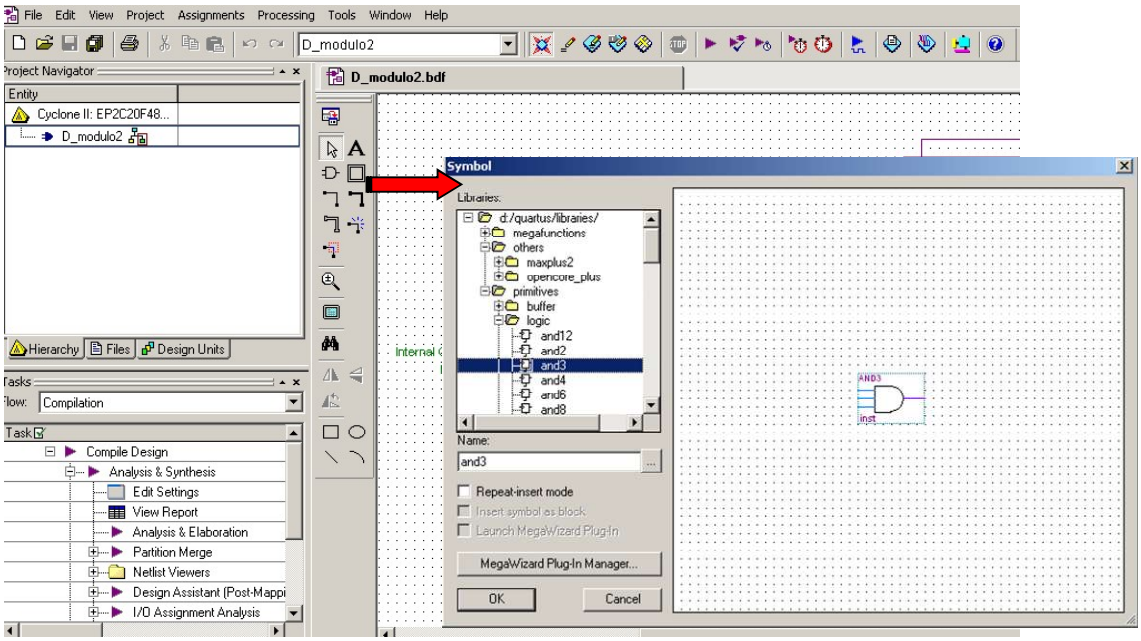
Po wczytaniu projektu wybierz z menu **Tools|Programmer** a następnie rozpocznij programowanie naciskając **Start**.



Rys.12. Przesyłanie projektu do realizacji w module DE1 (programowanie).

4. Modyfikacja schematu układu

Gotowe i działające projekty można modyfikować usuwając z niego wybrane elementy lub dodając nowe. W tym celu z listwy narzędziowej wybierz **Symbol Tool** (ikona z symbolem bramki AND) i rozwiń drzewo dostępnych elementów. Do wyboru mamy w tym momencie elementy biblioteczne oraz te, dla których zdefiniowano symbole w ramach projektu (**Rys. 13**).



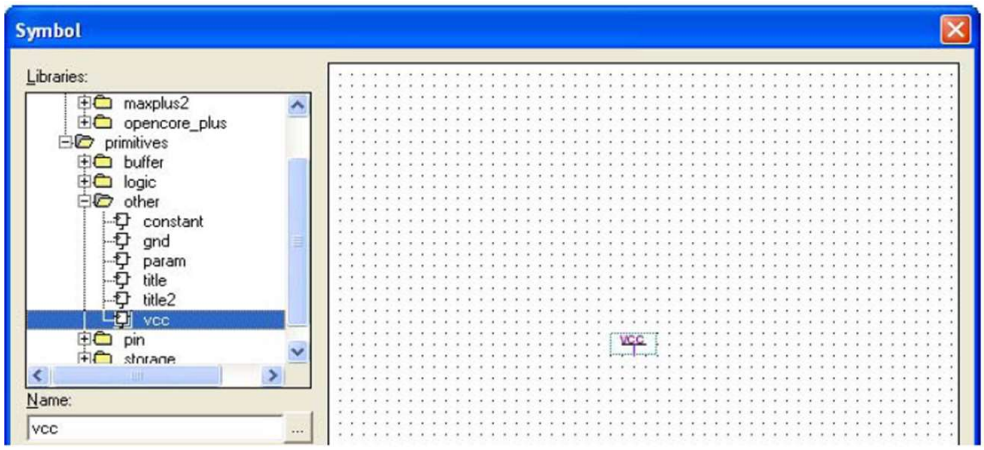
Rys.13. Dodawanie nowego elementu do schematu ideowego.

Aby umieścić wybrany element na schemacie naciśnij OK. a następnie kliknij lewym przyciskiem myszy w wybranym miejscu arkusza schematu. Aby wybrać inny element z biblioteki naciśnij strzałkę na listwie narzędziowej albo ESC i ponownie wejdź do biblioteki z elementami.

Aby narysować ścieżkę sygnału należy wybrać narzędzie **Orthogonal Node Tool** i ciągnąć myszką po arkuszu schematu. Podczas edycji można sterować powiększeniem wybierając z listwy narzędziowej lupę i klikając na schemacie lewym (powiększenie) lub prawym (pomniejszenie) przyciskiem myszki. Zapisz utworzony schemat poleceniem **File|Save.....** Upewnij się, że podczas zapisu zaznaczone jest pole **Add file to current project.**

Elementy umieszczone na schemacie muszą posiadać różne nazwy. Wybierz narzędzie do zaznaczania (strzałka) a potem dwukrotnie kliknij na symbolu układu scalonego. W oknie dialogowym wpisz unikalną nazwę. Alternatywnie, można dwukrotnie kliknąć na etykiecie elementu i edytować ją na schemacie. W podobny sposób nadaje się etykiety zaciskom.

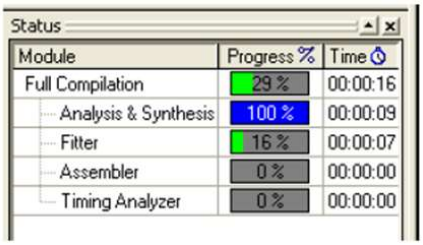
W przypadku potrzeby skorzystania ze stałych logicznych 0 i 1 możesz użyć elementów z biblioteki o nazwach *gnd* i *vcc*.



Rys.14. Dodawanie funkcji stałych z dostępnej biblioteki elementów.

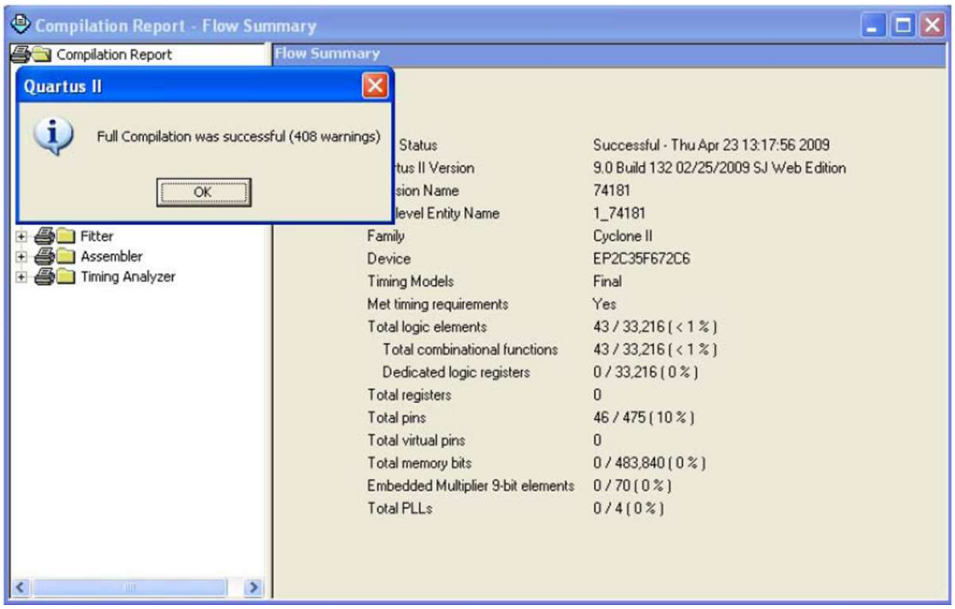
Po dokonaniu zmian w projekcie należy go bezwzględnie skompilować w tym celu wybierz polecenie **Processing| Start Compilation** Alternatywne sposoby uruchomienia kompilacji to skrót Ctrl+L lub ikonka z górnej listwy narzędziowej. Postępy kompilacji są pokazywane na bieżąco

(patrz **Rys.15**), dzięki czemu można śledzić jej przebieg, co ma istotne znaczenie zwłaszcza w sytuacji złożonych projektów.



Rys.15. Paski postępu kompilacji

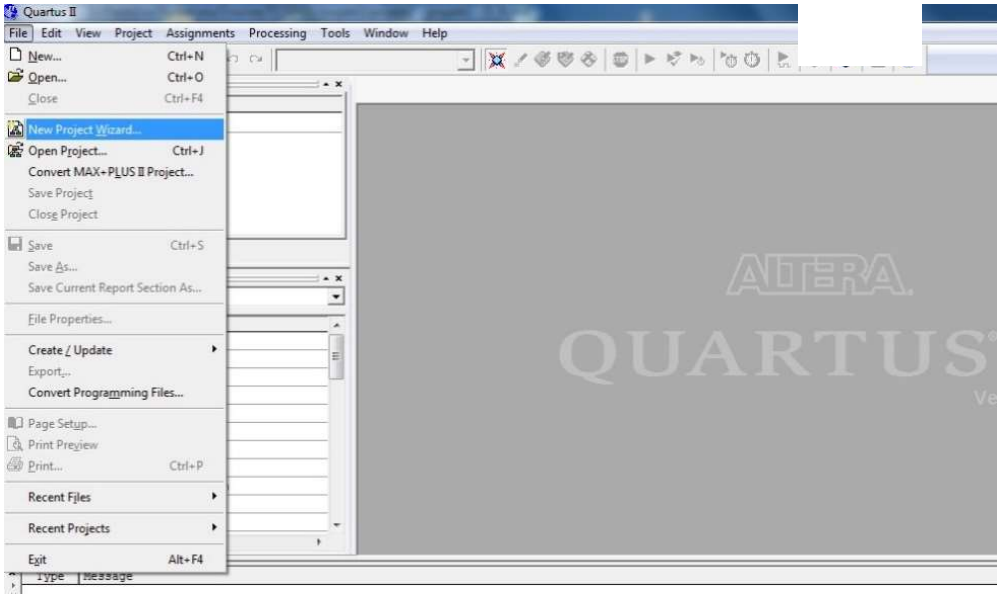
Pomyślne zakończenie kompilacji komunikuje ekran tak jak ten widoczny na **Rys. 16.**



Rys.16. Pomyślne zakończenie kompilacji

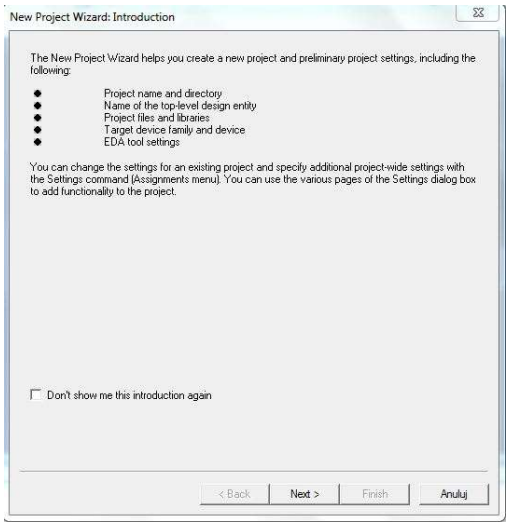
5. Tworzenie i uruchamianie nowego projektu

Aby utworzyć nowy projekt z menu wybierz polecenie **File|New Project Wizard** tak jak na **Rys. 17.**



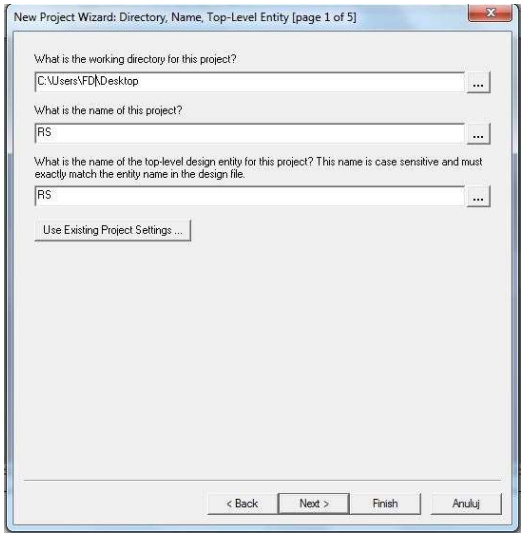
Rys.17. Tworzenie nowego projektu

Po zapoznaniu się z ekranem wstępnym projektu (patrz **Rys. 18**.) przechodzimy do kolejnego etapu wybierając opcję **Next.**



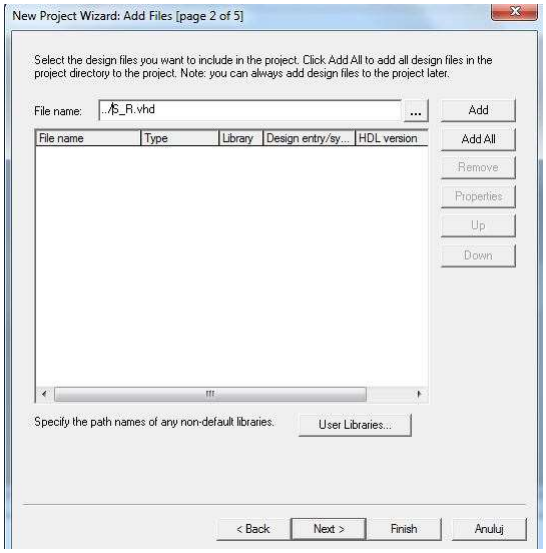
Rys.18. Ekran wstępny nowego projektu

W kolejnym etapie wybieramy katalogi robocze projektu i jego nazwę, tak jak na **Rys. 19.**



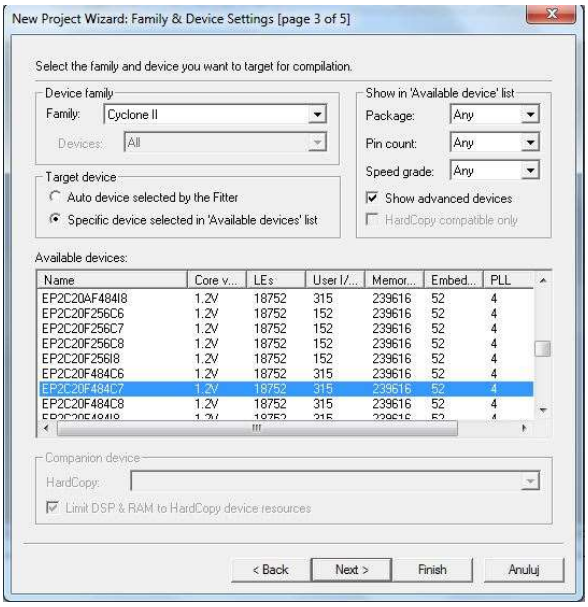
Rys.19. Ustalanie lokalizacji katalogów roboczych projektu i jego nazwy

W przypadku kiedy chcemy stworzyć projekt w całości od podstaw (*empty project*) wybieramy teraz opcję **Finish,** jeśli natomiast chcielibyśmy skonfigurować projekt z gotowych plików naciskamy **Next** i przechodzimy do kolejnego okna widocznego na **Rys. 20.**



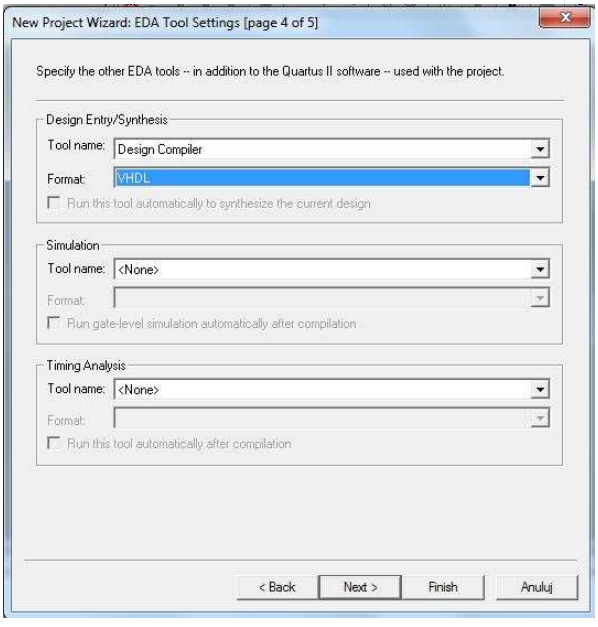
Rys. 20. Dołączanie plików do projektu

Okno to pozwala na dołączenie plików projektowych lub bibliotek, co realizujemy używając opcji **Add** (dla pojedynczego pliku) lub **Add All** dla wszystkich plików z danej lokalizacji. Tu znów możemy zakończyć tworzenie nowego projektu (opcja **Finish**) lub też przejść do dalszego etapu (**Next),** który ilustruje **Rys. 21.**



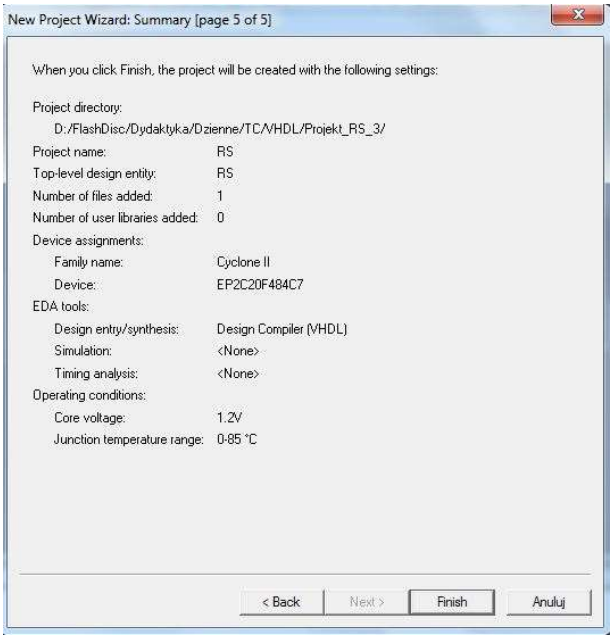
Rys. 21. Konfiguracja urządzenia dedykowanego dla projektu

W ramach tego kroku można skonfigurować urządzenie dedykowane dla tworzonego projektu. W przypadku kiedy używamy modułu Altera DE1 należy wybrać z listy urządzenie **EPC2C0F484C7** i zakończyć tworzenie projektu lub przejść do kolejnego etapu celem konfiguracji dodatkowych narzędzi projektowych. Okno dialogowe czwartego kroku umożliwia wybór narzędzi EDA (*ang. Electronic Design Automation*), które mogą być używane do zadań związanych z projektowaniem układów FPGA, takich jak symulacje, weryfikacje, analizy czasowe, generowanie raportów i wiele innych. Szczegółowy opis wspomnianych narzędzi znajduje się w dokumentacji technicznej oprogramowania Quartus II, która jest dostępna bezpośrednio na stronie internetowej producenta. Widok okna dialogowego przeznaczonego dla tej części tworzenia projektu ilustruje **Rys. 22.** W podstawowej wersji projektu jaki będziemy realizować na zajęciach laboratoryjnych wystarczy aktywować tylko opcję **Design Entry/Synthesis** i wybrać format **VHL**, co pozwoli na uruchomienie procesów analizy i syntezy projektu pod kątem plików VHDL. Pozostałe opcje pozostawiamy nieustawione i przechodzimy przy użyciu opcji **Next** do ostatniego kroku tworzenia projektu który przedstawia **Rys. 23.**



Rys. 22. Wybór narzędzi do analizy i syntezy projektu

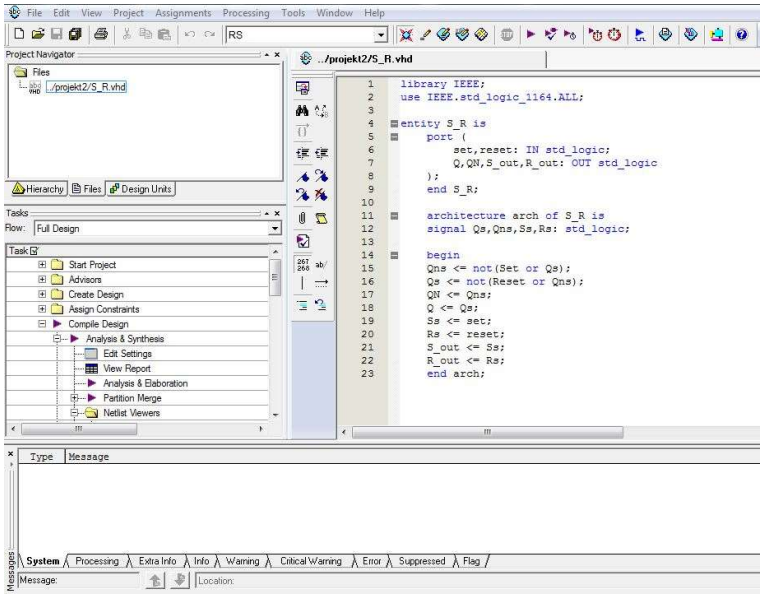
Widać tu podsumowanie stworzonego projektu i w przypadku kiedy podane informacje są zgodne z założeniami wstępnymi możemy zakończyć proces tworzenia nowego projektu wybierając opcję **Finish.**



Rys. 23. Podsumowanie projektu

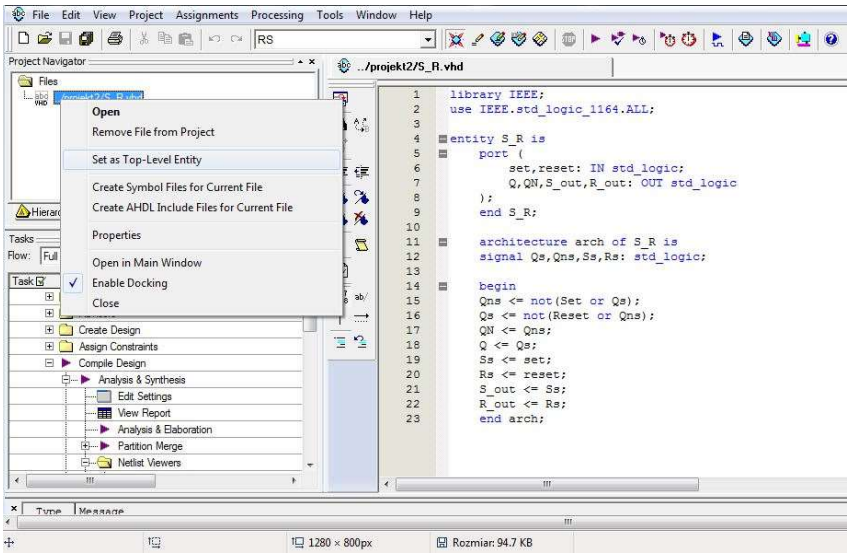
Efektem końcowym dotychczasowych prac opisanych w tym rozdziale może być projekt którego postać ilustruje **Rys 24**. Projekt ten składa się z jednego pliku o nazwie **S_R.vhd** znajdującego się w katalogu **…/projekt2**. Kod źródłowy w języku VHDL zawarty w powyższym pliku jest widoczny po prawej stronie obszaru roboczego w zakładce o tym samym tytule. W oknie zadań (**Tasks**) widać skonfigurowane narzędzia analizy i syntezy, natomiast w oknie wiadomości nie ma jeszcze żadnych komunikatów.

Aby uruchomić nowoutworzony projekt należy go skompilować (Patrz **Rys. 15-16**) wybierając ikonkę ▶ z górnej listwy narzędziowej lub polecenie **Processing | Start Compilation** lub też to skrót klawiszowy Ctrl+L. Czynność tą należy jednak poprzedzić ustawieniem hierarchii plików w projekcie wybierając tzw. plik nadrzędny (**Top entity**).



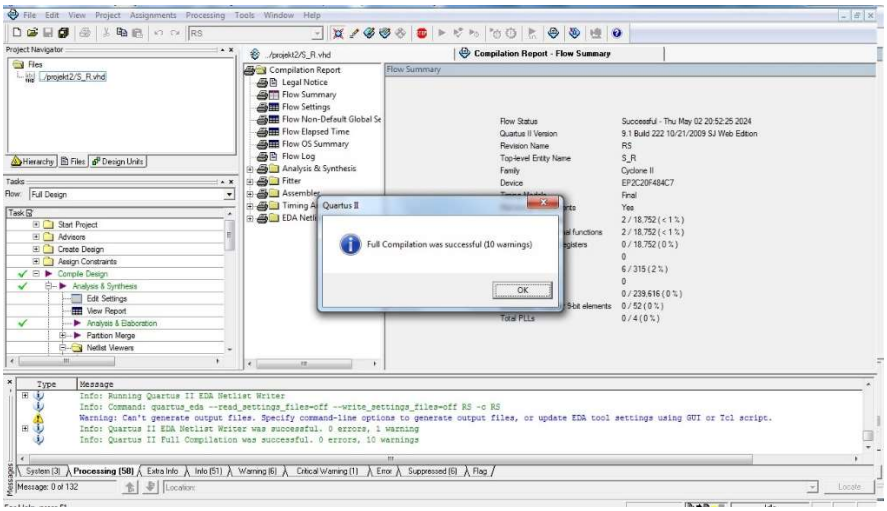
Rys. 24. Widok przykładowego projektu opartego o plik .vhd

Można to zrealizować poprzez menu podręczne wywoływane prawym przyciskiem myszy w sposób zilustrowany na **Rys. 25**. Wybieramy wtedy opcję **Set as Top-Level Entity.**



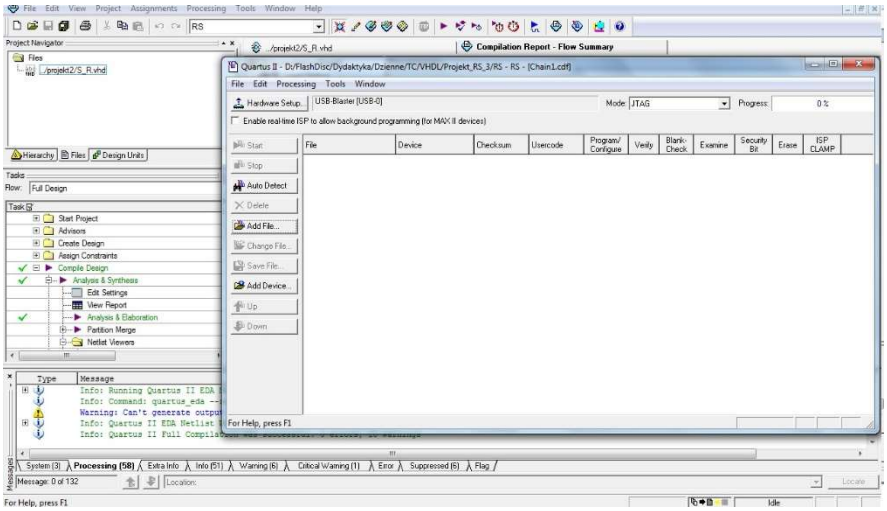
Rys. 25. Ustawienie nadrzędnego pliku projektu.

Po dokonaniu kompilacji otrzymamy raport tak jak na **Rys.26.**



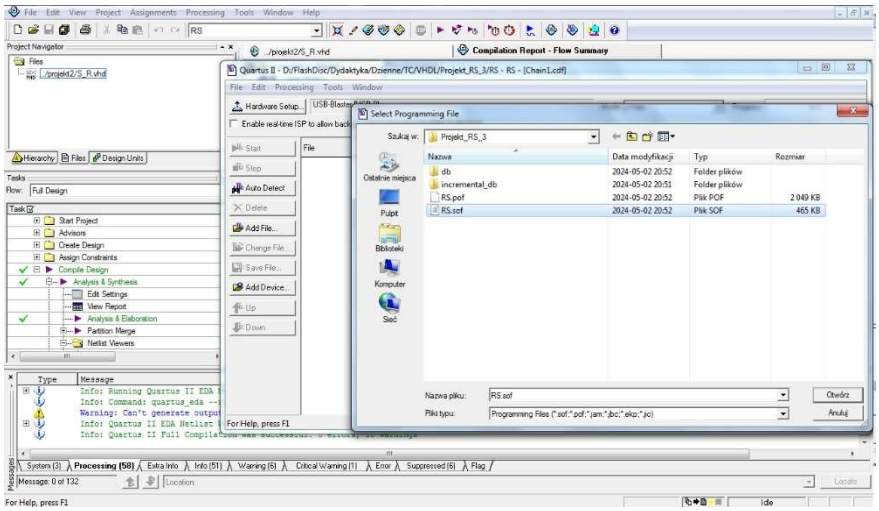
Rys. 26. Ustawienie nadrzędnego pliku projektu

Aby uruchomić nowoutworzony i skompilowany projekt należy teraz przejść do opcji **Tools|Programer** lub nacisnąć ikonkę  z górnej listwy narzędziowej na skutek czego, możemy otrzymujemy ekran jak na **Rys.27**



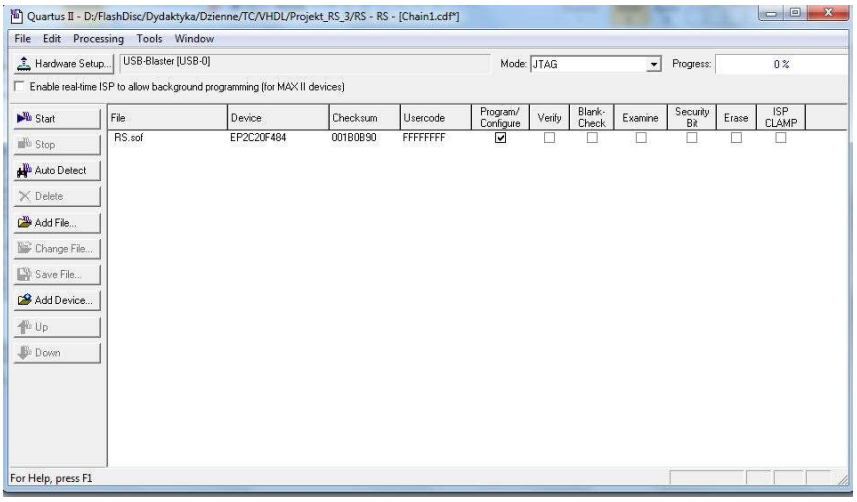
Rys. 27. Puste okno programowania modułu

Jak widać okno programatora nie zawiera żadnego pliku do konfiguracji modułu, należy więc go dodać używając opcji **Add File** w sposób pokazany na **Rys 28.**



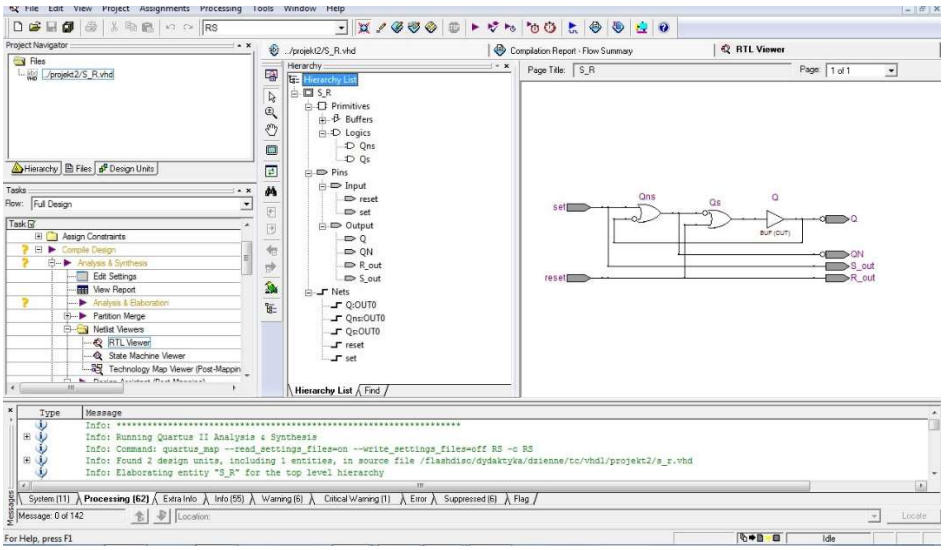
Rys. 28. Dodawanie pliku konfiguracji modułu do okna programowania

Poszukujemy pliku o nazwie takiej samej jak projekt z rozszerzeniem .**sof**. Można go znaleźć w tym samym katalogu co inne pliki powstałe po procesie kompilacji projektu. W efekcie dodania pliku konfiguracji modułu otrzymujemy widok okna programatora jak na **Rys. 29**



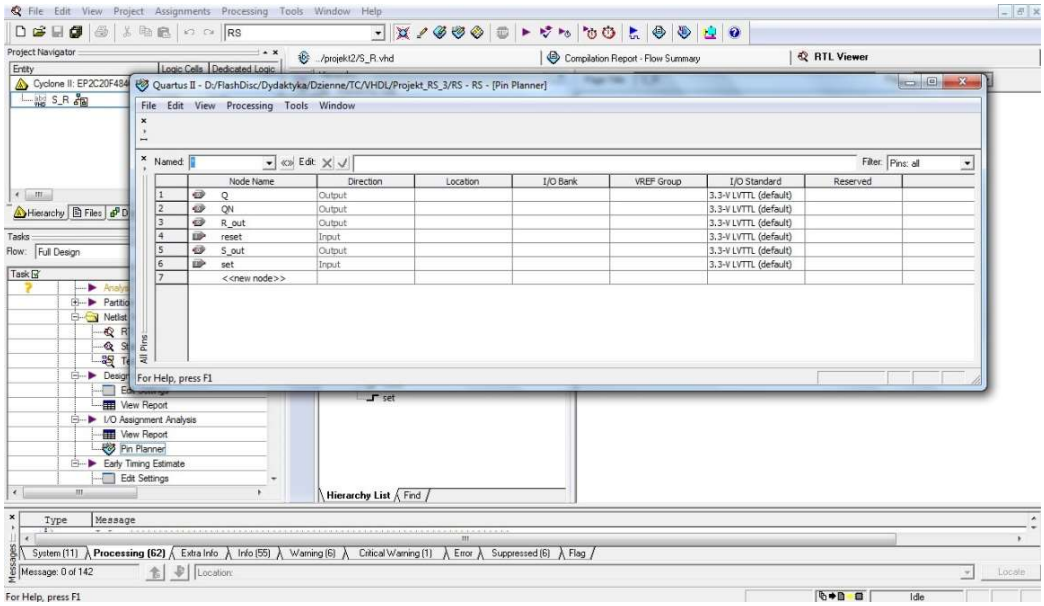
Rys. 29. Widok okna programowania po dodaniu pliku konfiguracji modułu.

Po zweryfikowaniu prawidłowości pliku (właściwa nazwa urządzenia (**Device)**, zaznaczona opcja (**Program/Configure**)) można zaprogramować moduł tak jak na **Rys. 12.** W rezultacie otrzymujemy działający sprzętowo projekt, którego strukturę możemy podglądnąć używając zakładki **Hierarchy** natomiast schemat logiczny korzystając z narzędzia **RTL Viever.** Obszar roboczy projektu po wykorzystaniu powyższy narzędzi może wyglądać tak jak na **Rys. 30.**



Rys. 30. Obszar roboczy projektu po uruchomieniu narzędzi RTL Viever i Herarchy

Można też podglądnąć konfigurację pinów modułu i ich połączenia do elementów schematu realizowanego projektu poprzez użycie narzędzia **Pin Planer** tak jak pokazuje to **Rys. 31.**



Rys. 31. Sprawdzanie konfiguracji pinów modułu z użyciem narzędzia Pin Planer

Jak widać wszystkie elementy schematu logicznego zostały prawidłowo skonfigurowane, jednak projekt nie używa dostępnych na płycie modułu elementów sterujących i sygnalizujących takich jak switch czy dioda LED, co można zmienić po przeprowadzeniu działań opisanych w kolejnym rozdziale.

6. Modyfikacja projektu VHDL

Chcąc dokonać modyfikacji takiego projektu należy określić przede wszystkim charakter i kierunek zmian. Zmiany te mogą dotyczyć zarówno struktury projektu (dołączenie kolejnych plików) jak też treści kodu źródłowego opisującego projektowany układ. W przypadku dokonania jakichkolwiek zmian należy zawsze ponownie skompilować projekt poprzedzając to ustawieniem hierarchii plików poprzez wybór tzw. pliku nadrzędnego (**Top Entity**).

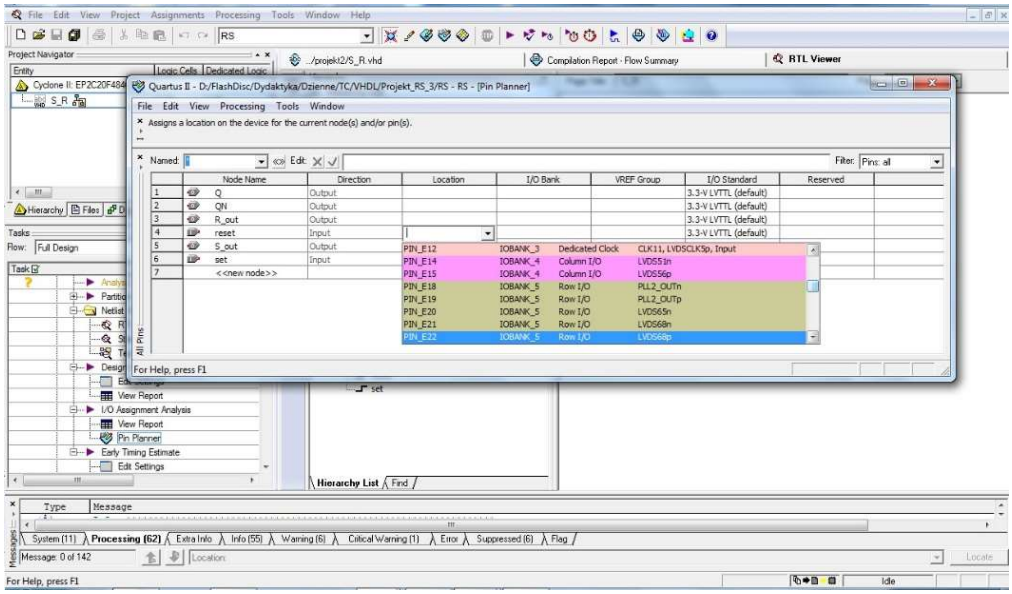
W sytuacji kiedy zamierzamy używać do sterowania układem projektowym dostępnych w ramach modułu przełączników (**switch**) lub innych elementów sterujących i sygnalizacyjnych (np. diody **LED**) należy je podłączyć i skonfigurować, co najlepiej wykonać z użyciem wspomnianego w poprzednim rozdziale narzędzia **Pin Planer**

Posługując się dokumentacją techniczną modułu Altera (DE1_UserManual), gdzie można znaleźć numery pinów wraz z ich opisem tak jak pokazuje to **Rys. 32,** należy wybrać pożądane elementy i odnosząc się do ich lokalizacji na płycie modułu podłączyć je do właściwych miejsc układu.

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_R22	Pushbutton[0]
KEY[1]	PIN_R21	Pushbutton[1]
KEY[2]	PIN_T22	Pushbutton[2]
KEY[3]	PIN_T21	Pushbutton[3]

Rys. 32. Wybrana tabela dokumentacji technicznej modułu DE1 Altera opisująca piny przycisków (KEY).

wyboru lokalizacji poszczególnych pinów ilustruje **Rys. 33.**



Rys. 33. Podłączanie pinów wybranych elementów modułu Altery do wybranych miejsc projektowanego układu.

Należy kliknąć w pole **Location** przy docelowym połączeniu (**Node Name**) i wybrać z listy rozwijanej właściwy element do konfiguracji. Po doborze wszystkich elementów projekt należy ponownie skompilować i przy braku błędów układ można zaprogramować.